PhD thesis: A designerly methodology for software development

Els Du Bois
University of Antwerp

TU Delft
Delft University of Technology

Universiteit Antwerpen

A designerly methodology for software development

Els Du Bois

A designerly methodology for software development

Els Du Bois

A designerly methodology for software development

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op donderdag 24 oktober 2013 om 10 uur

door Els DU BOIS
master in de productontwikkeling, Hogeschool Antwerpen
geboren te Antwerpen (Deurne), België

Dit proefschrift is goedgekeurd door de promotor
Prof. dr. I. Horváth

Copromotor: Dr. Ing. K.A.M. Van Doorsselaer

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof. dr. I. Horváth, Technische Universiteit Delft, promotor
Dr. Ing. K.A.M. Van Doorsselaer, Universiteit Antwerpen, België, copromotor
Prof. dr. D. Talaba, Universitatea Transilvania Brasov, Roemenië
Prof. dr. K. Váradi, Budapest University of Technology and Economics, Hongarije
Prof. dr. J. Verelst, Universiteit Antwerpen, België
Prof. dr. ir. J. Geraedts, Technische Universiteit Delft
Prof. dr. J.C. Brezet, Technische Universiteit Delft
Prof. dr. E. Giaccardi, Technische Universiteit Delft (reservelid)

Artesis University College Antwerp made important contributions to the work described in this dissertation

A designerly methodology for software development
Els Du Bois
PhD thesis, Delft University of Technology, Delft, The Netherlands
ISBN: 97890-6562-334-8
Published and distributed by VSSD, Delft, www.vssd.nl/hlf

elsdubois@gmail.com

Table of Contents

Chapter 1

Introduction

1.1. Setting the stage

1.1.1. Background of research: Trends influencing industrial product development

The knowledge platform of this promotion research is industrial product development, which has been strongly influenced by four major trends in the last decades. These trends are: (i) diversification of human and social needs, (ii) rapid development and uncontrolled proliferation of advanced technologies, (iii) evolution of product manifestations and implementations, and (iv) sophistication of design approaches and methodologies. Together, these trends have led to a shift in the approaches for realizing the products. The first trend, diversification of human and social needs, is associated with striving after better well-being. As a result of this trend, there is an intensification of consumer-oriented economy, as well as a continuous change in humans' needs. The different kinds of human needs and their relationship have been described by Maslow's model of hierarchy of needs [1]. Represented as a pyramid, this model includes five hierarchically arranged levels that build upon each other [2]. At the bottom of the hierarchy are the physiological needs, which are complemented by the safety, social, and esteem needs. The top-level expresses the self-actualization needs. To fulfill the higher level needs, large percentage of lower level needs should be fulfilled [3]. Designers usually face complex design tasks when they pursue the fulfillment of the higher levels of needs, in particular, when this challenge needs to be handled in constantly changing situations. For these reasons, designing products for the satisfaction of the higher-level needs entails new ways of thinking, novel strategies, and innovative concepts [4].

The second trend, 'rapid development and uncontrolled proliferation of advanced technologies', can be observed in many forms, for instance, it is reflected by the rapid developments in the electronics industry. Electrification started some 150 years ago. Some 90 years ago, complex electronic controller solutions were developed in the military industry. Seventy years ago, the concept of digital computing was introduced and rapidly proliferated in the industry. From the 1960s the emerging computer technologies have been complemented by advanced software and information processing technologies, advanced material technologies, and energy provisioning technologies. As a result of these, technological evolution has gained an even larger momentum. The evolution of technologies has had a strong influence on the manifestation as well as on the realization of the product. The nature of modern products is not anymore determined by their hardware part, but often by their embedded software components and knowledge bases. The range of physical hardware components has been extended with processors, sensors, actuators and transformer components [5]. Not only the variety and functionality of hardware components have changed drastically, but also their scale and integration level [6]. Dominant change has been miniaturization, which leads to micro- and nano-scale components. The software technologies have also been rapidly developing both in terms of their enabling algorithms and in terms of their information/knowledge contents.

The third trend, 'observable evolution of product manifestations and implementation' is fuelled by the above developments of technologies that have contributed to the formation of new product paradigms, which in turn lend themselves to completely different material, energy, and information flows in modern products, such as cyber-physical consumer durables and services. The emerging new product paradigms not only imply a sophistication of products, but also change the meaning of products. According to the current knowledge, three generations of products can be identified - see Figure 1.1. The first-generation products are assembled hardware products, software implementations, and pure services. The second-generation products show a growing level of integration and move towards integrated systems. They are instantiated by three types of systems: (i) product-service systems, (ii) embedded systems, and (iii) information systems. The third-generation products are complex systems and environments, which are conceptualized and implemented according to the principles of cyber-physical systems [7].

Therefore, third generation products are often referred to as cyber-physical consumer durables and services. They involve high level interaction with people, their embedding environment, and other products, or alternatively, they may work autonomously and adaptively. Other infrastructural systems with resembling functionality and implementation technologies are called 'the Internet of things' or self-contained systems

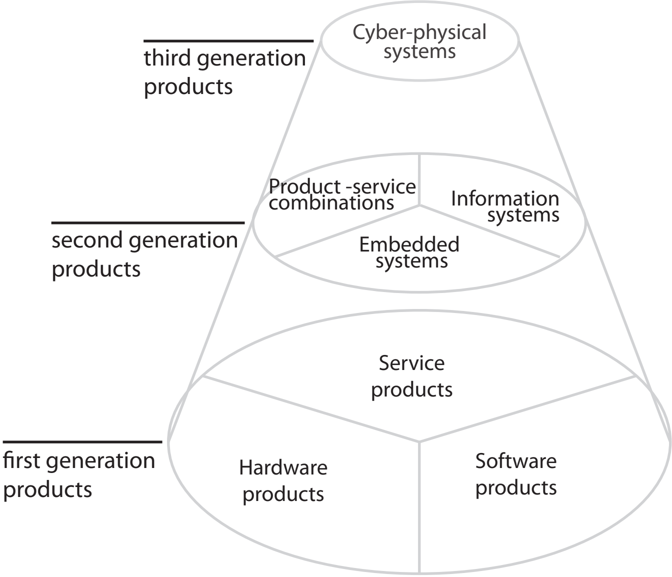


Figure 1.1. ***Visualization of the evolution of product development manifestations***

as 'complex adaptive systems', but these names also indicate some notional and conceptual differences. The third generation high-end products are highly complex, decentralized, open, adaptive, intelligent, or even evolving. Typical features of these cyber-physical systems are strong multidisciplinary and penetration into human social and cognitive domains [8, 9]. These systems brought affront different forms of human-system interactions. For the sake of completeness, we must note that a new generation of products does not fully replace the older generations, but coexist with them. It means that, at a given point in time, different generations of products can be seen on the market.

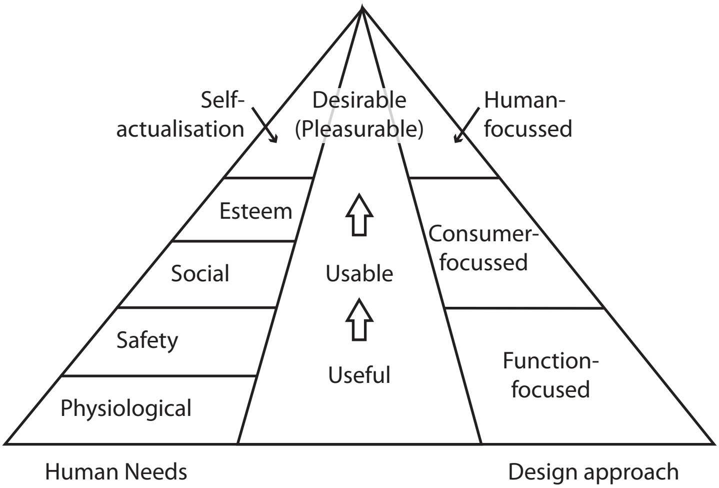


Figure 1.2. A conceptual model on the evolution of design concerns (based on [3])

The above mentioned three trends jointly lend themselves to a fourth trend. This trend concerns the change in design approaches, methodologies and technologies. These are becoming more sophisticated due to the broadening of the domain of opportunities (the affordances of technologies) and to the demand of fulfilling new customer needs and increased user-expectations [10, 11]. The influence of this trend is also reflected by the shift in the attention of designers. Namely, their attention is shifting from pure form, function, materials and manner of production to utility, the human experience of usability, and desirability concerns [12]. What it means is that in addition to form, function, and materialization, the meaning of the products is also becoming an important phenomenon for designers. The shift in design approaches from function-focused ones through consumer-oriented ones to human centered ones is an essential development and this gave motivation for, and played an important role in this promotion research. We observed that the progression indicated by the above trends can be blended with Maslow's model of human needs. We have extended the coverage of this model with the above discussed phases of change in the focus of designing and with the varying design approaches [4]. As shown in Figure 1.2., this compound conceptual model expresses all of the concerns that industrial design engineers should address when designing competitive products. Based on Figure 1.2, the objective of this promotion research is to address the issues of the top layer of the extended Maslow model, which incorporates the needs for self-actualization, desirability and human-centered approach.

1.1.2. Trend of emerging and proliferating of human centeredness in design

Considering the above-mentioned trends of design, we define design (thinking) as: “a human-centered approach to innovation that draws from the designer’s toolkit to develop products that fulfill the needs of people, integrate the possibilities of technology, and incorporate the requirements for business success.” ~ Tim Brown, president of IDEO. In the past, product development was mainly focused on products and objects, especially technological objects. An although the focus on products remains, there is a remarkable evolution in the last 20 years in design [13]. What has changed in our understanding of the problem of design knowledge is a greater recognition of the extent to which products are situated in the lives of individuals, in the society and culture. We are concerned with the experience that human beings have on products: how they interact with products and how they use products as a mediating influence in their interactions with other people and their social and natural environments. The evolution brought up a big challenge for designers, especially due to the changed scope. Design thinking was stimulating the realization of different design methodologies that were developed to improve creative efficiency of designing and extend design to other areas of practice. Starting from the mid-1980s, it was a race to discover new methods for improving business, service and design. Before focusing on the main approaches, we have to note that there was no clear linear progression of methodologies that arose, as many were developed at the same time in different faculties and industries. Considering the chronological advancements in the major procedural trends in design, we identified four important evolutionary steps: (i) participatory design, (ii) user-centered design, (iii) usage centered design, and (iv) human-centered design. Although these terms are often used as synonyms, they each have their unique characteristics, as shown in Table 1.1. In the next Sections we will further elaborate on the process of evolution

Table 1.1. Evolutionary steps towards human-centered design (based on [13])

Participatory design	⇨ **User-centered design**	⇨ **Usage-centered / service design**	⇨ **Human-centered design**
focus on product functionality/utility	focus on the user - usability	focus on the usage - journey + experience	focus on humans - empathy
user testing	user experience	user journey	society - environment
efficiency	needs	value	understanding
end-user development	user at center of development	stakeholder culture	holistic community development
/	/	improve	empower
evolved as method for user-involvement in design	evolved as method of consulting with users	evolved as method of empathic design for observation of usage	/

towards these steps and their characteristics.

This evolution toward an increased human centeredness in the design had a direct influence on other aspects of the design, especially on the product complexity and on the design process complexity [10]. The product complexity increased as a result of the fulfilling of the high-level needs [15]. Often complex solutions are needed to achieve the objective. Since these needs are focused on individuals or small groups that are constantly changing, the products must show evolvability that also increases the complexity. Process complexity was mainly caused by the increased multi-disciplinarity of product design, due to the technological evolution and to the new disciplines that emerged from the human centeredness, such as societal design, service design, (web) communication design, and environmental design, sustainable design. Design and realization of complex products need a proper combination of disciplinary experts, working together according to shared objectives.

1.1.3. Historical overview of human centeredness in design practices

Industrial design always cared for "human needs", but it caters for different needs in different contexts [16]. Before the 1950s, design mainly focused on functions. The development of functional products, graphics, and interiors was the core of what both practitioners and clients assumed design was about. In the mentioned time, typically, it was investigated what people wanted or how things worked, were used, maintained, and disposed. From 1950s to 1980s, design began to be consumer focused. In the late 1960s and 1970s an important new element entered the design repertoire: human factors, or as it was known in Europe, ergonomics, which grew mainly out of the need during World War II to adapt the design of complex military systems to the physical and cognitive capabilities of operators. Designers discovered that these disciplines could make products, services, environments, and communications more usable and useful. Much design, such as home furnishings, continues to focus on aesthetics. After 1990s, design paid more attention to the different levels of human needs. Nowadays, it is not enough if a product's function and usability are well thought out. It should also meet higher-levels of human needs. The needs of self-actualization comprising cognitive, aesthetic, self-actualization and self-transcendence aspects reveal the tendencies for future design. Future design will satisfy a wide range of human needs, even subtle needs which users have not recognized. The core of today's human-centered innovation, is about balancing between human needs, functionality, marketability, usability, and sustainability. The external look became only one of the dimensions among many in the complex interactions by which people discover, understand, learn, and adopt artifacts and construct the meaning of products by using them.

Participatory design

In the 1960s, participatory design was gaining momentum through research in the framework of the design methods movement. Dubbed the Scandinavian approach, participatory design was about integrating end-users into the development (prototyping) phase of projects [17]. Technological developments during the end of this decade caused a shift in the participatory design objective, from a social method to a technological method. In the 1980s, participatory design became synonymous with the emerging field of interaction design. Many of the techniques used in participatory design were borrowed from science, such as usability testing. Other techniques included mock-ups, prototyping and even role playing. Nevertheless, there were some critical disadvantages, for instance negligence towards user experience and stakeholder input. Usability was the most important aspect, but emotional response to gadgetry was largely ignored. In many instances user testing was abandoned, when user decisions conflicted with those of the stakeholders and the designers [14].

User-centered design

In response to the end-user dilemma of participatory design, discussions concerning co-design (cooperative design) or collaborative design began to take place. This alternative method aimed to transform passive users into cooperative 'designers'. The most significant contribution to the transformation of user development in design was introduced by design theorist Donald Norman in 1987 [18]. He re-defined participatory design into what he coined as user-centered design [19]. User testing became less about usability and more about users' interests and needs. Norman tried to stimulate user-control and humanized participatory and system design by "making things visible"[20]. This was to ensure that users could discover errors and have control over resolving them. Another objective of the move from participatory to user-centered design was to place the user at the center of the development process. It highlighted the benefits of understanding user experience over user testing. Based upon behavioral sciences, user-centered design emphasized experience over efficiency and adopted a more humanistic approach with the involvement of the user throughout the development of a product or system [21]. User-centered design grew out of speculations towards elevating users from guinea-pigs to co-developers of systems. This new methodology is widely spread in the industry and practice.

Usage-centered design

Usage-centered design was introduced by Larry Constantine and Lucy Lockwood in 1999 [22]. Usage centered design evolved as a software engineering alternative to user-centered design. Usage-centered design is a systematic, model-driven approach to user interface engineering for software and web-based applications, which puts the focus on user intentions and usage patterns [23]. Usage-centered design can be distinguished clearly from the more widely recognized and practiced user-centered design. As the name suggests, users are not in the center of attention but usage, namely the tasks intended by users and how these are accomplished. Consequently, utility, functionality, usability, and usefulness are more important than users, and supporting effective user performance is more important than promoting good user experience [24]. When designers concentrate on creating good user experience, they often fail to support the performance of the users. Usage-centered design analyzes users in terms of the roles they play, in relation to systems and employs use cases for task analysis. It derives visual and interaction design from abstract prototypes based on the understanding of user roles and task cases. Beginning with early work on task modeling based on use cases, it has evolved into a sophisticated process that has proved itself on projects of widely varying scope and scale in a variety of application areas.

Service design and metadesign

In the literature, in a similar direction, two parallel domains evolved based upon user-centered design. In interface engineering, usage-centered design appeared which is discussed above. In addition, service design emerged to be a design discipline in the early 2000s. We found that the developments of participatory design, user-centered design, and the evolution of customer experiences all have shaped the basis of service design. In the review of [14], development of service design is interpreted as: ‘‘[it] Draws on several traditions including product, environment, experience and interaction design” [25]. It was argued that the distinction between a service and product becomes irrelevant, for everything could be interpreted as a type of service that supports the value creation [26]. Service design aims to understand how and what the user does with a product (or service), including their journey and experience. Rather than thinking about end-user experience of a product or service (user-centered design) attention has shifted to understanding the usage, the interaction and journey of the product/service after it has left the hands of the provider. Another important aspect is the holistic perspective of service design. Instead of focusing only on end users, service design seeks to collaborate with all stakeholders. Consequently, service design focusses on building relationships among all stakeholders and supporting communication for the exchange and development of value and knowledge.

Human-centered design

Since the 1990s, the terms ‘human-centered design’ (HCD) and ‘user-centered design’ (UCD) were often used interchangeably, regarding the integration of end users within a design process. We consider HCD as a broader concept and strategy than UCD. Human-centered design became a concern in technological and product system industries and was especially growing in the domain of human-centered interaction. HCD was further specialized to cover the roles of all stakeholders in complex systems, enhancing human abilities, aid to overcome human limitations and foster user acceptance [27]. In its final (and current) phase of evolution, HCD is considered beneficial for resolving wider societal issues. The broad holistic perspective introduced in service design allowed for human-centered design to [28]: (i) aggregate knowledge with stakeholders, (ii) achieve validation with peers in feedback systems, and (iii) involve all stakeholders in a participatory design process [17, 29].

1.2. Research domain and problem

In this research project, we focus on knowledge exploration and synthesis for methodology development. The main research problem addressed is conceptualization and implementation of a methodology to support realization of design software tools by which designers can develop complex second and third generation products. As an implication of the interacting trends, some specific requirements should be considered. The methodology has to be: (i) procedurally structured, (ii) human-centered, (iii) adapted to thinking of designers, and (iv) supporting the designers with relevant specific methods, instruments and techniques. The close relationships between products and their users entail the need for a more intense (multiple) stakeholder involvement during the design process. This enables designers to understand the needs for change and to cope with the challenges of complex functionalities and fast realization processes in a context-dependent manner.

As the literature shows, several generic methodologies have in the past been developed without considering the specificities of concrete applications. However, we presumed that, in case of a concurrent elaboration, we have the opportunity to implement a kind of 'reflexive practice', or, in other words, to follow an approach that allows fine tuning the methodology to representative applications and achieving efficiency through practical experiences. However, as explained below, only one complex reference case could be developed in this promotion research project due to capacity limitations. The novelty of the reported research approach is in the epistemological, methodological and procedural symbiosis of the methodology development and the development of the application case. It was assumed and has been confirmed that the dialectic interaction of the support tool and the application case provides benefits for both.

To further detail the problem domain, we examined in Sub-Section 1.2.1 the second and third generation products in more detail. In addition, we detail the current practice of human-centered design in consumer product development in Sub-Section 1.2.2. Because the domain of product development is too broad, we narrowed our focus in Sub-section 1.2.3. and we considered the current practice of human centeredness in this focused domain of software development in Sub-Section 1.2.4.

1.2.1. Second and third generation products

In this Section, we discuss the third trend that was introduced in Section 1.1. To explain how the separation of product generations was achieved, we based on a literature review in which we found three authors who proposed explanation for a kind of stratified interpretation of manifestation of design in a widening context. In 1999, Buchanan introduced four orders of design [12], in 2005, Van Patter and Pastor introduced four fundamental shifts in design [30, 31]. In 2012, Wassermann discussed how products grew from 'stuff' to socio-technical systems, identifying 4 design generations [32]. As shown in Table 1.2, we built further upon these authors' stratifications and identified four product generations, which are a result of the evolution.

What has also been mentioned above is that current methodologies are mostly focusing on first generation products. Consequently, in this promotion research we are only focusing on second and third generation products. As shown in Table 1.3, the second and third generation products differ from first generation products. In this table, a comparison of their characteristics is given. In general, they have a higher complexity, as they are combining the physical, cyber, and service domains and are consequently comprised of many components

Table 1.2. Overview of the separation of product generations

Buchanan	Van Patter (Humantific) (NextD)	Wasserman	Product generations
Symbolic and visual communications	Design 1.0: traditional design	Design 0.0: designer artisan	(handcraft products, everything before industrialization of products)
Material objects and things	Design 2.0: products and services	Design 1.0: making and selling stuff: industrial aesthetics	First generation level: products, services and software
Activities and services	Design 3.0: organizational transformation design	Design 2.0: human centered innovation: field building and embedding	Second generation level: product-service combinations, information systems and embedded systems
Complex systems and environments	Design 4.0: transformation design	Design 3.0: changing the world	Third generation level: CPS and complex systems

and stakeholders. On the other hand, there is a higher diversification in products and today's products are better adjusted to the users' needs, even if the needs are constantly changing. This is realized through a higher intangibility, a higher user interaction and interaction with environment, using a range of devices, sensors, controllers and other ubiquitous technologies, and an increased information transportation, collection, and transformation.

1.2.2. Current practice of human-centered design in product development

It is our understanding that human-centered design (HCD) is at the core of consumer product development (PD) practice. We discuss the current practice of HCD in PD by its basic principles: (i) front-loaded, (ii) interdisciplinary, dynamic and creative teamwork, (iii) balance between configuration and verification, and the use of methods en systematic approaches [33, 34]. At the start of the development process, the knowledge is rather limited, but also the cost of effort, time, and resources that are needed to introduce changes. Moreover, the most important decisions are made. Efficient product development aims to test the ideas, concepts, and systems as early as possible in the process to check the usefulness, usability, desirability, added value, diversity, and feasibility, and checks if the design and solutions respond to the objectives and specifications [35]. Especially due to the high fuzziness and undefined problem, stakeholder involvement is important as early as possible, and to clarify the problem domain [36].

Due to the complexity, there is a need for people with various expertise to work- together in the development team [37]: technology (mechanics, electronics, algorithms, computation

Table 1.3. Characteristics of the different product generation levels

Characteristics	Generation 1	Generation 2	Generation 3
Focus	Things	People	Society
Design practice	Artifact - centric	User-centric	Socio - centric
Added value	Basic needs	Psychological needs	Self-fulfillment needs
Complexity	Low complexity →	→	High complexity
complexity of challenge	Products, and service messes	Organizational messes	Societal messes
Challenge scale	Small scale →	→	Large scale
fuzziness of challenge	Product, service, experience challenges	Systems, organizations, industry challenges	Economic, society, and planet challenges
Problem challenges	Defined challenges →	→	Undefined challenges
Process start	Design brief →	→	Fuzzy problem
Process end	Final solution →	→	Evolutionary product
Number of stakeholders	Few stakeholders →	→	Many stakeholders
Type of product - interaction	Utility interaction →	Emotional interaction →	Cognitive and motoric interaction
Team	Single designer / small team →	→	Multidisciplinary team
Process	Sequential	Parallel	Web
Tools	Existing tools →	Need for new tools and methods	
New design practices and disciplines that emerge to fulfill the needs of each generation	Communications Marketing Human factors Service design Product design Hardware design Software design	Systems design Entrepreneurship Corporate strategies Design language Design research Interdisciplinary design Co-design New media Interaction design Experiences design Brands Eco-design	Sustainable development Social innovation Public policy Future scenarios National innovation strategies International field innovation

and data processing…), economy (strategy, marketing, market research, stakeholder identification, positioning…), human (usability, ergonomics, anthropometry, psychology, sociology…) [38]. The advantages of working interdisciplinary or in multidisciplinary team is that it endeavors better understanding of stakeholders' interests and motivations and a better conversion and implementation in products and systems. Collaboration exists in teams with external stakeholders such as suppliers, clients, and other designers. Teamwork and communication becomes challenging as teams become bigger, more diverse (i.e. using different ontologies) and temporary.

Regarding human centeredness in the methods and systematic approaches, we can conclude, from our literature review, the following: The design process is characterized by divergent and convergent steps. These enable stakeholder involvement in two different manners and with two different aims: (i) generation of data and investigation of problems, and (ii) idea selection, decision making and change proposals [39]. Furthermore, the development process is split in different phases, which increases the manageability and control. Different processes of product development can be found in literature [40-43]. In general, the three most important phases, where stakeholder involvement is beneficial, are: idea generation, concept design and detailed design.

1.2.3. Narrowing the research domain to software products

In this research, we concentrated on the development of software as products or components of complex systems. This problem domain was chosen because software (i) yields the largest opportunities of meeting the requirements rooted in of complexity and evolvability, and (ii) has a large influence on the sophistication of products, but (iii) is also the most difficult part to develop in complex systems (see Figure 1.3).

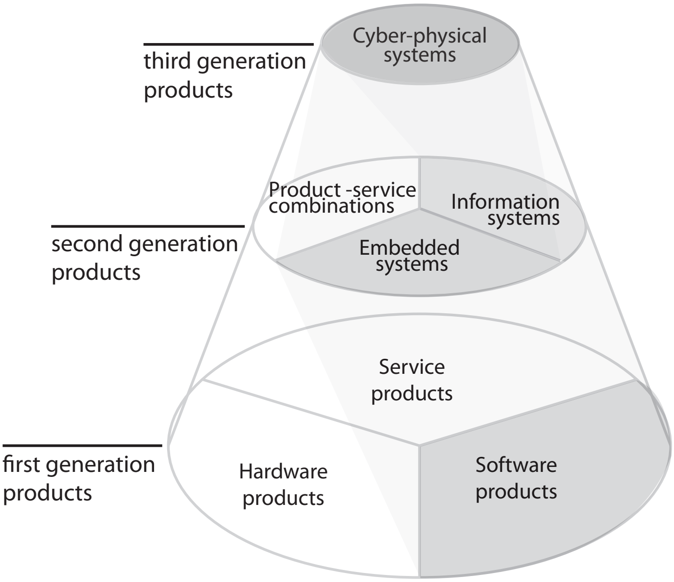


Figure 1.3. Evolution of products and growing importance of the domain of software development

In the last decades, there was an intense diversification of software products and this process continues even now. Software products manifest in many forms, e.g. as self-contained application packages, embedded software for controlling systems, agents of complex information systems, or synergetic constituents of cyber-physical systems. In Figure 1.4, an overview is given of the different types of software products [44-46]. Software applications are no longer merely tools for professional instrumental productivity, but also (re)constitute and mediate different social structures and practices as a result of personal content production. This kind of user-generated content integrates words, pictures, videos and audio into human-technology interaction with the aim of sharing stories within a certain virtual community. In order to be able to develop these software products efficiently and effectively, current human-centered (participatory) software development is lagging behind [47].

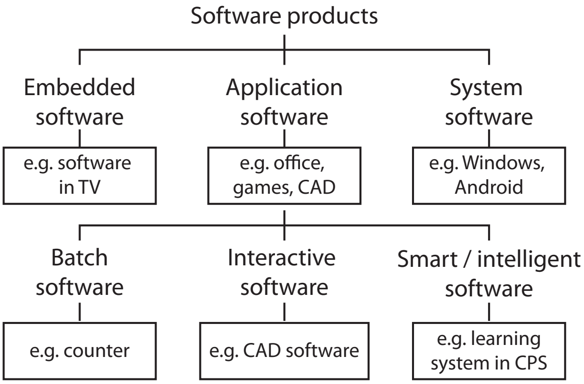


Figure 1.4. Types of software products

From the range of software systems, we only focus on interactive software applications in this promotion research. This category of software has the aim to process data under the control of human users. This form of software operation is typical in design support tools for which there is a growing need in the industry. For this reason, we have decided to specifically focus on interactive and knowledge-intensive design support tool that can facilitate concept generation and trade-off forecasting in case of ubiquitous augmentation of domestic appliances. A software tool providing the necessary functionality for this application was selected as a test case for our human-centered software development methodology. This test case also plays the role of an archetype of a family of similar design support tools. By using it as a reference case, we could consider a family of design support tools in our work and grasp a range of technical and human issues associated with a dedicated software development methodology. Interactive software applications, in particular, application-focused design software tools, are used by designer who expect the software tool to (i) support their thinking and creation processes, (ii) allow large freedom in conceptualization and investigation of solution concepts, and (iii) to process dynamically changing real-time data, while (iv) also allowing easy and effective interaction and data/knowledge retrieval and management. As a consequence of these expectations, the development of this family of software tools needs an intense stakeholder involvement.

1.2.4. Current practice of human centeredness in software development

In this Subsection, we will give a brief overview of the current practice in software development, and of the current stakeholder-oriented software development approaches. It is widely accepted that users should be involved in software development [28, 34, 48-51]. Involving the end users and learning their real needs is proven to be beneficial.

Spending time in the product environment to understand user requirements is an important prerequisite for sound design practice, irrespective of the design approach or philosophy. Since there is often no trigger to use a user-centered approach, many projects plunge too quickly into software design and construction. As described by [52]: the result is the illusion of progress (“we’re in the first week and we are already coding!”) purchased at the price of premature commitment to particular solutions that invariably compromise utility and usability (“too late to fix that, it’s already hard coded). Consequently, there is a duality: in contrast to the recognition of the importance of stakeholder involvement, many authors mention that involving the users is often difficult and quite rare in the practice of software development (SD) organizations, especially in the design phase [53, 54].

Methodologies and processes of current software development

Typically, a software development lifecycle includes various stages from preliminary development analysis to post-development software testing and evaluation. To handle these activities, several software development methodologies are used use today, i.e. sequential, incremental, evolutionary, agile…. [55]. Some companies have their own customized software development methodology but the majority uses traditional or agile methodologies (as shown in Figure 1.5). Traditional methodologies, also known as heavyweight methodologies or plan-based development [55], support designers using comprehensive planning, detailed documentation, and structured methods. These methodologies, such as waterfall, V-model and the Rational Unified Process, are intended for large-scale projects, involving multiple systems, for whom the detailed approaches and offered control methods are crucial. However, emergent changes later in the development process have a large cost.

The agile software development (ASD) or lightweight methodologies, in contrast to traditional approaches, employ short iterative cycles, and rely on tacit knowledge, existing informally within a team, as opposed to documentation. In literature, different comparisons can be found of these two major methodologies [56-60]. ASD is a philosophy or a way of thinking about software development and there is no unified agile methodology to follow. ASD refers to a number of different iterative and incremental software development methodologies, such as extreme programming [61], scrum [62], and feature-driven development [63]. that share common principles and practices [64]. These principles are bounded in the agile manifesto, which focusses on the development process in a human-centric manner. The four key characteristics are: individuals and interactions over processes and tools, working software over

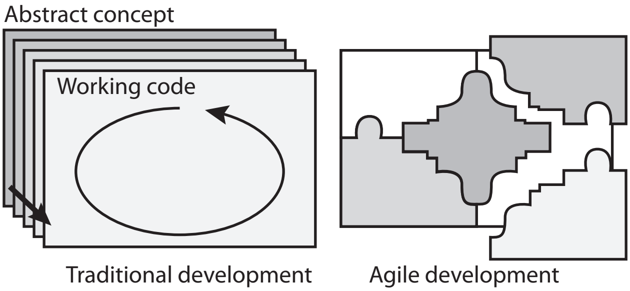


Figure 1.5. Difference between the traditional and the agile approach

comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [65]. ASD have short release periods, are flexible, require minimal documentation, rely on individuals, and use self-organizing teams. However ASD does not promote a prescriptive process, it is incremental as the system is developed (and released) in small parts. In a perfect world, ASD would not be iterative, as iterating completed functionality means rework and waste of resources [66].

At its core, agile and traditional are based on similar values [67]: doing a good job, leading a team, and delivering measurable results. Nevertheless, some project management professionals may discard the principles of ASD, if they are unable to accept all its components and practices. In contrast, as it is stated in the agile manifesto: “while there is value in the items on the right, we value the items on the left more” [65]. In industry, successful agilists use a number of activities, tasks, and deliverables that are not called ‘pure agile’ [60]. This mixing and adjusting of software development process elements from agile and systematic approaches is a much more practical way of using these methods [57, 59]. Nevertheless, while it has been argued that agile methods are compatible with traditional disciplined processes, actual project experience indicates conflicts can arise [58].

Stakeholder involvement in software development

While both approaches are potentially beneficial to software development, they both don’t cover a whole stakeholder-oriented approach. After investigating the literature for stakeholder involvement approaches in traditional software development, we can conclude that not involving users is still one of the main problems. In order to tackle this problem, different user-centered design (UCD) methods were developed, which were intended to get integrated into, and to work as a sub-process for, any traditional software development methodology. Many different concepts can be found in literature related to these approaches of stakeholder-oriented software development, each having their own focus or interpretation. The most common ones are [18, 52, 68, 69]: User-centered design (UCD), human-centered design (HCD), user-centered systems development (UCSD) , User experience (UX), usability [70], human-computer interaction (HCI), interaction design [71], goal-oriented design or usage-oriented design, cooperative design, participatory design, co-design, contextual design, and user involvement [50]. They fall under the general category of human-centered design, but have all different flavors [51, 72]. To summarize, these HCD methods for traditional development approaches only deal with the user research and the design and evaluation of the user interfaces. Agile methodologies on the other hand seem to forget end-users and usability altogether in practice.

Examining agile software development approaches, we noticed that user-centered design and agile development share some common aspects, but also have differences in philosophy and practice. Agile software development from a HCD perspective has qualities that can provide a solid foundation for user-centered attitude: focus on people, communication, customer collaboration, adaptive processes and customer/user needs. However, Agile development cannot be considered to be user-centered as its values do not have the necessary focus on users and usability: some of the agile processes' prioritized areas of interest can prevent a user-centered attitude: a focus on programming and programmers, automated tests, very short iterations and fast increments, and executable software as a measure [66]. Other problem areas are the confusion between users and customers, unsatisfactory techniques for modeling users and tasks (i.e. user stories and use cases), the fear of early design as well as insufficient activities for interaction design. It seems that the production of working software at quick and constant pace provides a great setting for usability evaluations, but the traditional usability testing conducted in a laboratory hardly fits this process. While there is little time to do usability testing, there are many discount methods that can be used in agile development. Scheduling and reporting on usability studies need both to be reconsidered in agile development [51].

We also searched for reasons as to why involving the users is often difficult and rare in software development organizations. There isn't much critique represented against HCD, but the most common arguments are that it can cost a lot and take a long time to do slowing down the development process. One of the great usability myths is that usability is just common sense. The biggest problem is that, since it is vaguely defined, it can be applied in a variety of ways. This may lead to poor quality and poor usability of the product and misconceptions about the effectiveness of HCD. Although standards are clarifying the user-centered process, it is too abstract to be integrated into an existing software development process as such. Constantine and Lockwood state that although the three main HCD techniques (user studies, rapid prototyping, and usability testing) are useful, they still are not substitutes for good design [73]. They further state that: user studies easily confuse what users want with what they really need; rapid iterative prototyping is often a sloppy substitute for systematic design; and that usability testing is often an inefficient way to find problem that could have been avoided through proper design.

1.3. Needs for stakeholder involvement

We conclude based on the preceding discussion that stakeholder involvement in software development has in practice a somewhat negative flavor, we experience that less creative methods are used, and that stakeholders are not involved during the development process, but before or after. The traditional human-centered design methods do not go beyond typical customer research and consider the overall utility and the design and evaluation of the user interfaces. On the other hand, the currently known agile methodologies overlook the end-users and the usability aspects altogether in practice. In this research we should take over the benefits from both agile and traditional development, i.e. especially the flexibility and the systematic approach, and optimize the needed stakeholder involvement

The aim of this PhD research was to support the software development process towards the development of complex human-centered software systems or components, with the objective of making systems more successful. This need for human centeredness in the design process emerges in different perspectives: (i) team perspective (dealing with (large) multi-disciplinary teams), (ii) process perspective (intense stakeholder involvement), and

(iii) product perspective (design of software is inseparable of design of human activities). The handling of these perspectives individually and in combination is challenging as they usually involve many different types of stakeholders, such as end-users, suppliers, clients, marketers, management, knowledge experts, and IT maintenance expert, who are involved in different phases (specification, algorithm development, coding and production, distribution, usage, maintenance, etc.) of the product life-cycle, and context of the system. On the other hand, there are no general rules for optimal stakeholder involvement, since it always depends on the concrete cases.

1.3.1. Team perspective

As explained above, new generation products feature an increased complexity, which is a critical issue in the development of large software-intensive systems. Complexity may appear in multiple forms, such as functional, structural, computational, technological, cognitive, application and usage. In the overall process of product development, we can separate: system development and software development. System development is concerned with the development of the whole system. The result of systems development are documents that describe the system architecture and the functions and connections of the system functionality. Software development is concerned with the development of software and knowledge components. The result of the software development are software artifacts and knowledge contents [74]. Consequently, teams or even multiple multidisciplinary teams are needed for the development of complex systems [38]. Often, however the teams responsible for software and hardware development are not harmonized [75]. This situation leads to misunderstandings and impedance mismatches in the developed artifacts. Usually, interdisciplinary teams are needed for component development, as they should blend aspects as that of the programmers, designers, architects, psychologists, economics, or other domain-specific experts [76].

1.3.2. Development process and product perspective

Nowadays, the common understanding of innovation builds on the observation that firms rarely innovate alone and that the innovation processes include interactive relationship between producers, users, and many other different actors [72, 77]. Software development is a knowledge-intensive work where different stakeholders should exploit their existing knowledge and create new knowledge to find the best product solution with an optimal product-user interaction. To support concern-based, knowledge-intensive software development, the software products should expose the knowledge related to specific concerns of stakeholders and allow embedding the necessary knowledge in the software means. This implies an extensive knowledge aggregation, representation and sharing activities due to the involvement of different kinds of stakeholders and large heterogeneous repositories of knowledge [78]. Though it is widely accepted that users should be involved in the interactive systems development, most frequently, this is not happening optimally in software development organizations [54]. Stakeholder-oriented software development has substantial economic and social benefits [51]: This strategy (i) saves development costs and time, reduces maintenance costs, and redesign costs; (ii) decreases the need for customer/user support, (iii) is easier to understand and use, thus reduces the training and support costs, (iv) reduces discomfort and stress, improves user satisfaction, increases ease of learning, and trust in systems, (v) improves the productivity of users, and reduces user errors, (vi) produces financial benefits due to increased sales and leads; (vi) improved product quality, appeal to users, and avoiding litigation (by taking care of product safety), and (vii) results in benefits for in-house development.

1.3.3. Concluding remarks

Experiences showed that involving stakeholders in multiple phases of the software development process has many benefits. However, it has to be seen as a trade-off issue because of organizational and financial overheads. Based on the preliminary analysis, we can hypothesize that there is a need for both methodological frameworks and for instrumental enablers that allow effective human-centered and participatory software design approaches. The development of complex software systems is a challenging design activity. The process is difficult "not only because of the complexity of the technical problems, but also because of the social interactions that take place when users and system developers learn to create, develop and express their ideas and visions" [79]. Designing complex software systems is an intrinsically collaborative process, which raises the need for synthesizing the different stakeholders' reasoning. The major challenge for software technologies of the future is to provide support for achieving a shared understanding among groups of people that see the world in fundamentally different ways [80].

1.4. Research vision and main objectives

This PhD research strongly envisioned that software tools belonging to the category of interactive application software (e.g. design support tools) should be developed according to a participatory design strategy. To our opinion, human-centered design of interactive software has often not reached the desired and possible level, compared to the case of many consumer hardware products. The research vision was that a methodology was needed to solve the mentioned problem of interactive software development. As a research problem this poses two challenges: (i) re-conceptualization of the development process of interactive software towards a designerly (stakeholder-oriented) approach, and (ii) establishing a robust basis for a new methodology that covers the early phases of software development where critical decisions are made. Our primary objective was not increasing the efficiency of the product development, but increasing the utility and quality of interactive software products. By involving the stakeholders in the early phases, software products can be made more customized and better fitting the needs [81]. Despite the additional time and efforts needed, utility and quality enhancement of software is worth involving the stakeholders. Obviously, the stakeholders have to be involved in the most critical points of the process, and in order to achieve a significant impact, some reconceptualization of the process is deemed to be necessary. As widely known, the most critical decision points are in the fuzzy front end and in the conceptualization phases of software design, though, typically, those decisions that are made in the implementation phase can neither be neglected. Considering these facts, we hypothesized that a combination of different single-phase methodologies are needed to provide effective support to every particular phase and to the whole software development.

Consequently, the objective of PhD research was set to conceptualize, elaborate and test a designerly software development methodology (DSDM) that supports stakeholder involvement in the most critical phases of software design. We decided to apply a structured view on the software development process and introduced a methodological framing by which we could focus on the subsequent phases. It is our belief that stakeholder involvement has to start when the design requirements are to be identified and when an overall conceptual framework of the software tool is constructed. Stakeholders should also be involved when the concept of the software tool has been developed (it should be demonstrated to stakeholders and justified and validated through their involvement). Finally, stakeholders should be involved when a pre-implementation version is completed and take part in the testing and critiques. To complete these activities efficiently, the above phases need dedicated methodologies that we called single-phase (component) methodologies. They were coherently and transitively integrated into the targeted multi-phase support methodology, called DSDM.

1.5. Research hypothesis and assumptions

Focusing on humans and their experiences is a key-issue in current product development. Our generic research hypothesis suggests that software development could benefit from following the principles of human centeredness that have been applied in traditional product development. Based on our forerunning literature study and practical experiences, we investigated the differences between the development of hardware and software products. Furthermore, we have investigated why we cannot directly use the human-centered design principles of consumer durables to software development. Our research hypothesis also claims that specific methodological principles gathered from the domain of modern consumer durable development could be used as a basis of the targeted designerly software development methodology. We define the word designerly as “based on the principles of designing consumer products”. A graphical illustration of our hypothesis is shown in Figure 1.6. It has a broader relevance than just to the area of interactive design support tools – its claims can in principle be extended to the domain of cyber-physical systems too.

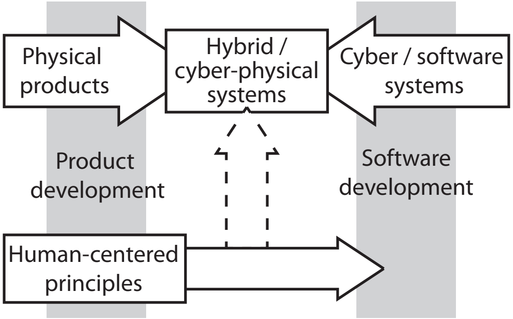


Figure 1.6. Visualization of the main hypothesis

The above hypothesis rests on the assumptions that traditional way of consumer durables design offers useful design principles and that they can be taken over to the development of interactive software products [82]. These assumptions seemed to be defendable for the reason that there is an extensive literature on the principles and approaches of human centeredness in consumer durables design, where optimal physical and cognitive interaction with humans is an important factor of the success of products on the markets and in applications. In this domain, designers have a strong intention to customize the product to end users [83]. Towards this end, they closely involve and interact with various stakeholders in the development process. The stakeholder involvement is supported by the use of various demonstration means, visuals, and virtual and physical prototypes, such as sketches, mock ups, CAD models, and tangible prototypes [81]. In the most decision-intensive parts of the design process, prototypes are used to discuss and evaluate the design with stakeholders [84]. Verifications and validations happen in different phases of the development process and consequently different means are used. Taking over the relevant principles of consumer durables development to the domain of interactive design software development is however not straightforward. There are some important differences between the two domains. The most significant ones are: (i) the difference in the tangibility or material manifestations of products, which entail different prototyping means, and (ii) the difference in the interaction with the physical product and software products. It seems that it is more difficult (and time consuming) to make concrete early demonstrations of intangible products and consequently, they require a higher ability from stakeholders to internalize and empathize with the design and to be able to provide suggestions for improvements. Often companies do not want to spend more time on testing and prototyping early in the process. However, this extra time is returned as the product is optimized and should not be revised late in the development process.

1.6. Research objectives

1.6.1. General objective

The objective of the PhD research was to increase the stakeholders' involvement in the software development process using the principles of ordinary product development. To be concrete, stakeholders can be everyone who is involved in the development, distribution, usage, maintenance, and context of the system. At the end of this PhD research we wanted to have a designerly software development methodology that supports stakeholder involvement in the most critical phases of software design (= main hypothesis). Behind this needed methodology, we identified two scientific problems that should be solved: (i) we need a reconceptualization of the software development process to increase stakeholder-involvement, and (ii) extra enablers must be developed to achieve higher efficiency. We note that efficiency is in this research not towards a shorter time-frame nor less bugs, but towards better adapted products being more user-centered. We also had to consider that integrating new methods into established work practices is difficult and therefore the introduction of new complicated methods and means often fails [85].

1.6.2. Specific objectives of the single phase methodologies

Because of the difference in the nature and characteristics of the phases, different single phase methodologies are needed to be included and integrated into the multi-phase DSDM. These most critical phases are: (i) ideation and framework development, (ii) concept development, and (iii) system development or detailed design. During the development of software products, stakeholder involvement should start with the identification of the requirements and the framework development. Afterwards, the concept of the software should be presented to stakeholders and verified. Lastly stakeholders should be involved in the pre-implementation activities of design support tool development through using testable tangible prototypes. The objective of the various parts of this research was to study the context of the demarcated development phases (framework and requirement ideation, concept integration, and system development) from a designer's perspective, with the aim to convert/apply the obtained knowledge in software development, and to develop and test each of the single-phase methodologies.

1.6.3. Objective of the reference case development

In the discussion on the research domain and problem, we argued about the necessity of developing a reference application case parallel with the multi-phase software development methodology. The very reason was that, at the time of developing an execution plan for the research project, we also realized that a methodology development cannot be separated from the definition of the family of application cases that it is intended to support. We realized the practical advantages of considering some concrete reference cases from the very beginning of the development of the designerly software development methodology. Consequently, decision has been made to elaborate and learn from a reference case already in the conceptualization phase, but also in the implementation phase of the multi-phase methodology. The co-development of the methodology and the reference case resulted in a co-evolution during the research process. Ideally, multiple cases should have been developed and investigated, but due to time and capacity limitations, we had to make a compromise on conducting a single-case study. On the other hand, in defining this reference case, we had in mind that this particular reference case should be a representative of a family of relevant application cases. We believe that this traversal (intertwined) development of the DSDM with the reference case did not impose any limitation on the obtained results. Contrarily it not only introduced a novelty in the conduct of the research, but offered the opportunity for an in-process concept and construct validation. This novelty came from the fact that the DSDM coupled with the reference case was used as an evolving research means in the research cycles which were framed as design inclusive research.

For this promotion research, the type of software was defined by a recognized real-life need, namely, the need for interactive and knowledge-intensive design support tool that can facilitate the concept generation and trade-off forecasting in case of ubiquitous augmentation of domestic appliances. A software tool with the necessary functionality was selected as a test case for a human-centered software development methodology.

This also played the role of an archetype of similar design support tools. By using it as a reference case, we could consider a family of design support tools in our work and grasp a range of technical and human issues associated with a dedicated software development methodology. Interactive software applications, in particular, application-focused design software tools, are used by designers who expect the software tool to support their thinking and creation processes, to allow large freedom in conceptualization and investigation of solution concepts, and to process dynamically changing real-time data, while also allowing easy and effective interaction and data/knowledge retrieval and management. As a consequence of these expectations, the development of this family of software tools needs an intense stakeholder involvement.

A typical example of highly interactive software applications are the various applications of design software tool, such as CAD, CAE, DFX, and CBR systems. These software products are strongly contextualized and process-related to be able to seamlessly support designers. The success of these products depends on how much they are adapted or adjusted to the way of working and thinking of designers, and how much they fit their natural way of thinking and doing. The selected reference case is a software tool for smart energy saving. This highly-interactive design support tool is intended to support designers in their decision making processes on smart energy saving using ubiquitous controllers. This case was selected because, to be able to support the software development process, stakeholder involvement was crucial. The specific aim of the tool development is to support the designers in this thinking process by offering them structural and functional information and trade-off calculations. The conceptual basis of the software tool is not a composition of algorithms, but the decision making process and mental reasoning of designers. The highly interactive nature of the considered design tool required a high amount of action-related and decision-making knowledge. An optimal development of this kind of software tool projects ahead the need for participatory conceptualization and design, in which the end-user (designer) is not the only stakeholder. Software developers and administrators of the software, as well as the concerned various knowledge engineers (such as energy saving experts and controller device suppliers) should be involved in the software development process as well.

1.7. Generic assumptions and implications

To briefly summarize here, the following operative and content oriented assumptions were taken into consideration in the variously focused cycles of research. Note that these assumptions concern (i) the objective of the research and the related software development methodology, (ii) the research approach and the work done, and (iii) the reference case. Figure 1.7 shows the assumptions together with their implications on the research.

Main assumption 1:

> We state that an increased intensive stakeholder involvement in software development can be achieved by means of using the analogies of ordinary product development means.

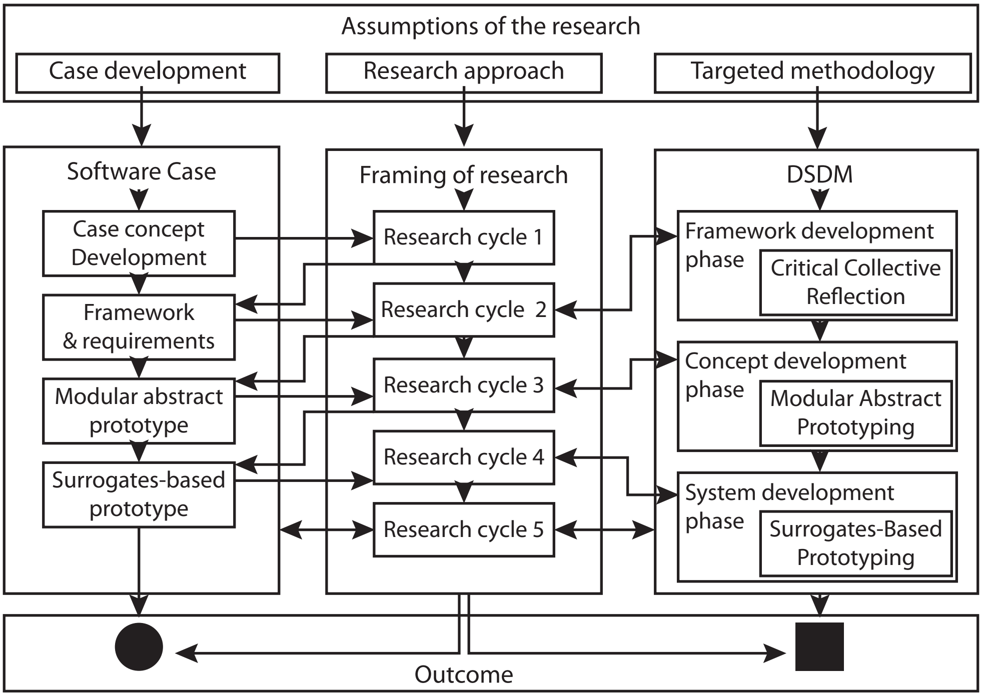


Figure 1.7. Overview of the work done in the promotion research (middle) together with the related methodology development (right) and the reference case development (left)

Implication 1:

We assume that there is a need for a methodology that supports this intense stakeholder involvement.

Implication 2:

The targeted methodology should be a multiphase methodology focusing both on decision making in the early ideation and concept integration phase, and on testing of both the concept and the implementation of the software.

Implication 3:

Towards a structured procedure, three phases were assumed to be important: (i) ideation and framework development, (ii) concept development, and (iii) system development or detailed design.

Main assumption 2:

The research should be broken down into different research cycles, each of them having specific objectives and done in specific contexts.

Implication 4:

The different research cycles were supposed to allow investigation of each phase of the development process separately and the development and testing of the entire multi-phase methodology and its different constructs.

Main assumption 3:

A reference case can facilitate a reflexive practice and in-process verification and validation in the research project.

Implication 5:

Without specific application domain the methodology and reference case cannot be derived, tested, and improved.

Implication 6:

The outcome of the research should be instrumental is towards both (i) a generic SD methodology, and (ii) a generalizable case.

1.8. Overall research approach

Due to the varying of objectives and contexts, a multi-methodological framing was applied to set up the research design. The whole of the PhD research was broken down into five interrelated research cycles (RC x), as shown in Figure 1.8. Each cycle had its own objectives, context, and framing methodology [86]. For this purpose, the methodological framing theory, proposed by Horváth [87, 88], has been applied . The objective of the framing of the research cycles was to streamline the research activities towards the specific research objectives and to take care of the investigation of the research context in the specific phases of the idealized multi-phase process (ideation and framework aggregation, concept development, and system elaboration). The investigation happened from the perspective of designers, with the aim to convert/apply participatory design principles in software development, and to develop and test a practical stakeholder-sensitive single-phase methodology for each phase.

In the first research cycle, we investigated the need for stakeholder involvement in the current software development approaches, and described the context of the research process. We analyzed the phenomena of stakeholder-oriented design, and considered the gaps and important issues to deal with in our methodology. During the execution of RC 2, 3 and 4, we investigated the three most critical phases discussed above. In research cycle 2 we examined the issue of methodology development in the context of requirements engineering and framework ideation. In research cycle 3, the context of contextualization and concept testing were considered and the influencing factors of enabling concept synthesis and demonstration were investigated. In research cycle 4, the research work focused on developing surrogate-based prototyping in the context of detailing functionality and usability testing. In research cycle 5, we concluded about the entire research through a multi-aspect external validation of the proposed multi-phase methodology.

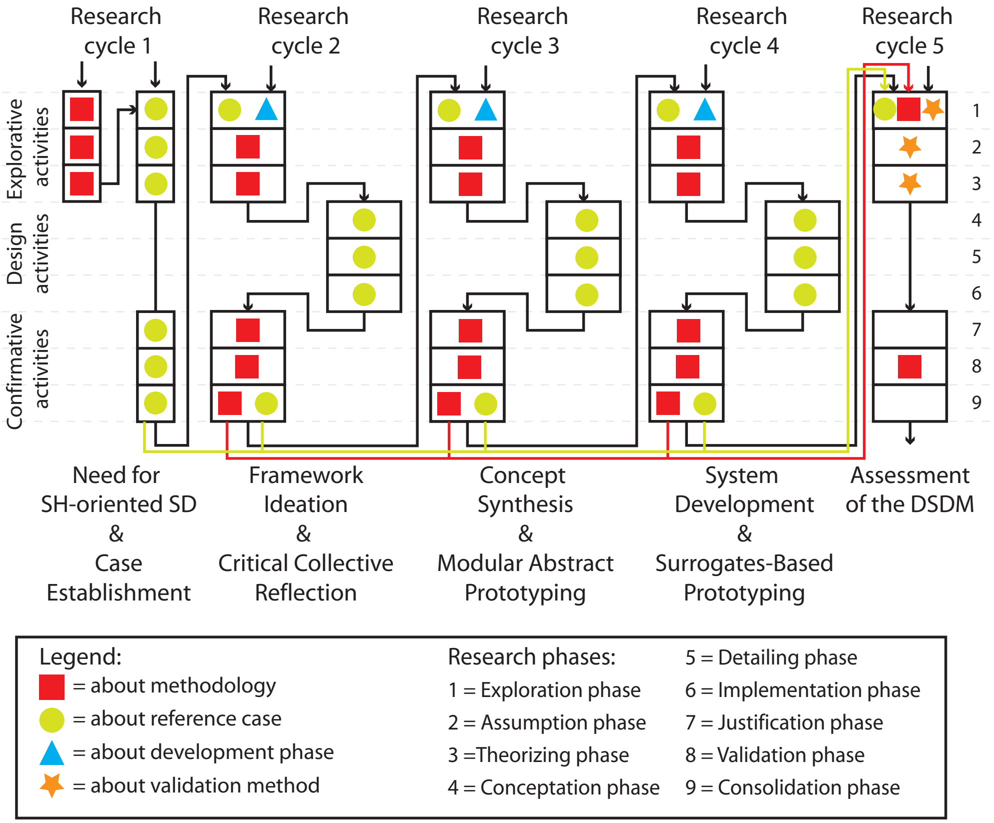


Figure 1.8. Methodological framing of the research

In order to support the execution of research cycles 2, 3 and 4, the framing methodology of design inclusive research was applied. In these research cycles, various implementations of the reference tool were used as dedicated research means. The applied framing provided a sufficient methodological support for each of the phases and facilitated the testing and validation of the conducted research actions and the findings, respectively. In research cycles 1 and 5, a higher level abstraction was applied for the reason that the focus of these cycles was on the multi-phase methodology, rather than on the single-phase methodologies. In the case of these two cycles, research in design context was used as methodological framing. The reason behind this decision was that we investigated phenomena closely related to design in specific contexts. In the schematic overview of the complete research approach, shown in Figure 1.8, the symbols used to depict the knowledge generated during the research activities. Namely, knowledge was generated: (i) concerning the whole of the targeted DSDM (and its component methodologies), (ii) related to the issues of the specific development phases, (iii) related to the reference case, and (iv) related to the needed validation method.

1.9. Structure of the thesis

From a structural point of view, the thesis consists of seven chapters, which presents the work and results in the sequence of the completed research cycles: In Chapter 1, as you just read, a general overview of the problem domain together with the needs, the research hypothesis and objectives, and the methodological framing of the research has been given. Chapter 2 zooms in onto the research objective and discusses the needed designerly software development methodology in a specific context. Here, also the needed reference case is introduced and discussed. Chapter 2 reports on the research work and results achieved in research cycle 1. Afterwards, Chapter 3 presents the research carried out in research cycle 2, i.e. the investigation of the framework ideation phase and the development and testing of the critical collective reflection methodology. In Chapter 4, we covered the research of research cycle 3, in which the concept integration was examined and a methodology for modular abstract prototyping was developed and tested. Chapter 5 reports on the investigation of the system development phase, and on the development and testing of the surrogates-based prototyping methodology, conducted in research cycle 4. Chapter 6 addresses the research carried out in research cycle 5. In this last cycle, the complete designerly software development methodology integrating its three single-phase component methodologies was externally validated. Finally, Chapter 7 gives conclusions on the overall PhD research and results.

1.10. Own publications

During the PhD project, parts of the research work and results, reported in this thesis, have been published in conference proceedings and reference journals. Publications were made on the topic of each research cycle: The study and description of the reference case (discussed in research cycle 1) was reported in [1] and [2]. In addition, we also discussed the framework ideation of research cycle 2 in [1]. Related to research cycle 3, the initial exploration towards abstract prototyping was discussed in [3], and the further developed modular abstract prototyping was proposed in [4], [5] and [6]. Finally, research cycle 5 was presented in [7]. The publications which have been processed in the thesis are listed below:

[1] Du Bois, E., Horvath, I., and Van Doorsselaer, K., (2010), "Critical review of smart energy saving in household electronics", Proceedings of the TMCE 2010, Delft University of Technology, Ancona, Italy, pp. 1147-1160.

[2] Du Bois, E., and Horvath, I., (2012), "An easy-to-use methodological proposal for considering ubiquitous controllers in energy use optimization", in: Design for innovative value towards a sustainable society, Matsumoto, M., Umeda, Y., Masui, K., Fukushige, S. (Eds.), Springer Netherlands, pp. 344-349..

[3] Du Bois, E., and Horváth, I., (2011), "Abstract prototyping in software engineering: A review of approaches", Proceedings of the ICED11, Technical University of Denmark, Copenhagen, p. 10

[4] Du Bois, E., and Horváth, I., (2012), "Modular abstract prototyping as an instrument to demonstrate software tool concepts for multiple stakeholders", Proceedings of the TMCE, Horvath, I., Albers, A., Behrendt, M., Rusák, Z. (Eds.), Karlsruhe, Germany.

[5] Horváth, I. and E. Du Bois, Using modular abstract prototypes as evolving research means in design inclusive research, in ASME 2012 International Design engineering Technical conferences & Computers and information in Engineering Conference (IDETC/CIE 2012). 2012: Chicago, USA.

[6] Du Bois, E., and Gerritsen, B.H.M., (2013), "Demonstration of software concepts to multiple stakeholders using modular abstract prototyping", CoDesign (special issue - Technologies for collaboration).

[7] Du Bois, E., and Horváth, I., (2013), "Operationalization of the quadrant-based validation in case of a designerly software development methodology", Proceedings of the ICED13, Seoul, South Korea, p. 10.

1.11. References

[1] Maslow, A.H., and Lewis, K.J., (1987), "Maslow's hierarchy of needs", Salenger Incorporated.

[2] Maslow, A.H., (1943), "A theory of human motivation", Psychological review, Vol. 50 (4), p. 370.

[3] Lomas, J.C., (2013), "Climbing the needs pyramids", SAGE open, Vol. 3 (3), p. 7.

[4] Zhang, T., and Dong, H., (2009), "Human-centred design: An emergent conceptual model", Proceedings of the Include2009, Royal College of Art, London, p. 7.

[5] Marwedel, P., (2011), "Embedded system design: Embedded systems foundations of cyber-physical systems", Springer Science+ Business Media, p. 400.

[6] Sanchez, E., Squillero, G., and Tonda, A., (2012), "Industrial applications of evolutionary algorithms (vol. 34)", Springer, p. 114.

[7] Horváth, I., and Gerritsen, B.H.M., (2012), "Cyber-physical systems: Concepts, technologies and implementation principles", Proceedings of the TMCE 2012, Karlsruhe, Germany, pp. 19-36.

[8] Williams, E., (2011), "Environmental effects of information and communications technologies", Nature, Vol. 479, pp. 354–358.

[9] Wiebe, J., (2009), "Seven levels of interaction", OneMind learning experience design, Vol. 2013, 2009.

[10] Revilla, E., Prieto, I.M., and Prado, B.R., (2010), "Knowledge strategy: Its relationship to environmental dynamism and complexity in product development", Knowledge and process Management, Vol. 17 (1), pp. 36-47.

[11] Mueller, W., Becker, M., Elfeky, A., and DiPasquale, A., (2012), "Virtual prototyping of cyber-physical systems", Proceedings of the Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific, IEEE, pp. 219-226.

[12] Buchanan, R., (1999), "Design research and the new learning", Design Issues, Vol. 17 (4), p. 21.

[13] Kelley, D., (2013), "The future of design is human-centered", TED global 2013, www.ted.com/talks/david_kelley_on_human_centered_design.html, 2013.

[14] Russo, S.D., (2012), "A brief history of design thinking: How design thinking came to 'be'", ithinkidesign.wordpress.com, Vol. 2013, Swinburne University, 2012.

[15] Kim, C., (2012), "Anticipating soft problems with consumer electronic products - how do soft problems interact with user characteristics and product properties?", PhD thesis - industrial design engineering, Technische universiteit Delft, p. 258.

[16] Beysen, A., Lameillieure-Kharatichvili, M.D., Lenstra, R., and Oskamp, J., (2012), "Cecilia's keuze - ontwerpen met meerwaarde op basis van gebruikersinzichten", LannooCampus, p. 144.

[17] Greenbaum, J., and Loi, D., (2012), “Participation, the camel and the elephant of design: An introduction”, CoDesign, Vol. 8 (2-3), pp. 81-85.

[18] Pea, R.D., (1987), “User centered system design: New perspectives on human-computer interaction”, Journal educational computing research, Vol. 3, pp. 129-134.

[19] Norman, D.A., (1988), “The psychology of everyday things”, New York, p. 288.

[20] Norman, D.A., (2002), “The design of everyday things”, The Perseus Books Group, p. 288.

[21] Constantine, L., Biddle, R., and Noble, J., (2003), “Usage-centered design and software engineering: Models for integration”, Proceedings of the IFIP Working Group, Vol. 2, Citeseer, pp. 3-10.

[22] Constantine, L.L., (1999), “Bare essentials: Simplifying user interfaces by simplifying use cases”, Application Note, p. 4.

[23] Constantine, L.L., and Lockwood, L.A.D., (2002), “Usage-centered engineering for web applications”, IEEE software, Vol. 19 (2), p. 15.

[24] Constantine, L., (2004), “Beyond user-centered design and user experience: Designing for user performance - preprint”, Cutter IT Journal, Vol. 17 (2).

[25] Kimbell, L., (2009), “Insights from service design practice”, Proceedings of the 8 th European Academy Of Design Conference The Robert Gordon University, Aberdeen, Scotland, pp. 249-253.

[26] Kimbell, L., (2010), “From user-centred design to designing for service”, Proceedings of the Design Management Conference, London, p. 9.

[27] Rouse, W.B., (1991), “Design for success: A human-centered approach to designing successful products and systems”, Wiley-Interscience New York, p. 304.

[28] Maguire, M., (2001), “Methods to support human-centred design”, International Journal of Human-Computer Studies, Vol. 55 (4), pp. 587-634.

[29] Steen, M., (2008), “The fragility of human-centred design”, PhD thesis - Industrial design engineering, Technische Universiteit Delft, p. 252.

[30] Van Patter, G., and Jones, P., (2003), “Understanding design 1, 2, 3, 4: The rise of visual sensemaking”, NextD Journal (Special Issue Peter Jones PhD interviews GK VanPatter - ReRethinking Design).

[31] Van Patter, G., and Pastor, E., (2011), “Next design geographies: Understanding design thinking 1,2,3,4.”, NextD Journal.

[32] Wassermann, A., (2012), “Design 3.0. How design grew from ‘stuff’ to sociotechnical systems and became too important to leave to designers”, Design to improve life education, p. 10.

[33] Baelus, C., (2003), “Methodologie van het ontwerpen 1”, Productontwikkeling, departement ontwerpwetenschappen, Hogeschool Antwerpen, Antwerp.

[34] Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., and Cajander, Å., (2003), “Key principles for user-centred systems design”, Behaviour and Information Technology, Vol. 22 (6), pp. 397-409.

[35] Thomke, S., and Fujimoto, T., (2000), “The effect of “front-loading” problem-solving on product development performance”, Journal of product innovation management, Vol. 17 (2), pp. 128-142.

[36] Alam, I., (2002), “An exploratory investigation of user involvement in new service development”, Journal of the Academy of Marketing Science, Vol. 30 (3), pp. 250-261.

[37] Braet, J., and Verhaert, P., (2007), “The practice of new products and new business”, Uitgeverij

acco, Leuven, p. 392.

[38] Pei, E., Campbell, I.R., and Evans, M.A., (2010), “Development of a tool for building shared representations among industrial designers and engineering designers”, CoDesign, Vol. 6 (3), pp. 139-166.

[39] Zwicky, J., (2013), “What design decisions did you make today?”, in: designfulcompany.com (Ed.) design models, 2013.

[40] Roozenburg, N.F.M., and Eekels, J., (1995), “Productontwerpen, structuur en methoden (2nd edition)”, Uitgeverij Lemma BV, Utrecht, The Netherlands.

[41] Buijs, J., and Valkenburg, R., (2005), “Integrale productontwikkeling. Derde druk.”, Lemma, The Hague, The Netherlands, p. 415.

[42] Verhaert, K., (2009), “Het verhaal achter nieuwe producten - toegevoegde waarde door productontwikkeling en industrieel design”, Stichting kunstboek, p. 240.

[43] Ulrich, K.T., and Eppinger, S.D., (2008), “Product design and development”, McGraw-Hill p. 368.

[44] Anonymous, (2012), “Components of a computer system and modes of use: Types of software”, Vol. 2013, en.wikibooks.org/wiki/A-level_Computing/ CIE/Computer_systems_communications_and_software/ Components_of_a_computer_system_and_modes_of_use/ Types_of_software, 2012.

[45] Anonymous, (2013), “Types of computer software”, www.mapsofworld.com/referrals/ computers/types-of-computer-software.html, 2013.

[46] ChinUSA Innovators LLC, (2008), “Types of software systems”, Vol. 2013, www.osait.com/ event/197/txt/Types-of-Software-Systems.html, 2008.

[47] Gasson, S., and Holland, N., (1995), “The nature and processes of it-related change”, Proceedings of the IFIP WG8.2 Conference: Information Technology and Changes in Organizational Work, Cambridge, UK

[48] Majid, R.A., Noor, N.L.M., Adnan, W.A.W., and Mansor, S., (2010), “A survey on user involvement in software development life cycle from practitioner’s perspectives”, Proceedings of the Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, pp. 240-243.

[49] Iivari, J., Isomäki, H., and Pekkola, S., (2010), “The user – the great unknown of systems development: Reasons, forms, challenges, experiences and intellectual contributions of user involvement”, Information Systems Journal, Vol. 20 (2), pp. 109-117.

[50] Kujala, S., (2003), “User involvement: A review of the benefits and challenges”, Behaviour & information technology, Vol. 22 (1), pp. 1-16.

[51] Rannikko, P., (2011), “User-centered design in agile software development”, MSc. thesis, p. 75.

[52] Constantine, L., (2004), “Beyond user-centered design and user experience: Designing for user performance “, Cutter IT Journal, Vol. 17 (2), p. 12.

[53] Seffah, A., Gulliksen, J., and Desmarais, M.C., (2005), “An introduction to human-centered software engineering”, in: Human-centered software engineering—integrating usability in the software development lifecycle, Springer, pp. 3-14.

[54] Iivari, N., (2004), “Enculturation of user involvement in software development organizations - an interpretive case study in the product development context”, Proceedings of the Proceedings of the third Nordic conference on Human-computer interaction, ACM, Tampere, Finland, pp. 287-296.

[55] Mujumdar, A., Masiwal, G., and Chawan, P., (2012), “Analysis of various software process

models", Analysis, Vol. 2 (3), pp. 2015-2021.

[56] Awad, M., (2005), "A comparison between agile and traditional software development methodologies", honours programme of the School of Computer Science and software Engineering, The University of Western Australia, 2005, p. 77.

[57] Hass, K.B., (2007), "The blending of traditional and agile project management", PM world today, Vol. 9 (5), pp. 1-8.

[58] McMahon, P.E., (2004), "Bridging agile and traditional development methods: A project management perspective", The Journal of Defense Software Engineering, p. 5.

[59] Rico, D.F., (2008), "What is the roi of agile vs. Traditional methods?", TickIT International, Vol. 10, pp. 9-18.

[60] Vinekar, V., Slinkman, C.W., and Nerur, S., (2006), "Can agile and traditional systems development approaches coexist? An ambidextrous view", Information Systems Management, Vol. 23 (3), pp. 31-42.

[61] Beck, K., and Andres, C., (2004), "Extreme programming explained: Embrace change", Addison-Wesley Professional, p. 224.

[62] Schwaber, K., (2004), "Agile project management with scrum", O'Reilly Media, Inc., p. 170.

[63] Palmer, S.R., and Felsing, M., (2002), "A practical guide to feature-driven development", Prentice Hall, p. 304.

[64] Leau, Y.B., Loo, W.K., Tham, W.Y., and Tan, S.F., (2012), "Software development life cycle agile vs traditional approaches", Proceedings of the International Conference on Information and Network Technology (ICINT 2012), IPCSIT (Ed.) Vol. 37, IACSIT Press, Singapore.

[65] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., and Jeffries, R., (2001), "The agile manifesto", www.agilemanifesto.org/principles.html, Vol. 7 (08), p. 2009.

[66] Blomkvist, S., (2005), "Towards a model for bridging agile development and user-centered design", in: Human-centered software engineering—integrating usability in the software development lifecycle, Springer, pp. 219-244.

[67] Glass, R.L., (2001), "Agile versus traditional: Make love, not war!", Cutter IT Journal, Vol. 14 (12), pp. 12-18.

[68] Gasson, S., (2003), "Human-centered vs. User-centered approaches", Journal of Information Technology Theory and Application, Vol. 5 (2), pp. 29-46.

[69] Kaindl, H., Constantine, L., Pastor, O., Sutcliffe, A., and Zowghi, D., (2008), "How to combine requirements engineering and interaction design?", Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08, Barcelona, Catalunya, pp. 299-301.

[70] Nielsen, J., (1994), "Usability engineering", Morgan Kaufmann, London, p. 362.

[71] Nakakoji, K., and Yamamoto, Y., (2004), "Knowledge interaction design for creative knowledge work", Transactions of the Japanese Society for Artificial Intelligence, Vol. 19, pp. 154-165.

[72] Martini, A., Massa, S., and Testa, S., (2012), "The role of social software for customer co-creation: Does it change the practice for innovation", International Journal of Engineering Business Management, Vol. 4, pp. 1-10.

[73] Constantine, L., and Lockwood, L., (2000), "Structure and style in use cases for user interface design", in: Object modeling and user interface design., Harmelen, M.v. (Ed.).

[74] Holzl, M., Rauschmayer, A., and Wirsing, M., (2008), "Engineering of software-intensive systems: State of the art and research challenges", in: Software-intensive systems and new

computing paradigms, Martin, W., Jean-Pierre, B., Matthias, H., Axel, R. (Eds.), Springer-Verlag, pp. 1-44.

[75] Stompff, G., (2012), “Facilitating team cognition - how designers mirror what npd teams do”, industrial design engineering, Technische Universiteit Delft, p. 344.

[76] Broy, M., (2006), “The ‘grand challenge’ in informatics: Engineering software-intensive systems”, Computer, Vol. 39 (10), pp. 72-80.

[77] Ahmadi, N., Jazayeri, M., Lelli, F., and Nesic, S., (2008), “A survey of social software engineering”, Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008. , IEEE, pp. 1-12.

[78] Hammouda, I., Koskimies, K., and Mikkonen, T., (2011), “Managing concern knowledge in software systems”, International Journal of Software Engineering and Knowledge Engineering, Vol. 21 (07), pp. 957-987.

[79] Kyng, M., (1991), “Designing for cooperation: Cooperating in design”, Communications of the ACM, Vol. 34 (12), pp. 65-73.

[80] Fischer, G., Redmiles, D., Williams, L., Puhr, G.I., Aoki, A., and Nakakoji, K., (1995), “Beyond object-oriented technology: Where current approaches fall short”, Human-Computer Interaction, Vol. 10 (1), pp. 79-119.

[81] Postma, C., (2012), “Creating socionas - building creative understanding of people’s experiences in the early stages of new product development (phd thesis)”, Indsutrial design engineering, Delft University of Technology, p. 288.

[82] van Gameren, B., (2011), “Applying an industrial design engineering approach in software design”, industrial design engineering, Technische Universiteit Delft.

[83] van Kuijk, J., (2010), “Managing product usability - how companies deal with usability in the development of electronic consumer products”, PhD thesis - industrial design engineering, Technische Universiteit Delft, p. 370.

[84] Sleeswijk Visser, F., (2009), “Bringing the everyday life of people into design”, PhD thesis - industrial design engineering, Technische Universiteit Delft, p. 273.

[85] Bødker, K., Kensing, F., and Simonsen, J., (2011), “Participatory design in information systems development”, in: Reframing humans in information systems development, Isomäki, H., Pekkola, S. (Eds.), Vol. 201, Springer London, pp. 115-134.

[86] Horváth, I., (2013), “Structuring the process of design research - a necessary step towards ensuring scientific rigor”, Proceedings of the ICED13, Seoul, Korea, p. 10.

[87] Horváth, I., (2007), “Comparison of three methodological approaches of design research”, Proceedings of the International conference on engineering design, Design Society, Cité des sciences et de l’industrie, Paris, France, p. 11.

[88] Horvath, I., (2008), “Differences between ‘research in design context’ and ‘design inclusive research’ in the domain of industrial design engineering”, Journal of Design Research, Vol. 7 (No. 1), p. 23.

Chapter 2

Research cycle 1

Conceptualization of the designerly software development methodology

2.1. Introduction

2.1.1. Objectives of the research cycle

In this cycle, we want to deepen our insight in and clarify the phenomenon of having the need to support software development by a stakeholder-oriented approach. In the previous chapter, the domain was introduced and a general explanation about the context, evolution, and the relationship of the product development and software development domains was given. In this chapter, we narrow down to a more specific context and a more specific objective. This is done in order to get an overview of the stakeholder-oriented software development approaches and to identify the related issues, gaps, and opportunities. A detailed analysis was done to explore and describe the studied phenomena in sufficient details. In the rest of the chapter, we present the work done towards theory forming about the needed methodology. The methodology development included the compilation of an underpinning theory, the set of source methods, the execution procedure, the instruments, and criteria for goodness. We found that, in order to be able to investigate the researched phenomenon in context, we had to identify a reference case that provided the context for the stakeholder-oriented software development. The goal was to use the reference case throughout the whole research to validate and verify all development phases, activities, and methodologies. In the second part of this chapter, we explain the chosen case.

2.1.2. Research approach

The research cycle has methodologically been conducted according to the principles of research in design context. The overall approach of the research cycle is shown in Figure 2.1. First, we explored the current situation, the general knowledge problems, and the need for stakeholder (SH) involvement in the software development (SD) process (Section 2.2). The existing approaches of SH-oriented SD were examined and discussed together with its main aspects. Based on this exploration, we could revisit and refine the research problem (Section 2.3) to identify more detailed assumptions (Section 2.4), and we could formulate a theory for the needed designerly software development methodology (Section 2.5). The third conclusion from this exploration was that we cannot handle the complex problem in general. Consequently, another explorative activity was needed to define and explore a reference case that could be used as concrete problem demonstrator. In the course of this research cycle (research in design context), a concurrent dual exploration was needed as visualized in Figure 2.1. In this Figure, the red blocks refer to the knowledge that was generated for the methodology, and the green dots are related to the reference case that was explored and consolidated. In Section 2.6, this specific reference case is discussed. The requirements are defined in Section 2.7 and specifications are summarized in Section 2.8. Confirmative research actions were carried out to justify, validate and consolidate the research activities, methods and findings.

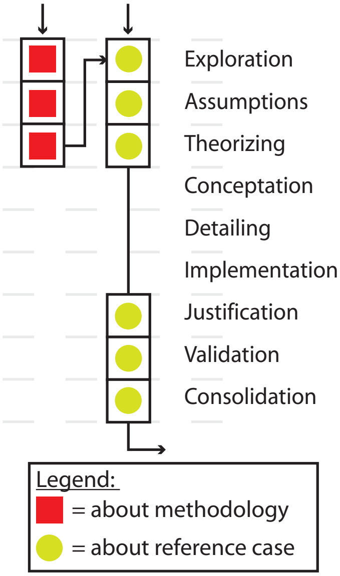


Figure 2.1. Approach in RC1

2.2. Knowledge aggregation on the development of second and third generation software products

To start the discussion, we refer to the issues accompanying the development of second and third generation interactive software applications. In this context, we discussed both the traditional and the agile development approaches. As shown in Table 2.1, neither agile nor traditional approaches are in themselves completely suitable for the development of these products. Therefore, at least a mix of the two approaches should be found. The literature is in agreement on the fact that the design of these products is a highly complex and demanding activity. Software designers often deal with changing requirements and technical environments. They often need to explore new problem and knowledge domains where the knowledge about a design cannot be found readily. The characteristics and behaviors of the software and the hardware systems to be considered in the design are often unknown and the uncertainty user and quality requirements are high. Under such complex environment, a software designer needs sound reasoning capabilities to make good design decisions and to devise a good design solution [1]. The use of agile methodologies has increased significantly over the past decade in the industry, promoting the value of human-centric software development processes [2]. This growing use entails the need to adjust agile methodologies to bigger, more complex-system development projects, where architecture plays a significant role. However, many experts believe that an essential conflict exists between the requirement of minimalism in agile methods and the need for well-defined and documented architecture in complex systems [3]. The main challenges that could be identified regarding the development of software components of second and third generation products are: (i) dealing with large projects with multidisciplinary teams, (ii)

Table 2.1. Matching characteristics of second and third generation products and agile and traditional methodologies (colored cells are best match)

characteristics of 2nd and 3rd generation products	**how agile methodologies deal with it**	**how traditional development methodologies deal with it**
large projects large multidisciplinary teams	best for small projects smaller teams	best for large projects larger teams
complex systems	incremental development to handle complexity	iterative development to handle complexity
often critical system	pair programming	various testing
constant evolving requirements that are unknown at start	unstable and volatile requirements, rapid change	requirements set early, largely stable
multiple and different stakeholders justification and validation	face-to-face communication with stakeholders SH involvement in process	documented communication, formal interaction, observations

synthesis of complex systems, (iii) implementation of critical systems, and (iv) adapting to change and dealing with uncertainty at the start, and (v) dealing with multiple stakeholders who have to validate the system.

Project size and development team

One of the limitations of agile approaches is project size [2]. The key elements and parameters are project size, budget, duration, and project team organization. The larger the team or more budget the project needs, the bigger the challenges raised by the project. Thus large projects go together with the problem of compiling a huge number of requirements, the demand for more people, and more coordination activities. Systematic methodologies support these by providing plans, documentation, processes, and better communication and coordination across large groups.

As shown in Figure 2.2, agile methods are developed for small close-located teams who deal with small-size projects. Agile software development (ASD) is especially useful for highly evolving projects. The core team usually consists of two or three developers who write code in pairs (for quality control), the customer/end user, IT architects, a business analyst and a project manager. The work is accomplished through a series of sessions, where the team discusses the possible concepts and solutions, writes code, then tests the working modules of the system, and repeats the process, if necessary. There is a minimal documentation as the team almost relies exclusively on informal internal communication in ASD. As opposite, traditional waterfall methods are more suited to large robust projects for which all requirements are known in advance. Critical factor in traditional development is the process, organization and the documentation, while ASD focuses on the talents and skills of individuals, and molds processes of specific people and teams. Agilists use talking face-to-face as the main communication means, while in heavyweight methodologies they prefer systematic documentation and collaboration.

	Small size project & team	Large size project & team
High change	Agile development	? (typical complex system)
Low change	Both	Traditional development

Figure 2.2. Influence of project size and evolvability on development approach

The biggest limitation of agile methodologies is that they cannot be used in the case of large projects and teams, because as the size of the project grows, coordination of the interfaces becomes a dominant issue. ASD relies on face-to-face communication, breaks down to person oriented task execution and becomes more difficult and complex with more than 20 developers. In contrast, heavyweight and plan-driven methods scale better to large projects. ASD relies on tacit knowledge embodied by the team, rather than on writing the knowledge down as documentation. However, there is often a risk that this may lead to architectural mistakes that cannot be easily detected by external reviewers due to the lack of documentation. There exist several difficulties in putting ASD into practice: one among these is caused by the significantly reduced documentation, which limits transparency, the opportunity of monitoring, and exploration of mistakes or errors. Often the code itself should act as a document. For this reason, developers who are accustomed to agile methods have a tendency to place more comments in the code as explanation and clarification. However, it is difficult for novice developers, or new team members, to complete tasks when they could not adequately comprehend the project. On the other hand, traditional methods stress the importance of documentation in providing guidelines and clarification on the project for the development team, by doing so, there is less concern that the developers are not knowledgeable of the projects' details or the availability of a knowledgeable developer when critical decisions are to be made [4].

The issue of complexity

The appearance of the second and third generation products displays an enormous increase in software complexity, shorter innovation cycles, and in the ever-growing demand for extra functional requirements (e.g. software safety, reliability, and timeless) at affordable costs [5]. Such a growth in complexity directly leads to difficulties in every step of product development, e.g. in determining the correct combination of parameters to obtain the desired behavior for a software tool. Each choice has intricate and sometimes not foreseeable repercussions. More and more, industry must resort to heuristic and meta-heuristic techniques to find the best alternative between different possibilities [6]. It is possible that single parts of a software tool become so intricate that the interaction between them can lead to extreme difficulties in making predictions on the behavior and the lifespan of the software. It is not clear whether agile or traditional approaches are better suited to handle this complexity. Traditional development bases on up-front planning to handle complexity, while agile development uses short and well-defined spans of work time, called sprints. Regarding the complexity of the software, a good architecture is crucial. Although ASD does not focus on up-front development, there seems to be an agreement in the literature do agree that architecture is just as important in agile (specifically, XP) projects as it is in any software project. Moreover, according to [7], Booch states that all good software-intensive architectures are agile [8], and Spinellis notes that architecture is always important in the case of large and complex projects, regardless of their development methodology: “Look at a large successful software system and beneath it you’ll find an architecture that’s kept its evolution on track.” [9].

Critical systems and risk

Project criticality is one of the most important risk factors in the software development process. Agile methods are used in applications that can be built quickly and do not require extensive quality assurance. Critical, reliable, and safe systems are more suited to a heavyweight methodology, where a plan-driven process is most needed for high assurance software [2]. If a project is critical, all requirements must be well defined before the implementation of the software. Heavyweight traditional approaches set affront goals such as predictability, repeatability, and optimization, which are often characteristics of reliable safety critical software development. Although the agile team identifies and prioritizes the feature based value, focusses on the high-risk components of the system, and produces the highest value features first, most agile approaches do not consider traditional walkthroughs and code inspections during the design process, it puts the emphasis on pair programming in small creative groups and informal reviews as their quality control mechanism.

Evolvability and uncertainty

The development of second and third generation products can be characterized as projects that are full of unknowns and with many uncertainties, such as vague and frequently changing requirements, unproven technologies, or unknown customers and other stakeholders. One major difference between agile development and conventional development methodologies is that the former ones possess the ability to deliver results successfully, quickly, and inexpensively in case of complex projects with ill-defined requirements. It is the ability to respond to change that often determines the success or failure of a software project. In contrast to the traditional approaches, the agile development avoid upfront requirement gathering as stakeholders often could not provide all requirements in sufficient details for implementation at the beginning of a project [10].

Regardless of what concrete process is used, iterative and incremental project planning is key to success [11]. The agile approaches incorporate many rapid iterative planning and development cycles, allowing a project team to constantly evaluate the evolving product and obtain immediate feedback from users or stakeholders. Short development iterations provide opportunities to shift priorities or change direction. No planning and project management effort can substitute for user and SH feedback. Even in case of small projects, user feedback and iterative planning are essential. It is a common phenomenon that stakeholders cannot decide on the features to be included in the software. The iterative approach also allows stakeholders to postpone decisions to some future iteration, when more information or technology is available to optimize the choice or solution [12].

However, flexibility, which is the main aspect of ASD, also has two big flip sides. One is the potential for scope creep, which can create the risk of ever-lasting projects. The other is much less predictability, at the start and during the execution of the project, about what the project is actually going to deliver. This can make it more difficult to define a business case for the project, for instance, to negotiate fixed price projects without a strong mature and clear vision, and the discipline of fixing. While agile methodologies are considered effective in projects with unclear requirements, they have actually little to say about how those requirements should be gathered and made clear in the early phase of development. Agile methodologies do not advise on how to do systems or requirements analysis. The team expects the stakeholders to deliver the requirements, but if that does not work out, the team itself has to build a first concept and ask if it was what the SH expected. As incorporating the effects of changes to requirements are more expensive after the implementation than before it, not doing at least some up-front requirements analysis would seem to contradict the idea of maximizing the work done [13].

Software justification and validation

Testing has always been an important part of the development process of software-intensive systems. The software community agrees that they should produce the highest quality software for the lowest cost [14]. To ensure the quality, testing plays an important and critical role in the process, because a comprehensive testing is much more than just finding and eliminating bugs in the software. Testing should extend to the evaluation of functional and non-functional properties, and to the satisfaction of the potential users to see if it fulfills the requirements. The first evaluation is automated developer testing (unit and integration testing) which is a prerequisite for producing high quality code. The second evaluation focusses on customer acceptance testing where the stakeholder representatives test the actual working software. It is obvious that if testing happens with the involvement of various stakeholders as an early confirmation, rather than as a retrospective analysis, then many iteration and adjustment steps can be eliminated and the confidence of the stakeholders can be increased. Proof of concepts, throwaway prototypes, user stories, and mockups are the different ways of capturing requirements. They improve the communication with clients and allow specifying and prototyping (a part of) the intended software system.

Testing can be conducted at the end of each sprint or as soon as a reasonable set of functionality, such as a user story, is complete [13]. Testing is also involved in agile software development. ASD is based on the idea of incremental and iterative development, in which the phases are revisited over and over again. It iteratively improves software by using stakeholder feedback to converge on solutions, while many traditional waterfall methodologies put feedback and testing at the last stage of their project lifecycle. However, agile methods do not consider the user side of software, i.e. user interface and usability. “It is not a weak point, it is an absence” [15]. When it comes to user interface design, agile processes prefer simplistic forms or iterative paper prototyping rather than model-driven design. Agilists believe that testing the user interface is labor intensive and time consuming.

2.3. Issues of stakeholder involvement

Stakeholder involvement offers many opportunities for testing and validating software products, especially complex systems. However, in practice, there are also some pitfalls related to the SH involvement: The first issue is related to the meetings that the development team and the stakeholders will hold after every deliverable. At these events, the team members communicate and summarize the results of the completed work done in the concerned iteration cycle. Most of the time, developers will find the regular meetings tedious and tiring as they would have to present their responsible modules to SHs and other members repeatedly. Moreover, various changes will most likely happen in every iteration cycle due to the changes in the requirements. The second issue is that interpersonal and social skills are crucial for the entire development team, to enable good communication and SH involvement. A third issue emerges when the SH is considered as part of the development team throughout the whole development of the software as there is a risk of shortfall of tacit knowledge. If we have only a limited number of participating SH, who are committed, knowledgeable, collaborative, and empowered, there is a chance that we will have a unified set of requirements. However, if we have many SH, we have to count on different viewpoints and conflicts between them. This risk could be reduced by plan-driven methods using documentation, planning, architecture reviews, and project reviews by independent experts.

A last important issue is that user-involvement happens in different manners in the current approaches of system development [16, 17]. Theoretically, stakeholder involvement in design can be seen as a creative and communicative process that involves interplay between setting and solving the problem, mutual reciprocal learning and design by doing. Comparable distinctions are made between the different approaches of SH involvement by [18-22]. They all vary between a passive or symbolic involvement to the other extreme of being part of the development team. The most used modes of SH-involvement were (i) face-to-face interviews, (ii) user visits and meetings, (iii) brainstorming, (iv) user observation and feedback, (v) phone, faxes and emails, and (vi) focus group discussions [19]. In-depth interviews and user visits to the service design sites, including team meetings, were the two dominant modes of SH involvement because interviews and group meeting were stated to be easier and inexpensive modes of obtaining user input. In traditional SD, stakeholder involvement was only performed sporadically. In predictive SD, stakeholders were the object of study and forecasts were conducted on how they will behave with the product, what their needs are and how it should be realized. In participatory SD, the SH were involved to give answers to the questions but also to reason about the implementation and realization. In this research, we position our approach to be part of the last category.

2.4. Detailing the research problem and objective

Building further on the core knowledge from previous chapter, we detailed the sufficiency aspects for the needed methodology. The sufficiency aspects of the needed methodology are more related to the clarification of the detailed problems of SH involvement. We concluded that although there are already many attempts in this direction, there are still various issues. We concluded that there is no link between why stakeholders need to be involved, how to involve them, when to involve them, who to involve and how it should be ideally happening. We analyzed the opportunities of ASD and systematic SD approaches and found that a mix of these approaches enables the best SH involvement.

To detail the needed software development methodology, we used the framework of Ross from 1975 [23]. Although at that time, there was no component or SH-oriented software development, this is still a very useful framework to identify and put emphasis on the process, principles and goals of software development methodologies, methods, techniques and tools. In this Section, we will use it to further detail the needed methodology. As shown in Figure 2.3, which gives a visualization of the framework, we bring together the four fundamental goals, seven principles and five basic process constituents of which the framework consists. In particular, we focused on the most relevant building blocks, which

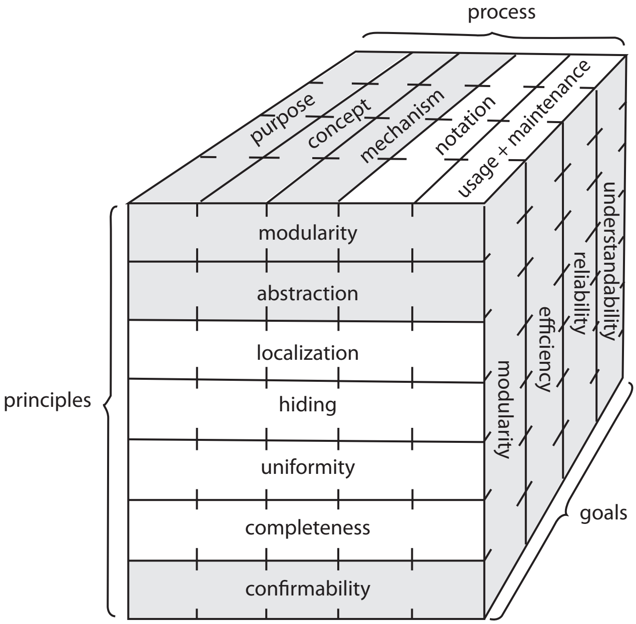


Figure 2.3. Important building blocks for the designerly software development methodology

are explained in Table 2.2. Based on the framework, the building blocks were used to explain the different aspects of the methodology, for example, how the concept phase will use modularity to achieve understandability. This contributed to achieving the goals of the software development methodology, which was supporting the development of complex software products that are constantly evolving, which stakeholders can rely on, and which are efficient in their usage. Towards this end, achieving a high level (insightful) stakeholder involvement was crucial.

2.4.1. Assumptions concerning effective stakeholder involvement in a software development methodology

Based upon the reasoning with the building blocks in Table 2.2, we could derive the following assumptions to define the SH-involving SD methodology:

Assumption 1:

Software development is a social activity, as it is (i) carried out in multidisciplinary teams, including domain specific experts, architects, marketers, designers, etc., and (ii) stakeholder-oriented because understanding the requirements that a product or part should meet is crucial for its success.

Assumption 2:

Collaboration is crucial to achieve an effective decision making and qualitative change proposals aiming at producing software effectively and efficiently.

Assumption 3:

A multi-phase software development methodology should focus on supporting a comprehensive stakeholder involvement in the most crucial phases of the development process, namely in the: (i) framework ideation, (ii) concept integration, and (iii) system development phases.

Assumption 4:

Increase of stakeholder involvement in the software development process can be achieved by using certain concepts and means (best practices) of consumer durables development.

Assumption 5:

A designerly software development methodology (DSDM) should be a multiphase methodology that (i) explain when stakeholders should be invited and clarifies how and who to involve, (ii) predict the advantages of collaboration to developers (saving time, fewer design cycles needed, higher acceptance, improved quality of change proposals, better motivated design decisions, etc.), and (iii) make stakeholder involvement easy by explaining what aspects to focus on and how to do it.

Table 2.2. Interpreting the framework for the needed methodology

goals:	modularity to control change and have an adaptable and evolutionary software	managing complexity is an important goal of the methodology to be able to develop complex systems and to be able to present and discuss them with stakeholders
	efficiency of both the process and the resulting software	process efficiency should be increased by early involvement of stakeholders in the process in addition, the efficiency of the result will also increase due to this involvement
	reliability prevention and recovery from failure	reliability should also be considered by involving stakeholders
	understandability acceptability of change + handling of the complexity	the stakeholder-orientation’ main goal is to increase the acceptability of the system as it better suits the needs also the evolvability should be considered for the development of the complex systems
principles:	modularity	separation of concerns should be used as an important principle to handle complexity on the different levels
	abstraction	the principle of abstraction combined with the principle of completeness ensures that a given decomposition level is understandable as a unit, without requiring either knowledge of lower levels of detail, or on how it participates in the system (as viewed from a higher level) to achieve this, the methodology should provide different levels of fidelity
	completeness, confirmability,	the lack of completeness and consistency, and managing the unnecessary differences are issues for stakeholder involvement, which is an important step towards confirmability completeness does not require that every detail is shown, but merely that the concept covers every important detail that is needed for discussion, to find out whether the stated goals have been achieved or what change proposals should be made
process:	purpose	is considered in the first phase of the development process
	concept	is considered in the second phase of the development process
	mechanism	is considered in the third phase of the development process
	notation	production of the software code will not be in the focus of this promotion research
	usage and maintenance	the specific issues of the usage and the maintenance phases will not be addressed, however, the development process will consider these

Assumption 6:

The DSDM should be capable to deal with the typical characteristics of second and third generation software products (e.g. design support tools).

Assumption 7:

The DSDM should be able to explain both the theoretical aspects and the development and implementation of second and third generation software products.

2.4.2. Assumptions concerning the reference case

Following assumptions have been made in the context of the reference case:

Assumption A:

A reference case for design support tool development should be a naturally complex and evolving software-intensive system.

Assumption B:

The reference case should convey an explicit need for a stakeholder-oriented software development approach.

Assumption C:

The reference case should raise the need for the involvement of multiple stakeholders, from all of whom requesting a high-level engagement.

Assumption D:

The reference case development and implantation should be event (or information) driven, instead of algorithmic (or computation) driven.

Assumption E:

The reference case should be a representative example for an ordinary family of software products.

In the actual conduct of the promotion research, these assumptions have been blended and considered concurrently.

2.5. Theory of the Designerly Software Development Methodology

From the literature, we learned that a methodology should rely on an underpinning theory and should offer procedural scenarios, problem solving instruments and a set of tested methods, and should define the criteria of goodness. The interrelationships among the underpinning theory and these implemented constituents of the methodology are graphically visualized in Figure 2.4. The DSDM is the implication or operationalization of the hypothesis that claims that there is a need for a designerly software development methodology that supports stakeholder involvement in the most critical phases of software

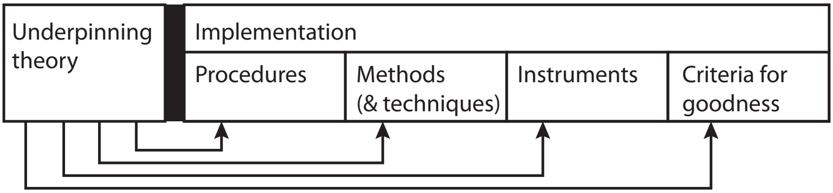


Figure 2.4. Visualization of the interrelationships among the underpinning theory and the different constituents of the methodology

design. In the next Section, we first present the ideas on which the underpinning theory is based and go into the details. Next, we discuss the implementation aspects.

2.5.1. Underpinning theory

The underpinning theory is guiding the practical implementation of the methodology, including that of the procedure, instruments, methods, and criteria. The specific characteristics of second and third generation software products (compared to first generation products) are their functional and structural complexity and the need for evolvability in order to keep fulfilling the stakeholders' needs. The aim of the DSDM is to support the development of the second and third generation software products having these specific characteristics, and to realize an optimal stakeholder involvement in order to increase the efficiency and effectively of the to-be-developed products. Consequently, the underpinning theory of DSDM formulates three principles on which the methodology was based: (i) context-sensitive stakeholder involvement, (ii) managing complexity and evolvability, and (iii) achieving an increasing level of fidelity.

Stakeholder involvement with a view to derive qualitative change proposals

In the application of the methodology, a key issue is to obtain constructive feedback from the stakeholders, including qualitative change and improvement proposals and/or quantitative measures, and to make strengthened decisions. In all phases of the multiphase methodology, this must be achieved, adjusted to the specific objective. The purpose of (software) development is, by means of methods and tools, to facilitate the definition of all desired goals and functionalities of the software. Consequently, the primary measure for an information system to be successful is the degree in which it meets the intended purpose. We based upon the principles of stakeholder involvement that are known for and applied in consumer durable product development. From these participatory approaches, we took over several methods, techniques and instruments such as the use of different demonstration means, and the use of focus group sessions.

Managing complexity and evolvability

To manage the complexities accompanying comprehensive systems, we build on the principle of separation of concerns. Namely, different approaches of concern-based or component-based prototyping have been used over the different phases. The complexity of the targeted products also increases due to the fact that it is not possible to consider all possible requirements, and the characteristics of the different stakeholders could not be known at the start of the development process. This leads to an issue of evolvability during the development process. We define evolvability as the ability to react upon changes in the requirements, type of stakeholders, and software concept. In Figure 2.5, the three interrelated aspects are shown. If one changes, it has an influence on the other two. One way to handle this clarification and evolvability is to use an evolving level of fidelity during the DSDM process that responds to the level of available knowledge on the three aspects.

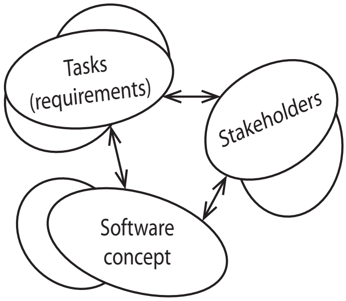


Figure 2.5. Aspects of software evolution

Changing fidelity

The outcome of the sequential application of the component methodologies included in the DSDM operates with an increasing level of fidelity. These levels depend on the amount of information available in the above-mentioned three phases. The methodology that we have developed, suggests to start with a high-level abstraction (that can be embedded in a low-fidelity prototype) and to finish with a high-fidelity testable prototype of the detailed software system. These subsequent forms of prototypes can be adjusted to the contents, stakeholders and contexts. In the process of exploring the stakeholders' opinion, ideas, and recommendations, the prototypes with a specific adjusted fidelity support both the

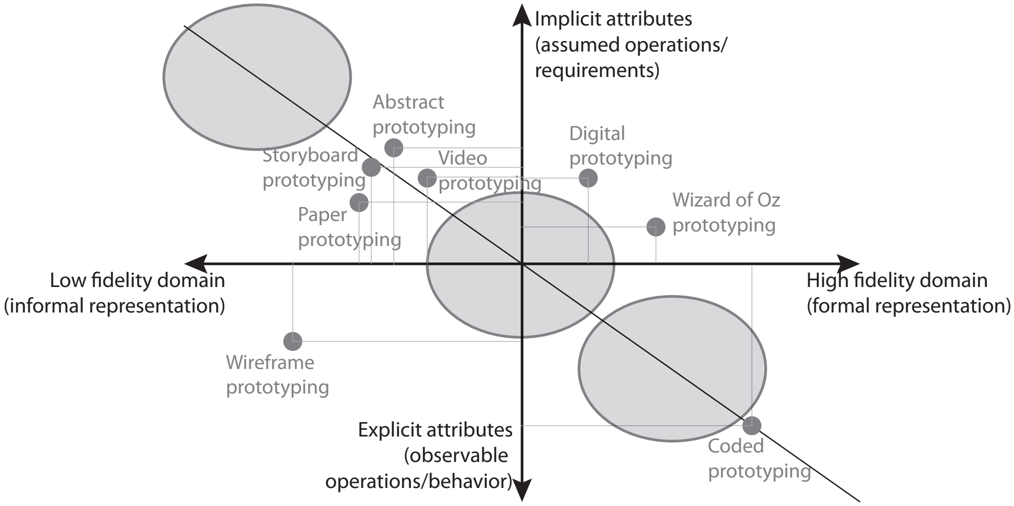
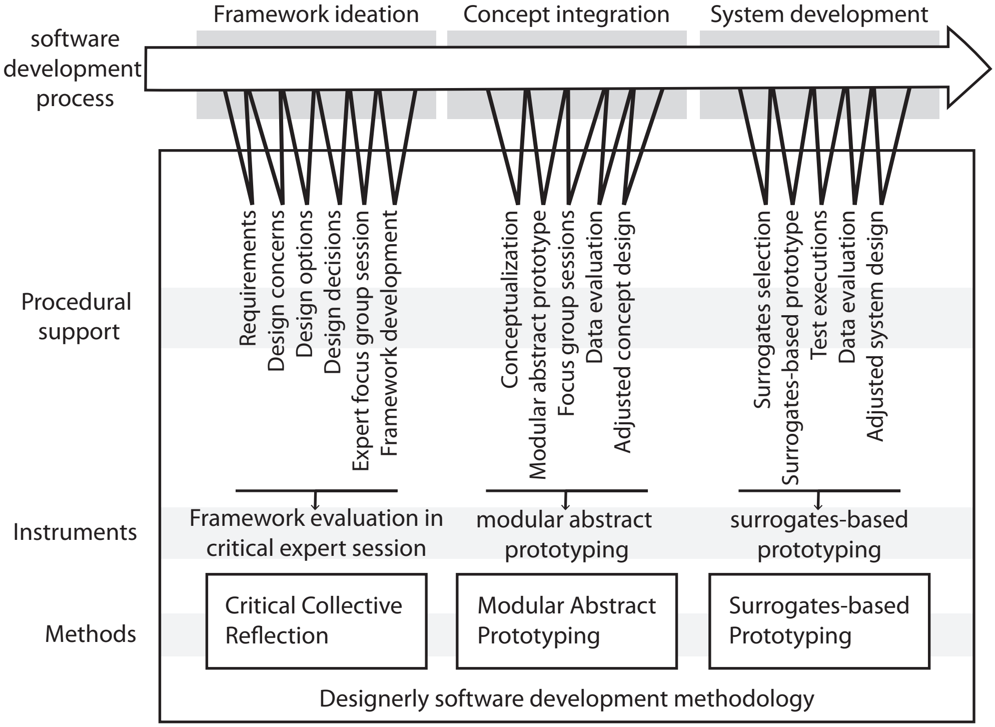


Figure 2.6. Representational fidelity and prototyping approaches of software products

interrogation and the constructive activities. They trigger stakeholders to judge specific design decisions and to give change proposals. Without these specific triggers, stakeholders will not be able to answer the questions and it would even not be possible to generate these questions at first [24]. In Figure 2.6, an overview is given of the areas of fidelity of the single phase methodologies.

2.5.2. Implementation of the designerly software development methodology

In order to convert the underpinning theory into an implementation of the DSDM, we first focus on three aspects: (i) what procedural support is offered by the methodology, (ii) what instruments are used, and (iii) what methods are available to conduct the tasks and how to use the specific methods. This is important since the nature of the three targeted phases of the software development process are completely different, consequently different prototyping and demonstration methodologies are needed for each phase. As shown in Figure 2.7, the DSDM consists of three single-phase methodologies (namely the methodology of critical collective reflection, modular abstract prototyping, and surrogate-base prototyping) that each offer a procedural scenario, instruments and methods in a specific phase.



Framework ideation
Concept integration
System development
software development process
Requirements
Design concerns
Design options
Design decisions
Expert focus group session
Framework development
Conceptualization
Modular abstract prototype
Focus group sessions
Data evaluation
Adjusted concept design
Surrogates selection
Surrogates-based prototype
Test execution
Data evaluation
Adjusted system design
Procedural support
Instruments
Framework evaluation in critical expert session
modular abstract prototyping
surrogates-based prototyping
Methods
Critical Collective Reflection
Modular Abstract Prototyping
Surrogates-based Prototyping
Designerly software development methodology

Figure 2.7. Implementation of the DSDM

Procedure of conduct

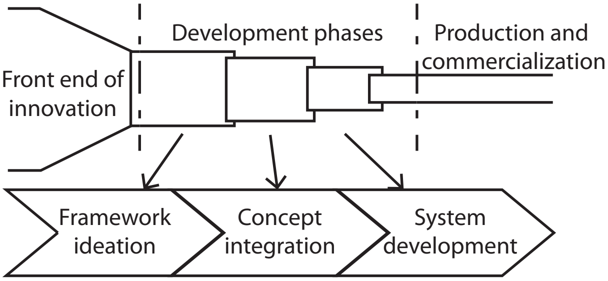


Figure 2.8. Overall process of the software development process

It has been explained above that the overall process of DSDM consist of three sub-processes. As shown in Figure 2.8, the overall methodology has been defined as a construct that provides proper support. To explain each phase on an action level: During the framework ideation phase, the critical collective reflection methodology helps converting the requirements into a design concerns. Finding possible conceptual solutions or design options, converting these into a functional and structural framework. The design decisions are evaluated in a collective critical reflection session with experts. This provides knowledge for the enhancement of the framework. In the concept integration phase, the modular abstract prototyping methodology supports the conceptualization and verification of the concept by modularly demonstrating the software concept to different stakeholders, and by enabling discussions. After data evaluation, the change proposals are used to improve the software concept. The surrogate-based prototyping methodology aims to support the system development phase. As a first step surrogate software is selected and combined to build the prototype. The testable, tangible prototype enables functionality and usability testing of the software quickly and at low cost. After data evaluation, the system design can be adjusted and improved. Further details on the specific phases can be found in respectively chapter 3, 4 and 5.

Instruments to support software development

To achieve the procedural support, in each phase different instruments are used. To give a high level expression, we introduce the most important instruments that are used as means for facilitating in each phase of the process. We can identify two types of instruments in each phase: (i) technical instruments, and (ii) organizational instruments. The main instruments used in each single-phase methodology are shown in Table 2.3.

Table 2.3. Main instruments for each phase

	technical instruments	**organizational instruments**
critical collective reflection	framework representation	expert sessions
modular abstract prototyping	modular abstract prototypes	focus group sessions
surrogate-based prototyping	surrogate-based prototypes	protocol-based software testing

Enabling methods

The three single-phase methodologies include a set of methods. The details of the specific methods are discussed in chapter 3, 4, and 5, but in this Section, we want to give a first overview. Each methodology has several design methods and research methods. Design methods are used to synthesize and develop the product, while research methods support the exploration. Both standard and specific research methods are used for interrogation, observation, experiments, interventions, aggregation, statistics and simulation. In Table 2.4, an overview is given of the most important methods used in the single-phase methodologies.

Table 2.4. Main methods used in the single-phase methodologies

	design methods	**research methods**
critical collective reflection	requirements list and morphological analysis	expert brainstorming sessions (interrogative)
modular abstract prototyping	prototyping and demonstration	focus group sessions (interrogative)
surrogate-based prototyping	prototyping and simulation	testing methods (interventional)

Criteria for goodness

To be able to justify the whole methodology and to check its logical correctness, different criteria of goodness were intended. In general, this logical correctness can be split up into: (i) reliability, (ii) consistency, and (iii) cohesion. A methodology is reliable if it has the ability to perform its required functions understated conditions for during its application. In a practical viewpoint it means that the reliability of the methodology can be interpreted in the reliability of information processing that it systematize. As criteria it implies that all elements of the methodology should perform the specified information processing functions, and they should support avoiding procedural and content-wise errors. Reliability theory and failure-mode analysis offer specific formal means to express reliability. In case of a multi-part methodology, consistency guarantees that the parts do not contain contradiction. The lack of consistency can be defined both in semantic and syntactic terms. Semantic consistency entails that the parts of the methodology follow the same model of logic and rely on inter-related (transitive) sets of information. Cohesion of a methodology is a measure of how much it can be integrated with the other parts of the overall methodology. This is an important feature of the component methodologies of a complex multi-part methodology. It also means how the various components (procedures, methods, and instruments) interoperate from a procedural and semantic point of view. For the reason that investigation and specification of some more concrete criteria of reliability, consistence and coherence would need further research, this thesis could not consider these aspects in full details, only in the validation of the DSDM methodology.

2.6. Introduction and specification of a reference case

The DSDM can be implemented when the to-be-developed software product responds to following criteria: (i) it should be a typical complex and evolving software-intensive system, (ii) it should explicitly need a stakeholder-oriented development approach, (iii) in which multiple stakeholders are involved, who all requires a high engagement. The software: (iv) should be event or information driven, instead of pure algorithmic or computation driven, and (v) multiple complexity categories should be handled. The identified reference case will be the development of a software tool to support designers in smart energy saving using ubiquitous controllers. Energy consumption and environmental impact issues are becoming a heavy concern all around the world. In order to reduce the energy consumption and impact, industrial design engineers have to use environmental friendly technologies and must fulfill environmental specifications and legislative norms. Contrary to the efforts, the progress in this direction is not optimal. The answer is somewhat obvious. Achieving optimal energy performance with electronic household products needs a wide ranging search for optimal solutions, as well as a complicated multi-criteria optimization process and decision-making.

The process of designing for sustainability is not a trivial problem due to the fact that the actual energy consumption related to them, but also the type of usage of the products, the user behavior, and a lot of other intangibles [25]. It is not a surprise that it is rather resource and time consuming to take all influencing parameters into account simultaneously. Support for designers is also needed since not only electronic household products but also consumer behaviors have become more complex. In order to detail the problem, we identified five related domains, who jointly add up to the knowledge necessary for a comprehensive understanding of the research problem. In terms of the knowledge domains, further articulation was necessary because of complexity and relevant issues regarding: (i) the challenge of energy saving in electronic appliances, (ii) the critical products and their characteristics, (iii) the opportunities of ubiquitous augmentation, (iv) energy efficiency related problems in the design process, and (v) the mathematical models for forecasting the trade-off to find the best solution.

2.6.1. The challenge of energy saving in electronic appliances for product designers

In this Section, we give an overview on how energy can be saved in electronic appliances. Before going further, we have to mention that we limited our focus to electronic household appliances. To be even more specific, we are only looking at how we can save energy during their usage phase. Several studies pointed at the fact that many household appliances have a strong environmental impact due to their high energy consumption and especially because of the amount of energy that is wasted during their usage [26]. Contrary to the efforts, the progress to reduce the amount of wasted energy is not optimal. Still plenty of household appliances are inefficient in terms of energy consumption. In literature, two types of energy saving strategies were identified for product designers: engineering and social. The engineering inspired energy saving strategies target changing product characteristics from energy consumption point of view. This can be achieved by using less energy intensive product technologies [27], such as a better engine or better isolation, or by eliminating useless energy consumption [28], such as adjusting the home heating system with a thermostat. Technology engineers play important role in energy saving by developing new product technologies (powering, controlling, materialization…) solutions. We postulate that it is an obligation of designers to be aware of these advanced solutions and to use and combine them in an efficient way.

A social approach of energy saving is changing user behavior to save energy. By education, information, legislation, etc. the user becomes aware, gets activated and motivated to change his behavior. Considering the role of the designers in the implementation of these strategies means that they should make users become aware of their energy consumption and to support them in saving energy in their products [29]. User awareness can be achieved by giving feedback on the energy consumption after and during the task [30]. Several ways of saving energy could be found in the literature, of whom an overview is shown in Figure 2.9. Designers are neither technology engineers nor can they educate people in how they have to behave with electronic products [31].

On the one hand, a designer is generally not supposed to get engaged with designing more efficient technologies, but to make the most out of possible technology combinations or adaptations by including them during design. This obviously means that they should be aware of what technologies exist, what functionality they are able to realize, and what the best ways of application are. On the other hand, as Crosby and Taylor did, we should make a distinction between consumer 'information' (e.g. specific information attached to a given product) and consumer 'education' (e.g. more generic data or training as how to judge the

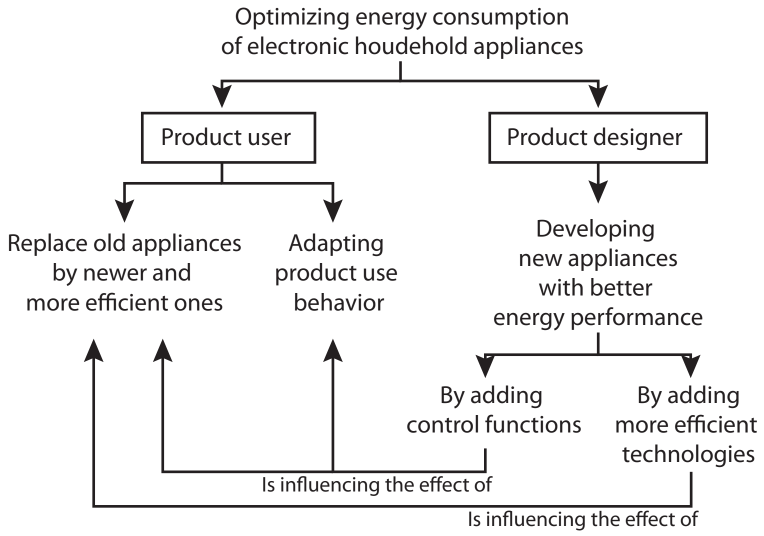


Figure 2.9. Energy saving options in electronic household appliances

performance of a product from its stated characteristics) [32]. Since designers can only inform users of their energy consumption through the design of the appliances but cannot educate them.

Literature also refers to the environmental influencing factors that have an impact on energy saving: (i) technological innovation, which can revolutionize energy consumption [33], (ii) stakeholders' cultural habits, daily routines and comfort aspects, (iii) company management, who have to realize the opportunities to improve the competitiveness of their products, and (iv) governments by introducing regulations and technical standards, by labeling and by controlling the cost of electric power through taxes.

2.6.2. Critical product categories and approaches

In this knowledge domain, we investigated what products have the highest potential for energy savings and what product characteristics are the bases for energy saving principles. As we discussed above, we limit our focus to household appliances, nevertheless we realize that the industrial appliances do have a high saving potential as well. The main motivation for this choice is (i) the amount of energy consumed by households and (ii) the aspect of comfort which make household appliances more complex to reduce energy consumption.

Energy consumption of a product

The electricity consumption of the total amount of appliances in a household is determined by two main factors: the type and number of electrical appliances in the property; and the use of these appliances by the members of the household. The family members influence the electricity use of a dwelling both by their purchase of electrical appliances and through their use of these appliances [25, 34, 35]. This can be expressed in the time of product usage. In a more articulated view, time actually means three different things: (i) duration of usage, (ii) frequency of usage, and (iii) product lifetime [29]. Longer use duration, larger use frequency, and longer lifetime mean larger total energy consumption. Longer use duration, larger use frequency, and longer lifetime mean larger total energy consumption. Keep in mind the paradox that a product with lower consumption but longer use time, and one with higher power consumption and shorten use time, may have the same total energy consumption. The use pattern is also heavily influenced by the environmental factors, such as geographical place of use, weather conditions, the wealth, and similar factors at selecting a truly critical product.

We can differentiate low-powered, such as mobile phones, shavers, toothbrushes, clocks, etc., and high-powered products, such as heating systems, air conditioners, washing machines, etc. Intuitively, high-power products are those who need critical energy conversion in product usage. Expectations are that high-powered products have a larger potential for energy saving. Most low-powered products are mobile products that operate on batteries. Since the energy provided by batteries is limited, the energy performance of this kind of products are usually optimized [36], and various technologies are used to improve the energy performance [37]. Unfortunately, we could not find useful information on energy efficiency enhancement of products with fixed placement. It may be considered as an indication that no research has been dedicated to this issue, or that consumption optimization of this category of products has not been considered an important factor yet [38].

Energy saving principles

Based on the energy-related characteristics (usage and power characteristics) of electronic household appliances we can distinguish three general energy saving opportunities: (i) the useless operation time, (ii) the useless operation, and (iii) the overload of power, (as shown in Figure 2.10.). Useless operation time can be found in products that are in active power but that are not used for a certain period of time e.g. light that is left on in the toilet when there is nobody, a television that is playing in the living room when the family is eating in the kitchen. Useless operation is when a product does something that is not needed by the user. Often this occurs as heating or lighting e.g. the light of a power button shining red when the appliance is switched off, the heat production of products components such as in a notebook. Overload of power can be located when a product is 'working harder' then needed. For instance a vacuum cleaner that sucks harder than necessary to suck the substance.

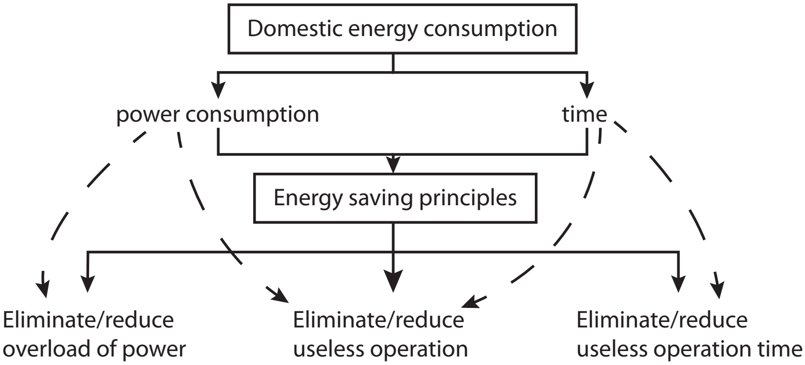

Domestic energy consumption
power consumption
time
Energy saving principles
Eliminate/reduce overload of power
Eliminate/reduce useless operation
Eliminate/reduce useless operation time

Figure 2.10. Energy saving principles

After locating the energy saving opportunities, it is also important to know how to reduce them. In contrast to designers, engineers can develop new less energy intensive product technologies, and powering and materialization solutions to eliminate this useless energy consumption [27]. Designers on the other hand should be aware of these advanced solutions, and use and combine them in an efficient way to make consumer products. If we suppose that they are using the most efficient technologies, designers can only save additional energy by adding auxiliary functions to control the products energy consumption and reducing the saving possibilities. As discussed above, control can be achieved by human control or automatically, and can vary from low intelligent functions, such as switches and buttons for more complicated and context aware controllers.

2.6.3. Opportunities of ubiquitous augmentation

Consider the following design case: a lighting solution for a public park should be designed in such a way that a reasonable energy saving is achieved, and, at the same time, a unique experience is provided for users. This is a typical case of using ubiquitous technologies as smart controllers. Based on a configuration of a large number of sensors, the period of the day, the number of people in the park, the activities of these people, and the location of the people etc. can be monitored and various lighting arrangements can be activated and deactivated. This may include attractive illuminations around the place where they are, adaptation of the light intensity, even entertainment with light effects, providing background music, and many more convenience features.

Obviously, the goal of applying ubiquitous technologies is not only to improve well-being of the people using the park, but also to reduce energy consumption. In the research of this knowledge domain, we are focusing on how adding extra smart energy saving functions can result in optimized energy consumption. The idea of using ubiquitous technologies as enablers of smart energy saving functions has emerged during the last decade. Products equipped with some level of intelligence are able to perceive their environment, can be aware of the presence of people and other agents, and can respond smartly to the needs of these agents [39].

The term ‘context-aware ubiquitous technologies’ describes a class of (still emerging) technologies that are everywhere and anywhere present to seamlessly assists us in our daily tasks, i.e. many functions are intelligently automated and can significantly contribute to the quality and sustainability of life. Information displays, computing, sensing and communication will be embedded in everyday objects and within the environment’s infrastructure [40]. For example, motion sensors can be used to increase energy efficiency: (i) for indoor and outdoor lights (the lights turn off once they stop sensing motion), (ii) motion sensor alarm: when it is triggered by activity, it activates a camera, (ii) use motion sensors to start music when you enter a room and stop it if you leave, and so on [41]. Though there seems to be an agreement on the enormous potentials they offer to change consumption patterns, the idea of using ubiquitous technologies as enablers of smart energy saving functions seems to be a grey, or even white, spot in research [42]. In [43] a functional clustering of all ubiquitous technologies is made, considering the characteristics of UTs. This should be further investigated to identify those who important in the context of smart energy saving in electronic household appliances.

2.6.4. Energy efficiency related problems in the design process

In this knowledge domain, we investigated the design process and reasoned about the influencing factors, the effort of introducing energy efficiency activities, the user behavior issue and the decision making between automation and user driven energy control. Difficulties and influencing factors in design activities: As discussed above, activities to reduce consumption and impact, are not yet embedded in designers’ daily activities, because achieving optimal energy performance with electronic household products features complexity, which is resource and time consuming to handle.

Introducing energy efficiency in the design process

To achieve involving energy efficiency in the design process, two major considerations have been proposed. Firstly, we have to be aware of the fact that energy efficiency should be seen as just one of the requirements to which the future product should come across. A general household product has multiple requirements that must be fulfilled. Thus designers must pay attention to all of them. In contrast, in this reference case we only consider the effectuation of the requirement of having an optimal energy performance during the product use. Secondly, considering this accomplishment we also have to realize in which stage in the design process the execution of energy efficient actions should be accomplished. We considered some common used models to structure the new product development process, to think through the most important phases for designing energy efficient products. [44-46]. As already mentioned above, crucial to energy efficient design is the phase in which the product specifications and requirements should be defined. Furthermore, the actual design action emerges during the concept development phases of the product. In this phase designers have to implement all requirements into their product.

User behavior influence on energy saving.

In literature, much effort is put into the motivation and education of users to change their behavior concerning energy savings. Different authors argued that the general conclusions are that end-users tend to apply energy-thrift actions if (i) they understand the benefits, (ii) they are motivated, and (iii) appropriate information–feedback techniques are applied [47, 48]. To reduce household consumption designers must combine socially and culturally sensitiveness with technically proved technologies [32]. Considering the level of automation [49-52], energy consumption can be reduced by people's decisions or by machine's decision, and the whole range between. In Figure 2.11., an overview is given of these two extremes with their most important considerations. If the responsibility is given to the user, designers must engage users in the design of control systems that they like in order to allow them to create the comfort conditions they want and which will support them, through using auxiliary technology, to reduce their energy consumption. In contrast, if users do not have any intention to reduce their energy consumption, automatically adapting the energy consumption is also a possibility.

One of the challenges of designing for sustainable behavior is that users' actions can be difficult to predict as they are driven by a complex array of internal and external influences. To minimize unpredictability and ensure compliance with energy saving goals it is possible to design highly autonomous systems which minimize or eliminate the need for human intervention completely or use constraints to prescribe actions [49]. However, by taking the decision making capability away from the user to prevent 'unsustainable' actions, we separate cause and effect. Some authors fear that without feedback on cause and effect

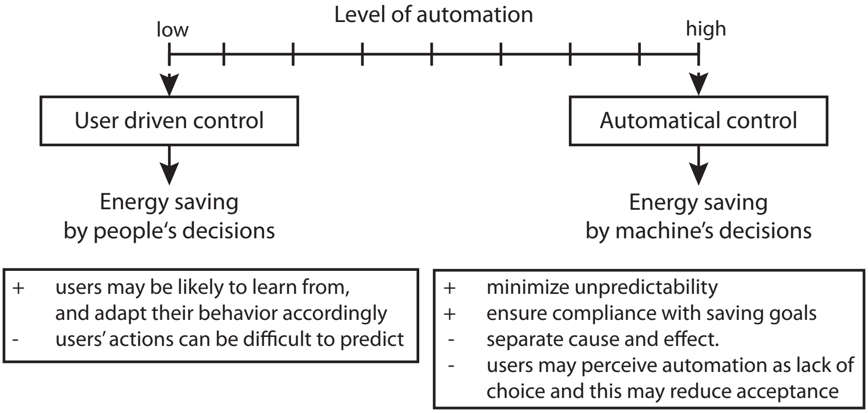

Level of automation
low
high
User driven control
Automatical control
Energy saving
by people's decisions
Energy saving
by machine's decisions
+ users may be likely to learn from, and adapt their behavior accordingly
- users' actions can be difficult to predict
+ minimize unpredictability
+ ensure compliance with saving goals
- separate cause and effect.
- users may perceive automation as lack of choice and this may reduce acceptance

Figure 2.11. Level of automation

users may be less likely to learn from, and adapt, their behavior accordingly. In addition, some authors argued that users may perceive automation as a lack of choice and this may reduce acceptance [29, 49, 53, 54].

2.6.5. Mathematical models for forecasting

In this knowledge domain, we investigated and developed the mathematical models that are needed as a basis for the software tool, because it is obvious that ubiquitous controllers cannot be considered if the extra costs are higher than the cost advantages that can be achieved by applying a smart energy saving mechanism. From an economic point of view, the necessary sensors, transmitter, and actuator units introduce extra costs, add to technical complexity, and increase energy use significantly when a large number of them are employed. Consequently, designers face a complex technological and economic trade-off problem, which seeks for a positive unbalance in terms of the additional costs of using sophisticated energy saving functions, and the amount of energy saving in a particular energy-consuming environment. In simple words, the use of sophisticated controllers means that the additional costs should be in proportion with the savings and designers must be able to find the energy saving controller or combination with the highest gains.

As a design optimization issue, economic trade-off raises the need for (i) an explicit calculation of the total costs of a product without and with the additional costs of ubiquitous controllers, (ii) an assessment of the marketing opportunities of the product with increased total price, (iii) the energy consumption (and wastes) of the product without and with the ubiquitous controllers, and (iv) the additional operational and maintenance costs. In the course of conceptualization of a product, designers have to complete these preliminary calculations, and to make decisions based on the characteristics of various alternative conceptual designs. In addition to the tangibles, which can be expressed in terms of financial means, we consider the user acceptance as an intangible, which is critical from the aspect of the appreciation of the product. In this calculation, we assume the users to be rational and so neglect the quantification of the intangibles and consider only the product cost, from the perspective of the user, the selling price of a new product, and the energy cost as determining parameters.

To estimate the trade-off, we should compare the total costs characterizing a product which is not equipped with a ubiquitous controller with the cost and appreciation of the product equipped with ubiquitous controller. From now on, we refer to the product not equipped with ubiquitous energy saving controller as the original product, and to the one equipped with this as the extended product. It should be assumed that the information about the possible energy waste and energy saving possibilities are known before the trade-off estimation. The financial trade-off for an extended product can be expressed mathematically as an optimum finding problem:

$$TO= \max(G_1, G_2, \ldots, G_n) \qquad (1)$$

Where:

TO is the trade-off result that takes the value of the maximum of financial gains which is determined by comparing the particular gains obtained for each considered extended product variant.

G_i is the financial gain (or saving) that can be achieved in the case of a particular product-controller combination (i) in comparison with the original product. The gain can be calculated as:

$$G=TPC_O-TPC_N=(PP_O+EC_O)-(PP_N+EC_N) \qquad (2)$$

Where:

G is the achievable financial gain (or saving) in the case of a new product equipped with ubiquitous energy saving controller in comparison with the initial product

TPC_O is the total product cost of the original product

TPC_N is the total product cost of the new extended product

PP_O is the product (sale) price of the original product

PP_N is the product (sale) price of the extended product

EC_O is the energy cost of the original product

EC_N is the energy cost of the extended product

To calculate the gains, we also need information about the sale prices of product variants (product costs). There have been many papers published both on quantitative estimation and on numerical calculation of sale price. Typical quantitative cost estimation methods assume that detailed design of a product has been completed [55]. Cost estimation tools to support early design are scarce and rough. However, various methods, such as case-based reasoning, decision support mechanisms, and analogical reasoning techniques have been successfully applied in a quasi-numeric or qualitative estimation of product price [56]. Typically, these techniques make use of past data to predict the costs of a new product without requiring precise information on the product itself.

Literature shows that calculation of product cost is a complicated summation with multiple unknown variables. What it means is that we can consider and use the selling price of the product as a substitute of the actual product cost. It is a frequently applied simplification [57] and this has in fact been considered in above Equation (2). In the case of an extended product, the total cost of the original product needs to be appended by the cost incurred by the applied ubiquitous energy controllers. The additional product cost components include the market price of the controller, the additional embedding cost, and the implementation cost.

Literature was investigated to see what software tools are available for energy cost calculation. What we found is that the currently available tools typically require very detailed information about the embodiment (manifestation) of the product. On the other hand, the need for supporting energy consumption and estimating the various costs in the early phase of product development has also emerged. In fact, some first steps have been made in this direction in [58]. A proper energy saving calculation should consider the hours of being in operation and in standby mode [25]. In order to make reliable estimations, we need detailed use scenarios and user behavioral patterns [59]. The most obvious measure is cost and this explains why everything is expressed in terms of money in our calculation scheme. In practice it means that the calculated energy consumption is converted to money by considering the energy prices. The energy consumption of the original product can be calculated as:

$$E_O=(OH_A\times NP_A)+(OH_L\times NP_L)+(OH_Z\times NP_Z)\times 365 \quad (3)$$

Where:

E_O is the energy consumption of the original product (expressed in kWh/year)
OH is the number of operation hours in a day for the original product
NP is the nominal power required for the operation of the initial product (expressed in kW)
A is the index of the active power mode
L is the index of the low power mode (stand-by)
Z is the index of the off or zero power mode

Furthermore,

$$\sum OH_i = 24h \quad (4)$$

For the original product, the energy consumption cost can be calculated as:

$$EC_O=E_O\times \sum(TU_p\times EP_p) \quad (5)$$

Where:

EC_O is the energy consumption cost of the original product
TU is the time in use (expressed in years)
EP is the energy price per kWh consumption
P is the time period in which the energy price is fixed
E_O is the aggregated energy consumption of the original product.

It can be assumed that equations for energy cost and energy consumption calculations remain the same for the original and the extended product only in incidental cases. Therefore, the above Equations (3) to (5) must be adapted to and specialized in terms of the descriptive variables and their relationships. Obviously, when the physical manifestation of a new product remarkably differs from the original product the variables, and consequently the respective equation will be partially or completely different. These changes may require a comprehensive redefinition of the workflow of computation and the descriptive equations [60].

2.7. Elaboration on the requirements for the reference system

To handle the complexity, a software tool is needed to support designers to support the process, offer the needed information, and execute the trade-off calculation. To build this tool, we investigated in the research of this knowledge domain different knowledge-driven processing mechanisms, which allow (i) qualitative reasoning in context, (ii) reasoning from past (design) cases, (iii) describing new technologies, and (iii) concrete design requirements. We have found a large variety of non-numerical knowledge-processing and reasoning mechanisms that have been applied either as general problem solving means, or means for specific application domains. This caused a kind of complexity, which we tried to cope with by applying a two-level survey strategy in our literature study. In the conduct of the study it meant that we first concentrated on finding and assessing the non-numerical knowledge-processing and reasoning mechanisms that have been applied in comparable tools. Afterwards, having hypothesized the appropriateness of a specific reasoning mechanism, we narrowed down our investigation to the issue of how this reasoning mechanism has been implemented and operationalized in available tools.

2.7.1. Survey of the reasoning mechanisms applied in comparable tools

According to our working strategy, the objective of our work has been the synthesis of the highly-interactive knowledge-intensive software tool. In addition, we also intended to save capacities for the necessary explorative and confirmative research actions. For these reasons, the goal was defined as finding an existing reasoning mechanism suitable for forecasting, rather than to invent a brand new reasoning mechanism. As selection and assessment criteria for non-numerical knowledge-processing and reasoning mechanisms published in the literature, we considered the specific requirements that have been formulated for the software tool based on the problem statement detailed in Section 2.4. Namely, the mechanism should (i) support representation of knowledge intensive problems, (ii) allow making inductive reasoning, and (iii) complement the thinking process of designers. Considering these requirements we have developed a reasoning model for our literature study, which identified the general and specific knowledge domains. Shown in Figure 2.12., this reasoning model leads us from general information processing mechanisms to smart reasoning mechanisms of knowledge-intensive systems, which have the capability to store and retrieve factual information, and allow for inductive reasoning.

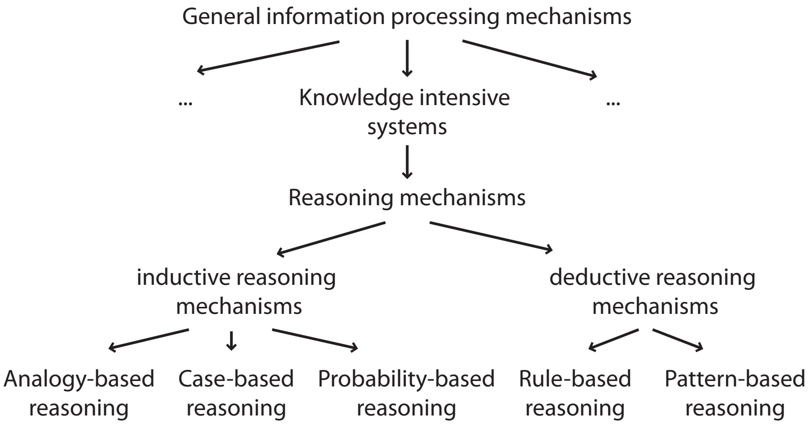

General information processing mechanisms
...
Knowledge intensive systems
...
Reasoning mechanisms
inductive reasoning mechanisms
deductive reasoning mechanisms
Analogy-based reasoning
Case-based reasoning
Probability-based reasoning
Rule-based reasoning
Pattern-based reasoning

Figure 2.12. The reasoning model applied in the literature study for reasoning mechanisms

Our untested assumption that inductive reasoning mechanisms are more appropriate for the forecasting problem than deductive reasoning mechanisms has also been considered as a selection criteria for comparable tools. Consequently, all tools using deductive reasoning mechanisms, such as expert systems and rule-based systems with backward chaining reasoning, have been excluded. The main argument behind this decision is that deductive reasoning mechanisms typically: (i) cannot consider emergent and instance-induced knowledge, (ii) need pre-programmed mechanisms for processing rules or patterns, and (iii) generates a large set of possibilities that has to be further processed. At the same time inductive reasoning procedures typically result in one relative optimum solution and allow knowledge mining in cases and or instances. Considering these facts we decided to concentrate on exploring (i) probability-based, (ii) case-based, and (iii) analogy-based reasoning mechanisms. Below, we briefly summarize the main characteristics of these approaches and argue on the appropriateness of the above mechanisms in the context of the tool to be developed.

Probability-based reasoning

Probability-based reasoning (PBR) mechanisms have been extensively studied in statistical research, spurred by a diverse range of applications, such as forecasting, pedigree analysis, troubleshooting, and medical diagnosis [61]. The aim of a probabilistic logic (or probability logic), also called evidential reasoning [62], is the extension of deductive logic to enable reasoning with uncertain statements. In other words, PBR combines the capacity of the probability theory to handle uncertainty with the capacity of deductive logic to exploit reasoning structure. Two appealing features of PBR for ITSs are its capabilities for principled synthesis of information from multiple, complex-structured observations, and for projecting beliefs about student-model variables to expectations for future observations, which can then be used for instructional decisions and, when compared with actual observations, for model improvement [61]. In the literature, we can notice several forms and application of Bayesian belief networks and probabilistic neural networks [63].

Bayesian Belief Networks (BBNs) use: (i) a graphical structure to represent causal relationships, and (ii) probability calculus to quantify these relationships and update beliefs given new information. Probabilistic neural networks were derived from BBNs and used to classify patterns based on learning from examples [64]. [65] used Bayesian networks for change impact analysis. Mislevy and Gitomer introduced an intelligent tutoring system based on probability inference [61]. Brachman combined probability with knowledge representation [66]. Other researchers used neural networks for modeling the appliance, lighting, and space-cooling energy consumption in the residential sector to determine causal relationships and energy consumption patterns [67]. As a conclusion we can say that probability-based reasoning is mostly used in tutorials, e-learning tools, and tests, but not yet in complex forecasting tools.

Case-based reasoning

Case-based reasoning (CBR) has since the beginning of 1980s been studied intensively as an approach of artificial intelligence research. It has been applied to solving problems, typically as a means of inductive reasoning based on the information/knowledge carried by individual past cases. It has been widely applied in both academic and industrial problems, such as design of mechanical devices, food design, architectural design, structural design of buildings, and product design [39]. As many other papers, we used the book of Kolodner [68] and the book of Maher et al. as a basis for the methodological review of CBR. Maher, Balachandran et al. [69], and Kuo [70] made a historical overview of the most influential publications. He discussed that among these, Schank [71] developed a theory of learning and reminding on the basis of retaining experiences in a dynamic, evolving memory structure. Kwong, Smith, and Lau [72] proposed a CBR system to determine injection molding parameters for producing a plastic part; Chiu, Chang, and Chiu [73] developed a CBR system to predict the due dates of different orders for a wafer fabrication factory. They used a k-nearest-neighbor-based CBR approach with dynamic feature weights and non-linear similarity functions to achieve performance improvement.

Veerakamolmal and Gupta [74] developed a CBR approach for automating disassembly process planning. Their approach involves procedures to initialize a case memory for different product platforms and to operate a CBR system. This approach can be used to plan disassembly processes. Chang, Liu, and Lai [75] developed a sales forecasting model by using fuzzy CBR for selecting past cases that are not similar to the current case, but that are useful for reasoning about the current case. The above authors also investigated the use of fuzzy sets and multi-criteria decision making for accurate, efficient, and flexible case retrieval in CBR with the objective of solving sales forecasting problems in PCB industries. Yang and Wang [76] presented a revised case-based reasoning algorithm to solve hierarchical criteria architecture problems based on multiple objectives decision. Pandey and Mishra [77] have developed an integrated model of CBR and combine rule-based reasoning for generating cases, and artificial neural networks for matching cases for the interpretation and diagnosis of neuromuscular diseases. Kofod-Petersen and Aamodt [39] made a CBR tool to reason about situation-aware ambient intelligence. In the field of computer design, Wang, Baek et al. [78] developed a case-base reasoned to cluster wireless networks in order to save energy. Lastly, in the domain of industrial design, Shih, Chang et al. [79] developed an intelligent evaluation approach for electronic product recycling via case-based reasoning.

Analogy-based reasoning

Analogy-based reasoning (ABR) has also frequently been used to benefit from principle-related knowledge or from past experience in solving new and different problems. It is noted that the terms CBR and ABR are sometimes used as synonyms (e.g. by Carbonell). However, CBR should be considered a form of intra-domain analogy oriented reasoning, while ABR is extern-domain of inter-domains reasoning, taking into consideration past cases from different domains [80]. Research on analogy-enabled reasoning is therefore a subfield which is concerned with mechanisms for identification and utilization of cross-domain analogies. Linsey, Wood et al. [81] developed a tool for design-by-analogy. Hall also discussed several techniques and tools in his paper: (i) a system for handling proportional analogies, which focuses on elaborating an analogical mapping between source and target descriptions; (ii) a model for analogical processes in which induction embeds analogical comparison in a general problem solving framework; (iii) the ZORBA-I system, which assists an automated proof of a target theorem by elaborating an analogy with a source proof supplied by the user; (iv) an incrementally extended reduction analogy to transfer problem solving expertise between domains; (v) a tool to transfer and repair specific problem solving methods to solve new problems in a reactive environment; (vi) a tool to transform solution paths or re-plays derivational histories in a reconstructive problem solver; and (vii) a tool which uses successful or failed problem solving cases to plan a solution for a new problem.

Concluding remarks

The above discussed three reasoning mechanisms offer various advantages and suffer from different limitations. Therefore, in order to obtain the best results in application cases, they are often used as parts of, or embedded in each other. Over the last few years, CBR has grown from a rather specific and isolated research area to a field of widespread interest [80]. Activities are not ceasing as shown by the increased number of research papers, the availability of these reasoning mechanisms in commercial products, and the growing number of reports on practical applications. Simultaneously considering this tendency and the conclusions of our previous literature study, the survey of the existing comparable tools, and the concrete objective of our research, we *hypothesized* that case-based reasoning seems to be the most appropriate reasoning mechanism for the software tool we are aiming at. This hypothesis is also underpinned by the fact that CBR offers an incremental way of including design knowledge in the support tools. Furthermore, it has been experienced by developers of design support tools that, though designers may have difficulties with generalizing their heuristics or styles of solving design problems, they can usually rather well describe previous design cases and tell stories about their decisions in various situations. It has been assumed that if this kind of reasoning is used as a basis of a computable model of design, the design tool based on it may be capable of learning from design experience and maintaining a reasonable competency in design without major programming [69]. In the next Section, we briefly analyze the general features of CBR.

2.7.2. Case-based reasoning as a knowledge processing mechanism

This reasoning mechanism lends itself to a type of knowledge-based systems whose philosophy is to use previous cases to interpret or solve a new problem [82]. A case-based reasoning system is a knowledge-based architecture, usually consisting of a case base, case management (describing and retrieving) means, and an inference mechanism. The quality of case-based reasoning depends on both the contents of the case base, and the inference mechanism. CBR suggests a model of reasoning that incorporates problem solving, understanding and learning, and integrates all of these with memory processes. Relying on previous similar cases and situations in reasoning in complex novel situations is often necessary and advantageous. CBR is a patterned process, which is often referred to as CBR cycle. Aamodt and Plaza [80] described the hierarchical structure of activities in a general CBR cycle as follows: (i) retrieve the most similar case or cases, (ii) reuse the information and knowledge in the case/cases to solve the new problem, (iii) revise the correctness and usefulness of the proposed solution, and (iv) retain the new solution in the case base for future utilization.

For the reason that problem descriptions are often incomplete, the information encapsulated in cases help make the problem descriptions more complete. In addition, inductive reasoning (or generalization) over the cases help understand and eventually solve the problems. The most important aspect of the efficiency of CBR is the usefulness of the cases from the perspectives of representation and inference. A case represents specific knowledge tied to a context. Stored in the case base, the formally represented cases may have different features than the targeted new situation. Hence it is often necessary to extrapolate from the stored cases or learn possible abstractions based on cases. Learning typically occurs as a natural consequence of recurrent reasoning. Thus, the CBR mechanism can become more efficient and more competent over time. Providing feedback, analyzing the associated explanatory reasoning and the feedback through follow-up procedures are necessary parts of the complete reasoning/learning cycle.

CBR mechanisms have been implemented in various forms depending on the objectives and fields of application. The implementations range between two extremes: (i) fully automated reasoning systems, and (ii) retrieval-only systems. Fully automated systems are developed to solve problems without any user intervention. These systems however have some means (e.g. sensors) to interact with their environment and to receive feedback on their decisions. Retrieval-only systems work interactively with the user to retrieve proper cases, which will be used to solve the problem at hand. Their role is to augment the user's knowledge, by providing cases that may be not known or the user may not aware of. In the case of retrieval-only systems, the user should come to, and will be responsible for, the actual decisions [83]. There are just very few fully automated CBR systems and practically all of them are focused on very limited domains. Most of the CBR systems suppose interaction with user and require human intervention, especially in the case reuse phase, when human interpretation is indispensable [84].

As an informed reasoning mechanism, CBR has both advantages and disadvantages. Some of the advantages are the following: (i) allows users to propose solutions to problems quickly, avoiding the time necessary to derive those answers from scratch, (ii) allows users to propose solutions in problem domains that are not completely understood by them, (iii) provides a means for users to evaluate solutions when no algorithmic method is available for assessment, (iv) can present cases that are useful for interpreting open-ended and ill-defined concepts, (v) can use the outcomes of solving similar problems in the past to warn users to avoid past mistakes, and (vi) can help users to focus on important parts of the problem in their reasoning by pointing out what features of the problem are the important ones. On the other hand, CBR also has some disadvantages: (i) users might be tempted to reuse old cases blindly, denoted by previous positive experience, but without validating them in the new situation, (ii) cases might bias users too much in their reasoning for solving a new problem, (iii) novice users are usually not informed about the choice of cases which are the most appropriate for the problem at hand, and (iv) CBR systems offer more for users than the most representative set of cases to start with.

As a conclusion, the development of a case-based trade-off forecasting mechanism seems to be not only feasible, but also beneficial from the aspect of saving time and resources related to the development of a testable prototype. It seems to be necessary to meet four basic requirements: (i) the CBR mechanism should provide sufficient support for the end users to arrive at meaningful decisions by inductive reasoning over past cases, (ii) the case base of the CBR sub-system should be extendable and maintainable by a knowledge engineer, or by the end users (product designers) themselves, (iii) the CBR sub-system should be able to store information about the new product equipped with ubiquitous controller as a case, as well as about the cases that have been used in the reasoning process, and (iv) the CBR mechanism should seamlessly complement the intuitive thinking process and creative actions of designers. These indicate that CBR should support both emergent problem modeling and informed decision making. Furthermore, CBR should be considered not only as an enabler of knowledge aggregation, but also as an enabler of experimentation, e.g. simulation of scenarios [83].

2.8. Specifications of a concept for the reference case

According to our assumptions, the software tool should leave the task of ideation and conceptualization of the ubiquitous controllers to the product designers, but it should support designers in their thinking process by contributing to the management of artifact- and technology-related knowledge to inspire the designers, simulation and evaluation of alternatives, and the completion of the calculations, which are needed for the trade-off estimation.

To be more concrete, the trade-off forecasting tool should fulfill a number of operational and use requirements. Based on the findings of our previous literature study, the most important requirements are: (i) prior to everything, the forecasting tool should provide information about available ubiquitous sensors, transmitters and networking technologies and products. This should be complemented with the capability of showing 'best practice' cases. The tool should make it possible to evaluate the behavior of smart controllers in order to underpin the designers' decisions. (ii) Furthermore, the tool should be able to support ubiquitous and collaborative design and to update its database continuously. The input and output data should be graphically visualized promptly in order to facilitate a high level interaction between the designers and the functional modules. (iii) It is assumed that the tool operates in real-time and with a high computational efficiency. Finally, (iv) the tool should be adaptable to the designers' way of working, which suggests that it should also be able to work with limited and to handle uncertain information.

Introducing a underpinning theory

Based on the analysis of the findings, we propose the following underpinning theory for the development of a trade-off forecasting software tool for smart energy saving. It seems to be proper to break down the process of calculating and forecasting trade-off into four major phases. These are visualized in Figure 2.13. The first phase involves a formal specification of the design task, and the retrieval of resembling product and usage characteristics based on this specification. The second phase consists of the cost estimation of new product concepts without and with various UTs-based, smart energy-saving control functions. In the third phase, the energy consumption is to be estimated for the product and for the control instruments, by considering various product use scenarios and various user attitude patterns. The last phase is a kind of summary of the simulation results and forecasted powering behaviors, ranking of alternatives and making decision on the most favorable control options. The process is envisaged as a close intellectual interaction between the product designer and the knowledge-intensive trade-off forecasting tool, accompanied by sophisticated visualization.

The above process structure denotes that the software tool will have four knowledge processing modules in addition to its user interface, contents visualization, and knowledge base modules. A case-based prediction of the costs, energy consumption, and waste seems to be an advantageous approach, but management of the data needs further investigations. The knowledge base module can incorporate a sufficiently large amount of past data in order to give a good prediction. In addition, the software tool should manage persona data for users and combine them with usage characteristics. The tool also enables designers to develop concepts for new ubiquitous control devices rapidly, and to estimate their production and operational costs. In an ideal case, it may also advise designers on energy saving possibilities and collect information about new technologies on the World Wide Web.

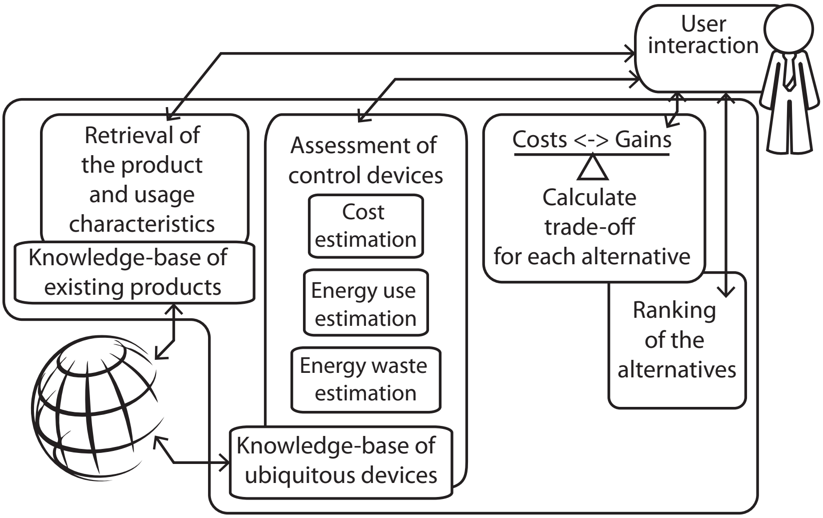

User interaction
Retrieval of the product and usage characteristics
Knowledge-base of existing products
Assessment of control devices
Cost estimation
Energy use estimation
Energy waste estimation
Knowledge-base of ubiquitous devices
Costs <-> Gains
Calculate trade-off for each alternative
Ranking of the alternatives

Figure 2.13. Graphical representation of the forecasting tool

The software tool should be implemented up to the level of a testable prototype. In order to build a first version of the prototype tool, the implementation should consider some existing programming tools, such as (i) a programmable graphical interface, (ii) a multi-part knowledge base with semantic query options, and (iii) a programmable calculation tool with macro programming facilities.

2.9. Confirmative research concerning the software tool

2.9.1. Justification of the underpinning theory of the software tool

Justification had to be carried out to verify if the developed case can be considered as an appropriate reference case. In order to know these, we refer back to the assumptions in Section 2.2.2. Is short, the reference case should have the following characteristics: A first assumption is that the reference case should be a multi-module and evolving knowledge intensive system. This assumption is obviously true since the forecasting tool is a design-support software. Design support software typically have following characteristics: (i) they are typically complex products or systems, (ii) they are based on engineering principles, (iii) they need research-oriented development projects, (iv) it is hard to formulate their requirements, and (v) they are currently often developed through non-systematic and informal procedures. An extra motivation for choosing this specific reference case is because of we are aware of the importance of such a tool in the education and support of industrial design engineers. This motivation also gives an answer to the second assumption which entails that to be able to develop the case. There should be an explicit need for a stakeholder-oriented software development approach. Furthermore, the case also answered to the need of stakeholder involvement. As multiple stakeholders are related to the software tool and those stakeholders require involvement and are necessary in the development process.

2.9.2. Internal validation

The research in this cycle was conducted for the theory forming of a reference case, in which stakeholder interaction is crucial. Literature analysis, web search, and logical reasoning were used to be able to derive the theory. In order to validate the outcomes, we have to validate (i) the concepts and (ii) the methods that were used.

Concept validation

The reference case was chosen because it is user-oriented and interaction is crucial to achieve result. Therefore it is necessary to include designers in the process because they interact with all components. Concept validation is necessary to check if indeed all components or elements are underpinning the main objective of the reference case, as these elements are operationalized in the case for further study. Hence, in this concept quality validation each element must reflect the pattern of interaction. The case is only valid if all parts are both requiring and supporting interaction. In Table 2.5, an overview is given of what interaction each element requires and what interaction it supports. We can conclude that all elements are support this interaction pattern and therefore we can conclude that the elements are underpinning the relevance of the tool, which means that the concept is valid.

Method validation

The process of how we got to the case theory and the used methods should be discussed on two levels: (i) on the level of the research actions: what we did, was it enough to conduct the research, to obtain the needed output, and (ii) on the level of the techniques used. What we did, was an investigation of the related knowledge domains and the contexts to identify, investigate and generate the problem description and the first theory of the software case. In this research we used literature study and web search in order to derive the context and domain information, logical reasoning was applied to combine all domains into a theory for the software tool. We can conclude that the research performed was enough to detect the most important aspects that are related to the software case and to make a detailed description of the theory of the problem. The techniques were applied in a linear study of all related domains. This study was chosen instead of a reflective or a comparative study because the aim was to explore the domain and retrieve a first relatively broad overview of the problem, and not to get exact detailed aspects. In order to make the study more robust, a mixed method was applied. Literature study was combined with practical results (which were found in web search) and critical reflection was used to validate the retrieved knowledge. The information found on different sources was triangulated by looking for congruence and differences in the data.

2.9.3. Specialization of the reference case

The reference case will be used in the next research cycle to test the approach of stakeholder involvement in the first phase of the software development. In this requirements

Table 2.5. Elements and argumentation on the needed interaction pattern

Element	Require interaction	Support interaction
energy consuming appliances	knowledge on past cases	offers multiple past cases
product and usage characteristics	information on past cases	offers similar cases for reasoning
ubiquitous controllers	knowledge on the possible controllers	offers an overview of controllers
energy saving solutions	information on the effect of application	offers applied cases
design process steps	need for guidance	visualization of process steps
application of the controller to the design	knowledge how to do it	shows aspects to think about
forecasting trade-off	comparison of solutions	offers calculation
cost estimation	results	calculate and shows
energy use estimation	results	calculate and shows
energy waste estimation	results	calculate and shows
case-base with existing products and ubiquitous devices	knowledge on past cases	collect, retrieve and show past case knowledge

engineering and framework development phase, the requirements generated in this phase can be directly used, without the necessity for transformation. This is because requirements engineering is a logical step in the process of framework development.

2.10. Concluding remarks

Propositions regarding the designerly software development methodology

Proposition 1:

The DSDM is underpinned by three ideas: (i) stakeholder involvement enables qualitative change proposals, (ii) managing complexity and evolvability is a critical issue, and (iii) changing fidelity during the process is a manner to handle the whole complexity.

Proposition 2:

The DSDM influences the software development process at three phases: (i) framework ideation, (ii) concept integration, and (iii) system development.

Proposition 3:

The DSDM is a multi-phase methodology that consists of three single-phase methodologies: (i) critical collective reflection, (ii) modular abstract prototyping, and (iii) surrogates-based prototyping.

Propositions regarding the reference case:

Proposition 4:

To successfully perform the research a reference case is needed

Proposition 5:

The reference case must characterize a family of software development cases

Proposition 6:

The proposed tool for smart energy saving is a good reference case

2.11. References

[1] Tang, A., and Vliet, H., (2009), “Software architecture design reasoning”, in: Software architecture knowledge management, Ali Babar, M., Dingsøyr, T., Lago, P., van Vliet, H. (Eds.), Springer Berlin Heidelberg, pp. 155-174.

[2] Awad, M., (2005), “A comparison between agile and traditional software development methodologies”, honours programme of the School of Computer Science and software Engineering, The University of Western Australia, 2005, p. 77.

[3] Abrahamsson, P., Babar, M.A., and Kruchten, P., (2010), “Agility and architecture: Can they coexist?”, Software, IEEE, Vol. 27 (2), pp. 16-22.

[4] Hammouda, I., Koskimies, K., and Mikkonen, T., (2011), “Managing concern knowledge in software systems”, International Journal of Software Engineering and Knowledge Engineering, Vol. 21 (07), pp. 957-987.

[5] Liggesmeyer, P., and Trapp, M., (2009), “Trends in embedded software engineering”, IEEE software, pp. 19-25.

[6] Sánchez, P.J., (2007), “Fundamentals of simulation modeling”, Proceedings of the Proceedings - Winter Simulation Conference, Washington, DC, pp. 54-62.

[7] Hadar, I., and Sherman, S., (2012), “Agile vs. Plan-driven perceptions of software architecture”, Proceedings of the Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on, IEEE, pp. 50-55.

[8] Booch, G., (2010), “An architectural oxymoron”, IEEE software, Vol. 27 (5), pp. 95-96.

[9] Spinellis, D., (2010), “Tools of the trade: Software tracks”, IEEE software, Vol. 27 (2), pp. 10-11.

[10] Paetsch, F., Eberlein, A., and Maurer, F., (2003), “Requirements engineering and agile software development”, Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’03), p. 6.

[11] Hirsch, M., (2002), “Making rup agile”, Proceedings of the OOPSLA 2002 Practitioners Reports, ACM, pp. 1-ff.

[12] Albers, A., and Lohmeyer, Q., (2012), “Advanced systems engineering - towards a model-based and human-centered methodology”, Proceedings of the TMCE 2012, Horváth, I., Rusák, Z., Albers, A., Behrendt, M. (Eds.), Karlsruhe Germany, p. 10.

[13] Rannikko, P., (2011), “User-centered design in agile software development”, MSc. thesis, p. 75.

[14] Eischen, K., (2002), “Software development: An outsider’s view”, Computer, Vol. 35 (5), pp. 36-44.

[15] Seffah, A., Gulliksen, J., and Desmarais, M.C., (2005), “An introduction to human-centered software engineering”, in: Human-centered software engineering—integrating usability in the software development lifecycle, Springer, pp. 3-14.

[16] Bødker, K., Kensing, F., and Simonsen, J., (2011), “Participatory design in information systems development”, in: Reframing humans in information systems development, Isomäki, H., Pekkola, S. (Eds.), Vol. 201, Springer London, pp. 115-134.

[17] Iivari, J., Isomäki, H., and Pekkola, S., (2010), “The user – the great unknown of systems development: Reasons, forms, challenges, experiences and intellectual contributions of user involvement”, Information Systems Journal, Vol. 20 (2), pp. 109-117.

[18] Majid, R.A., Noor, N.L.M., Adnan, W.A.W., and Mansor, S., (2010), “A survey on user involvement in software development life cycle from practitioner’s perspectives”, Proceedings of the Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, pp. 240-243.

[19] Alam, I., (2002), “An exploratory investigation of user involvement in new service development”, Journal of the Academy of Marketing Science, Vol. 30 (3), pp. 250-261.

[20] Iivari, N., (2004), “Enculturation of user involvement in software development organizations - an interpretive case study in the product development context”, Proceedings of the Proceedings of the third Nordic conference on Human-computer interaction, ACM, Tampere, Finland, pp. 287-296.

[21] Bannon, L., (1991), “From human factors to human actors: The role of psychology and human-computer interaction studies in system design”, Design at work: Cooperative design of computer systems, pp. 25-44.

[22] Kaulio, M.A., (1998), “Customer, consumer and user involvement in product development: A framework and a review of selected methods”, Total Quality Management, Vol. 9 (1), pp. 141-149.

[23] Ross, D.T., Goodenough, J.B., and Irvine, C.A., (1975), “Software engineering: Process, principles, and goals”, IEEE Computer, Vol. 8 (5), pp. 17-27.

[24] Roberts, D., (2005), “Coping with complexity”, in: Human-centered software engineering—integrating usability in the software development lifecycle, Springer, pp. 201-217.

[25] Firth, S., Lomas, K., Wright, A., and Wall, R., (2008), “Identifying trends in the use of domestic appliances from household electricity consumption measurements”, Energy and Buildings, Vol. 40 (5), pp. 926-936.

[26] Huisman, J., Stevels, A.L.N., and Stobbe, I., (2004), “Eco-efficiency considerations on the end-of-life of consumer electronic products”, IEEE Transactions on Electronics Packaging Manufacturing, Vol. 27 (1), pp. 9-25.

[27] Poortinga, W., Steg, L., Vlek, C., and Wiersma, G., (2003), “Household preferences for energy-saving measures: A conjoint analysis”, Journal of Economic Psychology, Vol. 24, pp. 49-64.

[28] Mok, H.S., Son, S.Y., Hong, J.H., and Kim, S., (2007), “An approach for energy-aware management in ubiquitous home network environment”, Proceedings of the 5th IFIP WG 10.2 Int. Workshop on Software Technologies for Embedded and Ubiquitous Systems, Santorini Island, pp. 293-300.

[29] Fischer, C., (2008), “Feedback on household electricity consumption: A tool for saving energy?”, Energy Efficiency, Vol. 1 (1), pp. 79-104.

[30] Brandon, G., and Lewis, A., (1999), “Reducing household energy consumption: A qualitative

and quantitative field study", Journal of Environmental Psychology, Vol. 19 (1), pp. 75-85.

[31] Morris, R., (2009), "The fundamentals of product design", AVA Publishing, p. 208.

[32] Crosbie, T., (2006), "Household energy studies: The gap between theory and method", Energy & Environment, Vol. 17 (5), pp. 735-753.

[33] Herring, H., and Roy, R., (2007), "Technological innovation, energy efficient design and the rebound effect", Technovation, Vol. 27 (4), pp. 194-203.

[34] Rodriguez, E., and Boks, C., (2005), "How design of products affects user behaviour and vice versa: The environmental implications", Proceedings of the 4th Int. Symp. on Environmentally Conscious Design and Inverse Manufacturing, Tokyo, pp. 54-61.

[35] Elias, E.W.A., Dekoninck, E.A., and Culley, S.J., (2009), "Quantifiyng the energy impacts of use: A product energy profile approach", Proceedings of the 16th CIRP International Conference on Life Cycle Engineering, University of Bath, Cairo, Egypt, p. 9.

[36] Bogliolo, A., Benini, L., Lattanzi, E., and De Micheli, G., (2004), "Specification and analysis of power-managed systems", Proceedings of the IEEE, Vol. 92 (8), pp. 1308-1345.

[37] Strijk, R., (2004), "Information technology impacts on the us energy demand profile", Proceedings of the Electronics Goes Green Berlin, pp. 1-6.

[38] Havinga, P.J.M., and Smit, G.J.M., (2000), "Design techniques for low-power systems", Journal of Systems Architecture, Vol. 46 (1), pp. 1-21.

[39] Kofod-Petersen, A., and Aamodt, A., (2009), "Case-based reasoning for situation-aware ambient intelligence: A hospital ward evaluation study", Proceedings of the ICCBR 2009, McGinty, L., Wilson, D.C. (Eds.), Springer, Berlin, Heidelberg, pp. 450 - 464.

[40] Harris, C., and Cahill, V., (2007), "An empirical study of the potential for context-aware power management", Proceedings of the 9th International Conference on Ubiquitous Computing, Innsbruck, pp. 235-252.

[41] eHow.com, (2011), "How to use motion sensors to increase energy efficiency", http://www.ehow.com/how_2386462_use-motion-sensors-home-more.html, 2011.

[42] Baldauf, M., Dustdar, S., and Rosenberg, F., (2007), "A survey on context-aware systems", Int. Journal of Ad Hoc and Ubiquitous Computing, Vol. 2 (4), pp. 263-277.

[43] Horváth, I., Opiyo, E.Z., Rusak, Z., and Koolman, A., (2009), "Towards ubiquitous design support", Proceedings of the International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/C, ASME, San Diego, California, USA, p. 10.

[44] Roozenburg, N.F.M., and Eekels, J., (1995), "Productontwerpen, structuur en methoden (2nd edition)", Uitgeverij Lemma BV, Utrecht, The Netherlands.

[45] Braet, J., and Verhaert, P., (2007), "The practice of new products and new business", Uitgeverij acco, Leuven, p. 392.

[46] Buijs, J., and Valkenburg, R., (2005), "Integrale productontwikkeling. Derde druk.", Lemma, The Hague, The Netherlands, p. 415.

[47] Wood, G., and Newborough, M., (2003), "Dynamic energy-consumption indicators for domestic appliances: Environment, behaviour and design", Energy and Buildings, Vol. 35 (8), pp. 821-841.

[48] Mansouri, I., Newborough, M., and Probert, D., (1996), "Energy consumption in uk households: Impact of domestic electrical appliances", Applied Energy, Vol. 54 (3), pp. 211-285.

[49] Lilley, D., Bhamra, T., Haines, V., and Mitchell, V., (2010), "Reducing energy use in social

housing – examining contextual design constraints and enablers", Proceedings of the 6th International Symposium on Environmentally Conscious Design and Inverse Manufacturing, Sapporo, Japan, p. 7.

[50] Parasuraman, R., Sheridan, T.B., and Wickens, C.D., (2000), "A model for types and levels of human interaction with automation", IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans., Vol. 30 (3), pp. 286-297.

[51] Das, S.K., and Cook, D.J., (2005), "Designing smart environments: A paradigm based on learning and prediction", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 3776 LNCS, Kolkata, 2005, pp. 80-90.

[52] Sauer, J., and Rüttinger, B., (2007), "Automation and decision support in interactive consumer products", Ergonomics, Vol. 50 (6), pp. 902-919.

[53] Hargreaves, T., Nye, M., and Burgess, J., (2010), "Making energy visible: A qualitative field study of how householders interact with feedback from smart energy monitors", Energy Policy, Vol. 38 (10), pp. 6111-6119.

[54] Froehlich, J., Larson, E., Gupta, S., Cohn, G., Reynolds, M., and Patel, S., (2011), "Disaggregated end-use energy sensing for the smart grid", Pervasive Computing, IEEE, Vol. 10 (1), pp. 28-39.

[55] Jiang, S., Lu, C., and Pan, S., (2007), "Product cost estimation model in early design phase based on cost cluster", Chinese Journal of Mechanical Engineering, Vol. 43 (6), pp. 205-209.

[56] Niazi, A., Dai, J.S., Balabani, S., and Seneviratne, L., (2006), "Product cost estimation: Technique classification and methodology review", Journal of Manufacturing Science and Engineering, Vol. 128 (2), pp. 563-575.

[57] Willems, M.L.A., and Stevels, A.L.N., (1995), "A financial model for environment-friendly changes in designs of electronic products", Proc. 1995 Int. Conf. Clean Electron. Products Technol., pp. 83-87.

[58] Raghavan, P., Lambrechts, A., Absar, J., Jayapala, M., Catthoor, F., and Verkest, D., (2008), "Coffee: Compiler framework for energy-aware exploration", Proceedings of the 3rd Int. Conference on High Performance Embedded Architectures and Compilers, Vol. 4917, Goteborg, pp. 193-208.

[59] Van der Vegte, W., (2009), "Scenario-driven simulation of manipulative interaction with products", Ph.D thesis, TU Delft.

[60] Satyanarayanan, M., (2001), "Pervasive computing: Vision and challenges", Personal Communications, IEEE, Vol. 8 (4), pp. 10-17.

[61] Mislevy, R.J., and Gitomer, D.H., (1995), "The role of probability-based inference in an intelligent tutoring system", User Modeling and User-Adapted Interaction, Vol. 5, Springer Netherlands, 1995, pp. 253-282.

[62] Mislevy, R.J., Almond, R.G., and Lukas, J.F., (2003), "A brief introduction to evidence centered design", 2003.

[63] Jie, S., and ZhaoHui, W., (2006), "Context reasoning technologies in ubiquitous computing environment", in: Embedded and ubiquitous computing, Sha, E., Han, S.-K., Xu, C.-Z., Kim, M., Yang, L., Xiao, B. (Eds.), Vol. 4096, Springer Berlin / Heidelberg, pp. 1027-1036.

[64] Specht, D.F., (1990), "Probabilistic neural networks", Neural Netw., Vol. 3 (1), pp. 109-118.

[65] Tang, A., Nicholson, A., Jin, Y., and Han, J., (2007), "Using bayesian belief networks for change impact analysis in architecture design", J. Syst. Softw., Vol. 80 (1), pp. 127-148.

[66] Brachman, R., (1992), “What is knowledge representation, and where is it going?”, in: Future tendencies in computer science, control and applied mathematics, Bensoussan, A., Verjus, J. (Eds.), Vol. 653, Springer Berlin / Heidelberg, pp. 187-203.

[67] Aydinalp, M., Ismet Ugursal, V., and Fung, A.S., (2002), “Modeling of the appliance, lighting, and space-cooling energy consumptions in the residential sector using neural networks”, Applied Energy, Vol. 71 (2), pp. 87-110.

[68] Kolodner, F.L., (1993), “Case-based reasoning”, Morgan Kaufmann Publisher, San Francisco.

[69] Maher, M.L., Balachandran, M.B., and Zhang, D.M., (1995), “Case-based reasoning in design”, Lawrence erlbaum associates, publishers, Mahwah, New Jersey, p. 260.

[70] Kuo, T.C., (2010), “Combination of case-based reasoning and analytical hierarchy process for providing intelligent decision support for product recycling strategies”, Expert Systems With Applications, Vol. 37 (8), pp. 5558-5563.

[71] Schank, R.C., (1983), “Dynamic memory: A theory of reminding and learning in computers and people. “, Cambridge University Press.

[72] Kwong, C.K., Smith, G.F., and Lau, W.S., (1997), “Application of case based reasoning injection moulding”, Journal of Materials Processing Technology, Vol. 63 (1-3), pp. 463-467.

[73] Chiu, C., Chang, P.C., and Chiu, N.H., (2003), “A case-base expert support system for due-date assignment in a wafer fabrication factor”, Journal of Intelligent Manufacturing, Vol. 14 (3), pp. 287-296.

[74] Veerakamolmal, P., and Gupta, S.M., (2002), “A case-based reasoning approach for automating disassembly process planning”, Journal of Intelligent Manufacturing, Vol. 13 (1), pp. 47-60.

[75] Chang, P.-C., Liu, C.-H., and Lai, R.K., (2008), “A fuzzy case-based reasoning model for sales forecasting in print circuit board industries”, Expert Systems With Applications, Vol. 34 (3), pp. 2049-2058.

[76] Yang, H.-L., and Wang, C.-S., (2009), “Recommender system for software project planning one application of revised cbr algorithm”, Expert Systems With Applications, Vol. 36 (5), pp. 8938-8945.

[77] Pandey, B., and Mishra, R.B., (2009), “An integrated intelligent computing model for the interpretation of emg based neuromuscular diseases”, Expert Systems With Applications, Vol. 36 (5), pp. 9201-9213.

[78] Wang, Y., Baek, K.W., Kim, K.T., Youn, H.Y., and Lee, H.S., (2008), “Clustering with case-based reasoning for wireless sensor network”, Proceedings of the Proceedings of the 2008 International Conference on Advanced Infocomm Technology, ICAIT ‘08, Shenzhen, pp. 124-130.

[79] Shih, L.-H., Chang, Y.-S., and Lin, Y.-T., (2006), “Intelligent evaluation approach for electronic product recycling via case-based reasoning”, Advanced Engineering Informatics, Vol. 20 (2), pp. 137-145.

[80] Aamodt, A., and Plaza, E., (1994), “Case-based reasoning: Foundational issues, methodological variations, and system approaches”, AICom - Artificial Intelligence Communications, Vol. 7 (1), pp. 39-59.

[81] Linsey, J., Wood, K., and Markman, A., (2008), “Wordtrees: A method for design-by-analogy”, Proceedings of the ASEE Annual Conference and Exposition, Conference Proceedings, Pittsburg, PA, p. 14.

[82] Yeh, A.G.O., and Shi, X., (2001), “Case-based reasoning (cbr) in development control”,

International Journal of Applied Earth Observation and Geoinformation, Vol. 3 (3), pp. 238-251.

[83] Kaster, D.S., Medeiros, C.B., and Rocha, H.V., (2005), “Supporting modeling and problem solving from precedent experiences: The role of workflows and case-based reasoning”, Environmental Modelling & Software, Vol. 20 (6), pp. 689-704.

[84] Watson, I., (2001), “Knowledge management and case-based reasoning: A perfect match?”, Proceedings of the Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, AAAI Press.

Chapter 3

Research cycle 2

Methodology of Critical Collective Reflection

3.1. Introduction

3.1.1. Objectives of the Research Cycle

In this chapter, we discuss the research conducted in Research Cycle 2, which was focusing on the first phase in the software development process, the requirements gathering and framework ideation phase. The objective of this second research cycle was to support stakeholder involvement in this first development step. In the framework ideation phase, a methodology was needed for the development of complex software systems that supports: (i) blending the knowledge of multiple domains into a consistent body of knowledge, and (ii) developing a system-level understanding and a conceptual framework of an abstract solution. Typical in this phase is the problem of incomplete context knowledge, and ill-defined and conflicting ideas. Stakeholders should be involved to argue about the expectations, needs and goals of the software and to discuss the critical design decisions.

The stakeholder involvement was achieved by getting feedback from industrial experts on the underpinning design decisions and the proposed manifestation of the framework. By blending the opinion of expert with those of the development team, we wanted to enhance the initial concept, as well as the planned implementation of the framework. Deficient requirements are recognized as the greatest single cause of failures of software projects. Hence, user participation is identified as the most crucial factor in the early requirements construction process in the software engineering literature [1]. Still, in the industry, there is a gap between intention and reality when taking the stakeholders' needs into account in the software development processes [2]. An extra difficulty emerged with the development of complex systems because stakeholders are not able to describe the full requirements. As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem [3].

3.1.2. Research methodological approach

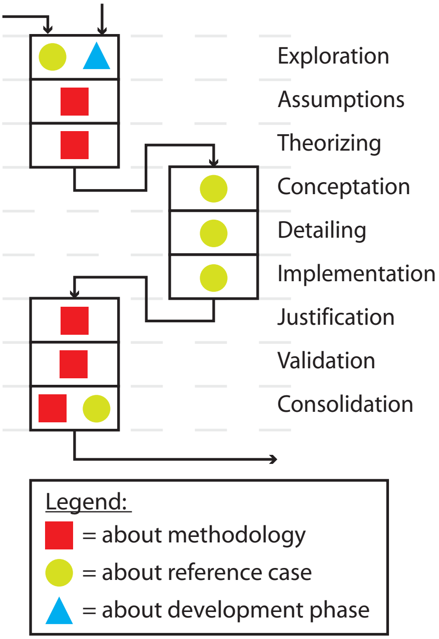


Figure 3.1. Approach of RC2

As explained in Chapter 1, and visualized in Figure 3.1., the research executed in this cycle is based on the framing methodology of design inclusive research. The exploratory research parts of this cycle involved a structured literature review on requirements engineering (Section 3.2.1), the transition of requirements (problem description) into an abstract solution (Section 3.2.2), the development of the abstract solution (Section 3.2.3), and framework development (Section 3.3.1). Investigation was carried out using the current stakeholder involvement and the related need (Section 3.3.2). Based on this information, assumptions for a new methodology were generated (Section 3.3.3) and furthermore the new methodology was build. Theory and implementation aspects of this methodology were presented in Section 3.4. The methodology was applied for the development of the reference case (Section 3.5) to be able to do confirmative experiments and studies (Section 3.6). Based on the conclusions, the justification, validation and consolidation of the methodology (Section 3.7) were achieved.

3.2. Explorative research towards requirements engineering and framework ideation

This phase can be characterized as a transition phase, in which the problem description is systematically converted into a high level solution. This high level solution can be best understood as the overview of the main building block for the software product. Since software has become increasingly complex during the history of computing, the design phase of the software life cycle has often been divided into high-level design and detailed design. Many concepts in the ordinary course (building) noted that the architecture will be useful to describe the software, which gave birth to the term “software architecture” [4]. The concept of software architecture has emerged as designing a solution to a high level of the problems of complexity. The term architecture is used to describe both the process and the result. However, in this research, to describe the result, we prefer to use the term conceptual framework, or just framework, to refer to the combination of both the functional and the structural framework of a conceptual software system. The latter is the part that in literature is called architecture. We choose the term framework as it is notional referring to the elements that are in, while the term architecture is more linked to the implementation.

At this phase of framework ideation, the objective is not yet towards implementation but as first, to show and clarify the conceptual ideas.

To support the explorative research we developed a reasoning model, which is shown in Figure 3.2. In this reasoning model, we deepen the transition from the problem definition towards the first software concept. This transition is characterized by the knowledge specification for the intended software product, and includes three aspects: (i) the requirements engineering, (ii) the abstract solution, and (iii) the framework development. Requirements can be defined as the representation of the problem definition of the to-be-developed application. In this phase the transition from this problem description into a first abstract solution is the main objective. The abstract solution should be represented through a framework. Hence, framework development should be conducted before going further to the development of the software concept (= next phase). The increased scope of design and the levels of complexity of information systems implementations are forcing towards the development and use of some logical construct or frameworks for defining and controlling the first level software solution, including high level description of the interfaces and components of the system. In any event, it likely will be necessary to develop some kind of framework for rationalizing the various architectural concepts and specifications in order to provide for clarity of professional communication, to allow for improving and integrating development methodologies and tools, and to establish credibility and confidence in the investment of systems resources [5, 6].

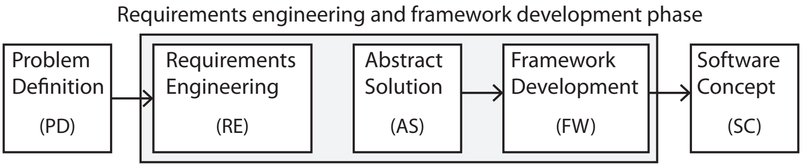

Requirements engineering and framework development phase
Problem Definition (PD)
Requirements Engineering (RE)
Abstract Solution (AS)
Framework Development (FW)
Software Concept (SC)

Figure 3.2. Reasoning model on the framework ideation phase

In the succeeding exploration, a literature review was carried out in the domain of requirements engineering to investigate the most important characteristics towards complex software-intensive systems. Next, we investigated why the problem and solution remains considered as two separate domains and why this transition is still an issue. Finally, we investigated how the abstract solutions can be generated and how to end up with a software architecture or framework.

3.2.1. Requirements engineering

Requirements engineering (RE) is concerned with identifying, modeling, communicating, and documenting the requirements of an application. Requirements describe what is to be performed but not how should be implemented. RE is an essential part in software development in both traditional and agile processes, and has following phases: elicitation, analysis, and validation [7]. The techniques used vary between the different approaches and the phases are sometimes not as clearly separated [8]. The main difference between the traditional requirements engineering and the agile approach is the amount of documentation that is required [9], together with the amount of detailing up-front [10]. Pressman emphasizes that we must design architectures explicitly, we will otherwise spend the rest of the software development project trying to make the design fit the requirements [3]. Even so, we are still unsure if all requirements and their intents are considered in the final system. A typical medium-sized software development project has about 2,500 distinct statements of requirements, which results in high complexity [4]. It makes it difficult to meet all functional and non-functional requirements when designing or modifying software architecture because each requirements statement may result in a variety of development specifications and rationales. To deepen the domain, we investigated how RE should be executed in complex systems and how to deal with the emerging issues.

Complexity of software-intensive systems

Especially in the development of software-intensive systems is RE an essential part. Obviously, there are types of software whose purpose is self-evident, and for which RE may therefore be unnecessary. Such software either does not form part of a software-intensive system or has become such a standard component that its purpose is completely understood. Due to the close link with other cyber and physical domains, the RE is much harder in the development of software components of second and third generation products and systems, because the integration of different disciplines (computer science, electrical engineering, mechanical engineering, etc.) as well as different domains (e.g., process automation, logistics, communication) are required. Thus an integrated understanding of the needs of each of the participating stakeholders in a stepwise fashion is required [11, 12].

Incomplete and conflicting knowledge issue

Conventional requirement engineering (RE) is based on the assumption that the knowledge, from which the requirements are formulated, exist a-priori, even though this knowledge is fragmented, distributed or tacit. However, considering software-intensive systems, this assumption does not hold, because incomplete knowledge of the context under which they must operate is available at design time. The RE starts with ill-defined, and often conflicting, ideas of what the proposed system is to do. The stakeholders often cannot provide all requirements in sufficient detail at the beginning of a project. Consequently agile development avoids upfront requirements gathering. But although its complete set cannot be retrieved yet, it is still important to uncover at least some requirements, because changes are more expensive later in the process. Complex systems cannot be defined from the beginning; they require an incremental, or even an evolutionary strategy [13]. The requirements should be added, refined and modified as the project progresses. Furthermore, as CPSs generally build on pre-existing infra-structure and are often constructed by integration of those, often there is no such thing as a master blueprint from the beginning [11].

Requirements engineering = purpose + contexts

Requirements engineering provides a framework for understanding the purpose of a system and the context in which it will be used [14]. A number of techniques exist for dealing with complexity. Systematic use of decomposition, abstraction, and projections are seen as the three most important general principles [15]. To obtain as much information at the start, context modeling is necessary to reduce the uncertainty. Different authors argue that investigation and reasoning is required about following contexts: space-temporal context, environmental context, personal context, task context, social context, information context [16, 17]. This context information should be used to create not only static models, which are descriptive and prescriptive, but also dynamic models of the contemplated system, showing its behaviors and effects. We have characterized software-intensive systems as being embedded in the context of human activity, and it is that activity that gives them their purpose. Therefore, a study of human activities is crucial in requirements engineering. Successful RE involves understanding of the needs of users, customers, and all other stakeholders. The fact that these stakeholders are usually multidisciplinary (including customers, visual designers, developers, QA staff, suppliers, etc.), they are often ill-defined, and application requirements change very fast, makes things even harder.

Communication and demonstration

The resulting requirements of artifacts have to be understood and usable by domain experts and other stakeholders, who may not be knowledgeable about computing. Thus requirements notations and processes must maintain a delicate balance between producing technical documents that are precise enough for downstream developers. A large variety of artifacts have been employed such as UML use cases and sequence diagrams, user interaction diagrams, task models, and navigation models [3, 6]. Many of them are not suitable to be used as communication tools with clients; others provide very informal ways of specifying the requirements, which cannot be then validated.

3.2.2. Transition of requirements engineering into an abstract solution

Differences between requirements and an abstract solution

Requirements engineering and software architecture have both become established areas of software engineering research, education, and practices [18]. Requirements engineering is concerned with discovering the purpose of a software system and the contexts in which it will be used. Software architecture is concerned with the study of the structure of software, including its topology, properties, constituent components and their relationships and patterns of combination. A problem with this focus on distinction is that the fuzzy line between what is called ‘requirements’ and what is called ‘architecture’ can be arbitrarily drawn. Commonly used criteria to tell requirements and architecture apart include ‘what’ versus ‘how’, ‘problem’ versus ‘solution’, and (a more pragmatic distinction used in industry) ‘determined before’ versus ‘determined after the contract with the customer has been signed’ [19].

Relationship between requirements and abstract solutions

Although the idea of requirements engineering as problem analysis - separated from solution considerations - seems conceptually clean, in reality this separation does not hold true. There is a rather intricate interplay between problem and solution. Choices for particular solution directions involve trade-offs that favor certain requirements over others.

The choice for a certain solution impacts not only which requirements can be satisfied, but - perhaps even more important - also which ones cannot be satisfied. At the same time, the choice for a particular solution may introduce new (sub)problems and hence new requirements [20]. Both requirements engineering and software architecture revolve around stakeholder concerns, needs, and wishes [21]. There are however, fewer consensuses on whether RE follows a constant movement between problem-finding and problem-solving, or a more creative-problem-solving-like process where problem-finding and -solving are sequential steps. A tighter integration of software architecture to requirements engineering is necessary across different subsystems due to the strong influence that architecture has on requirements engineering decisions [13]. There is an increasing recognition that the relationship between the problem space (where requirements life) and the solution space (where architectures life) is complex, fractal even, and that communication and the ability to navigate between the two spaces is necessary for real-world software development to be possible.

The issue of transition

The transition process from requirements to architectures requires expertise and extensive resources. At the moment, this process is mainly based on experience, intuition, communication, and domain knowledge of architects and designers. This makes the quality of the architecture and design heavily dependent on the skills and cognitive capabilities of developers. In other words, architecting and designing systems is still conducted in an ad-hoc, unsystematic and informal manner [22]. Moreover, many organizations struggle with defining sufficiently good software architectures. Architectures often solve the wrong problem, their importance is not seen by stakeholders, not understood by developers, or the created architecture does not support subsequent steps in the development process. Consequently, this reduces the capability of architecture to support communication, further analysis of requirements and constraints, and for system's feasibility evaluation. In addition, the architectural design becomes less appropriate for subsequent architecture-based implementation, which, in turn, results in poor quality of the final software product.

Combining methods

It is surprising how limited research has been carried out so far towards systematic architecture derivation and refinement based on requirements. Galster made an overview of the state of the art in SA & RE combining methods and concluded that no current approach provides a complete solution for direct mapping between requirements and architectural aspects [22]. Such direct mapping would require a greater focus on components already during the framework ideation phase. Even so the approaches described in the problem frames approach [23] and in the architecting requirements approach [22] have greater focus on components, they do not allow total direct mapping. Another important aspect is the classification of requirements and architectural aspects with respect to their impact on the architecture. This is only partially supported by the goal-based approach [22]. As we realized in almost all methodologies, considerable human input is required to perform the transition from requirements to architecture.

3.2.3. Development of the abstract solution

Software architecture plays an important role in managing the complex interactions and dependencies between stakeholders and serves as a reference artifact that can be used to stakeholders to share knowledge about the design of a system [19]. Architecture also facilitates early analysis of the system, especially with respect to quality attributes and maintainability of the system [24].

Evolution in the software architecture process

Past approaches defined software architecting as the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements, the composition of the elements into progressively larger subsystems, the architectural style that guides the organization, the elements and their interfaces, the collaborations, and their compositions. The focus was on components and connectors but fail to document the design decisions that produced the architecture, as well as the organizational, process, usage, functionality, performance, resilience, reuse, comprehensibility, business rationale, technological constraints, trade-offs, and aesthetics that are underlying those design decisions [7, 24]. From this point of view, a software system's architecture is no longer perceived as interacting components and connectors, but rather as a set of architectural decisions [25-27]. In addition, in agile development is software architecture an important aspect, they evolved towards just enough up-front design as an intermediate solution. Only the architectural decisions are made up-front, which allows development to get started [27].

New characterization of architecture

Architecture encompasses the setoff significant decisions about the structure and behavior of a system. These decisions will prove the hardest to undo, change, and refactor, which means to not only focus on architecture, but also interleave architectural stories and functional stories in early iterations [7]. They are cross-cutting to a great part of the whole of the design. Usually, each decision involves a number of architectural components and connectors, and influences a number of quality attributes. They are interlaced in the context of a system's architecture and they may have complex dependencies with each other. These dependencies are usually not easily understood which further hinders modeling them and analyzing them[25]. They are derived in a rich context. They result from choosing one out of several alternatives, they usually represent a trade-off, they are accompanied by a rationale, and they have positive and negative consequences on the overall quality of the system architecture [25].

Software design is derived from making many decisions. Capturing the most significant of these decisions would help convey significant insight and rationale behind the different aspects or features of the system architecture and design. However, the architectural knowledge provided by a simple enumeration of design decisions is often dry and difficult to pursue. At the moment, almost all knowledge and information about the design decisions, the architecture is based on, are implicitly embedded in the architecture, but lack a first-class representation [26]. If decisions can be browsed or visualized in an effective manner, the amount of time spent on communicating the software design with others can be reduced [28]. Defining software architecture to be a set of important design decisions suggests that we need to effectively capture, browse, and exploit such design decisions [28]. In contrast to software architecture models, architectural decisions are often not explicitly documented, and therefore eventually lost, which leads to the problems of knowledge vaporization. This contributes to some major problems, such as high-costs incurred during the development of the system: (i) high costs of changes, (ii) handling complex architectures, (iii) eroding architecture during evolution, (iv) stakeholders' miscommunication, and (v) limited reusability of the system's core assets [25, 26, 29]. These decisions should be regarded as knowledge assets that can be shared, discovered, and reused in different software development projects.

Method of making design decisions

Presently, several researchers are dealing with the documentation and representation of design decisions as formal structures within architecture, with the aim of stakeholder communication. However, we are focusing on the involvement of stakeholders in the decision making process. Therefore we need to investigate the exploratory nature of the design decisions themselves [28]. Architects often rely on their experience and intuition when making design decisions [4, 27]. Such an unstructured decision making approach has certain implication on design quality: experienced architects are more likely to make better design decisions. On the other hand, inexperienced designers may not design as well. To support design reasoning and framework development we base on a design reasoning method presented in [4], as it is giving a good overview of what many authors write. The method is based on a simple reasoning that comprises of three elements: inputs – decisions – outputs. The inputs are the requirements and goals that need to be met by a system; the decisions are the decisions made in designing the system; the outputs are the results of the design.

The reasoning model of the design process includes five steps: (i) specifying design concerns (based upon the requirements), (ii) associating design concerns by putting relevant concerns in conjunction to find a solution for them, (iii) identifying design options, (iv) evaluating design options, and (v) backtracking decisions to revise design concerns. During this process, the requirements are refined into operationalizations, which describe both design decisions and the decision rationale that is made to satisfy the established requirements [30]. Important lessons were to develop a well-structured feature list, to obtain a good understanding of the stakeholders' requirements, to use specification approaches that scale, to separate requirements and design decisions, and to establish a traceability model with a measurement process [13]. As discussed above, a tighter integration of software architecture to requirements engineering seems to be ideal across different subsystems due to the strong influence that architecture has on requirements engineering decisions. The principle of deriving design concerns is based on the idea of separation of concerns. This will reduce the complexity by separating the concerns that drive the design, and by handling them separately.

3.3. Knowledge aggregation and assumptions for collective evaluation

3.3.1. Framework development

The development of complex software products requires not only a rigorous methodology, but also information constructs that supports conceptualization and design of their problem solving and control algorithms. What we need now is an integrated understanding that will provide a full picture of a system [31]. The complex aggregate of information constructs is referred to as framework in this paper. Often there is confusion about the definition of a framework, but, in the information system community, an engineering framework is defined as a set of conceptual ideas, practices and procedures, to achieve predefined engineering goals, given a set of resources, constraints and a modeled application context. This abstract way of defining and preparing the programming of the system makes it easier to work with complexities, and to put together a bunch of components into something more useful [32]. Effective, adaptable and extendable frameworks are regarded as a key technology for future sustainable product realization approaches, in particular, the whole-life inclusive, holistic development of intelligent products, the internet-of-things, agile manufacturing, smart product bundling, closed-loop life cycle management etc. [33]. This also necessitates more research and developments towards multi-disciplinary frameworks.

One of the most active fields of use-driven framework development is multi-disciplinary design optimization. We have in several papers found examples of comparable design frameworks that deal with complicated process and complex design problems. For instance, Berends and van Tooren [34] developed a framework for their Design and Engineering Engine (DEE). The framework specification of DEE was also used to communicate about and to discuss the system. DEE is a complex system using knowledge-based engineering techniques, and aims at the automation of analysis and optimization steps in the multi-disciplinary design and optimization process of products. Barreiro et al. [35] developed a functional framework specification to support the development of an inspection integration tool. This framework was built up as a collection of information models. Based on the information captured in the framework, it was easier to put the system into operation and to make it more productive. Fan et al. developed a distributed collaborative design framework, which supported the specification of the system architecture [36]. This framework was also used as the basis of performing tests and further development. Romeiro-Hernandez et al. designed a multi-objective mathematical programming framework for a sustainability analysis of two wastewater treatment processes [37]. They used the framework to characterize three main aspects of the system, namely the flow rate, the inputs and the outputs, and used it as a basis for their case study. Based on the above discussion, it can be concluded that frameworks are used not only as structural guides for information system development, but also to communicate about the evolving system. The functional framework describes the functions of the tool as logical constructs, together with their functional relationships, in a representational (logical and figurative) manner. The functional framework also specifies the integration of all components of the developed software tools, including the system control functions and the human/system interface functions. The functional specification can be transferred into a structural representation that specifies the functional components and the information flows among the functional elements.

3.3.2. Needed stakeholder involvement

Software-intensive systems can typically be described as an interrelated set of human and system activities, supported by computer technology. Here the idea of human-centered design is crucial, because the underlying goal of an engineering process is to improve human activities in some way, rather than to build some technological artifacts [38]. As discussed in the previous chapter, different approaches of stakeholder involvement can be considered, namely stakeholders: (i) can be used as sources of obtaining sufficient amount of information about the problem at hand, (ii) can support the generation of novel concepts and ideas, (iii) can contribute to making decisions upon solutions, and (iv) can be used to validate the made decisions and obtained solutions. We should consider that the above kind of contributions assumes different degree of freedom. Taking part in generating ideas is more open and creative, than selecting from a predefined set of ideas. As regards idea generation, the main concern and challenge for companies is how to stimulate employees to reveal, disclose and transfer their innovative ideas.

Early software validation is critical to assure the optimal functional quality of the developed software. Quality of software is defined by factors, such as the completeness, correctness, consistency, feasibility, and verifiability in both the specification phase and the implementation phase. In the phase of conceptualization, on the one hand, the appropriateness and correctness of the ideas and the elements of the general concepts play an influential role. On the other hand, avoiding misinterpretation and eliminating ambiguity in communication are of significance from the perspective of stakeholders. For these reasons, safeguarding the quality of software tools in the early phases of their development has received large attention. According to the research of [39], the quality of a software concept depends on its explicit description in three behavioral dimension, namely, in terms of its functional architecture, static behaviors, and dynamic behaviors. The motivation behind the detailed elaboration of the functional architecture of the software tool is underpinned by the above findings. This provides an opportunity for validating the included concepts, the flow and details of operation, the interaction with end-users, and the fulfillment of the demands of stakeholders. Validation can be facilitated by various forms of prototyping and testing (emulated, simulated, or real) of the operation, implementation and use. It has been reported in the literature that simple methods of concept demonstration can be at least as effective in the early phases of software development, as very detailed prototypes in the later phases.

Stakeholder involvement also leads to various difficulties: (i) some approaches are not suitable for communication with clients as requirements are described in a too abstract or too specific way (and do not describe precisely interaction aspects, or put too much focus on the interface design). As a consequence, those details are discussed with customers too late. (ii) Currently, user involvement is mainly targeted for functional requirement gathering rather than non-functional requirement gathering [40], practitioners do not involve users in the non-functional requirements gathering which denotes that users are not involved in determining the kind of software used during the software development lifecycle process. (iii) Many developers are often isolated from the users and for them even identifying the users is difficult [41]. The requirements are transmitted to the development team through marketing. Finally, relatively short development cycles causes problems, there is no time for involving the users or for iteration. Literature also highlights the difficulty of getting user involvement accepted in organizations. (iv) Multitude of stakeholders: software systems have to cater for a variety of stakeholders such as business managers, owners, users and operators. These stakeholders all have their own concerns with respect to the system. Balancing these concerns and demonstrating how they are addressed is part of architecting the system. This denotes that architecture involves dealing with a broad variety of concerns and stakeholders, and has a multidisciplinary nature. Design decisions and rationales, considered different types of knowledge for representation and recording design information, might not have the same value or importance for all stakeholders. So, we should decide which type of knowledge would better fit each type of user. Groups of stakeholders, under architects' guidance, elicit these decisions, but the ultimate decision makers are the architects – often a single person or a small group [29].

3.3.3. Assumptions for an enhanced evaluation of the software framework

We concluded this exploration with the necessity for a methodology. Detailing this need based on this exploration in a literature review, we could derive following assumptions for the required methodology. Our main assumptions are:

Assumption 1:

Regarding the phase we assume that the transformation from problem to solution should be conducted by converting the problem into design concerns, retrieving design options and design rationale, making design decisions, and visualizing the outputs into one framework.

Assumption 2:

We state that a framework at the end of the ideation is needed to build a full picture of the software and to see the relationship among the design decisions for the concerns.

Assumption 3:

Regarding the stakeholder involvement, which is an important added value in the first decision making process towards a relevant solution, we assume that they should be involved to model the expectations, needs, and goals of the users and making design decisions from the very beginning of the development phase.

Assumption 4:

We state that to handle complexity in this early phase of software development, the problem should be split up into multiple smaller design concerns, which enables to reason about manageable parts and to see relationships among them.

Assumption 5:

We found the necessity for structured interrogation that should be executed with expert stakeholders in order to validate design decisions and to build a new framework.

Assumption 6:

We assume that through a focus guided analysis, collective assessment can be carried out on the different design decisions.

Assumption 7:

We also learned that it is crucial in this early phase that stakeholders develop a shared understanding among the to-be-developed software to be able to develop an acceptable product for all of them.

Assumption 8:

Consequently, we assume that stakeholder involvement in the decision making process enlarge the acceptance and creates interiorization.

Assumption 9:

We assume that at the start of this phase enough background research is conducted to offer the development team an appropriate view on the related knowledge domains and contexts.

3.4. Theory and realization of CCR

We developed a methodology based on the assumptions. The critical collective reflection (CCR) methodology enables better collective requirements engineering and framework conceptualization through direct reflection of expert stakeholders on the demonstrated proposal of the software developers. To discuss all aspects of the theory of the CCR methodology, we approached it two different perspectives: (i) in the underpinning theory we explained the idea behind the methodology on which we built the whole theory. These ideas can be seen as the implication or operationalization of the hypothesis. (ii) The implementation of the methodology, which can be split in three aspects: (a) the procedural aspect, (b) the methods and techniques used, and (c) the criteria of goodness of the methodology.

3.4.1. Underpinning theory

The Critical Collective Reflection methodology is based on a research methodology called triangulation, as it aims to compare the results of the development team and the focus group session with experts a specific form of concept validation that is in the literature. We showed the approach of CCR to achieve concept triangulation in Figure 3.3. Triangulation, originally proposed by [42], refers to the use of more than one method or approach to the investigation of a research assumption (question, hypothesis, and finding) in order to enhance confidence in the ensuing outcomes. Often distinction is made between between-method triangulation, which is interested in the validation of the data and interpretations, and within-method triangulation, which targets the validation of the applied research methods and means. Though well known in scientific research, an approach of contrasting propositions in the literature and the results of creative rationality, and expert vision and awareness in the context of early concept testing, is still a rarely used approach in activities for advanced information systems.

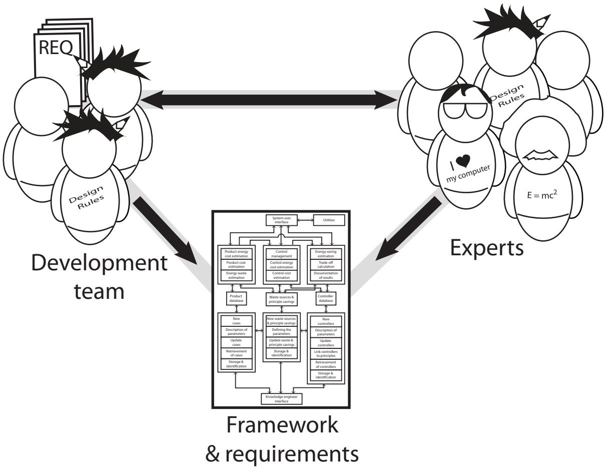


***Figure 3.3.* Concept triangulation**

The advantage of concept triangulation is contrasting formal knowledge with tacit knowledge (of users and other stakeholders) and thus to incorporate some elements of social construction of knowledge. The idea of triangulation has been extended beyond its conventional association with research methods and designs [43]. In fact, four concerns have been distinguished and formulated as strategies of triangulation: (a) data triangulation, which entails gathering data through multiple sampling techniques so that data are gathered under different assumptions, in different times, including a variety of people, and within dissimilar social situations, (b) investigator triangulation, which intends to point at the possible differences among various researchers in terms of using research means, and gathering and interpreting data, (c) theoretical triangulation, which refers to the use of more than one philosophical stance and theoretical position in conducting research and interpreting data, and (d) methodological triangulation, which refers to the concurrent use of more than one method in the various phases of research cycles and for gathering and processing data.

In CCR, triangulation features a simultaneous consideration of data triangulation and methodological triangulation. In fact, we intended to achieve a compound form of validity which is often referred to as convergent validity. In our case, it meant that triangulation has been performed by contrasting the functional framework that has been constructed based on the knowledge aggregated from requirement engineering and systematic conceptualization, and by the knowledge of independent but experienced experts, with the expectation of convergent validity. In the CCR methodology, the proposed framework is supposed to have sufficient general validity, and to be agreed upon by many people. We intended to show that the framework concept developed by the development team based on the knowledge obtained from the literature and those based on the knowledge about the possible manifestations of the tool provided by the industrial experts have a sufficient high level similarity and converge on a justified solution. Theoretically, the two bodies of knowledge should show a similarity. The conducted combined data and methodological triangulation was supposed to indicate converging results as a measure of validity.

3.4.2. Procedural aspects

In this Section we detail the procedural aspects of the CCR methodology as component of the whole DSDM. With regards to the process of CCR, since it is essential, we have to point at the fact that before modeling the requirements during the problem definition, a sufficiently deep investigation was conducted regarding the related different knowledge domains and the context of the problem. The CCR methodology is supporting the conversion of problem-description into a high-level solution. In order to be able to derive this solution or solutions, certain decisions have to be made regarding the identified design concerns. CCR is unique as it used a dual path at this moment to enrich the framework development and validate it at the same time. As shown in Figure 3.4., the process of CCR decomposes into five major steps: (i) deriving design concerns, (ii) generating design options, (iii) making design decisions, (iv) developing the functional framework, and (v) concluding about the conceptual distance.

Deriving design concerns
The objective of this step is to identify all design concerns based on current requirements' information and to select the most important ones that should be considered in this first design phase. Different types of design concerns exist [4]: (i) purposes and goals: the business goals of the system, (ii) functional requirements: functional goals of the system, (iii) non-functional requirements: quality attributes that the system must fulfill e.g. usability, performance, ... , (iv) business environment: organization and business environmental factors, (v) information systems environment: e.g. budget, schedule, expertise, etc., (vi)

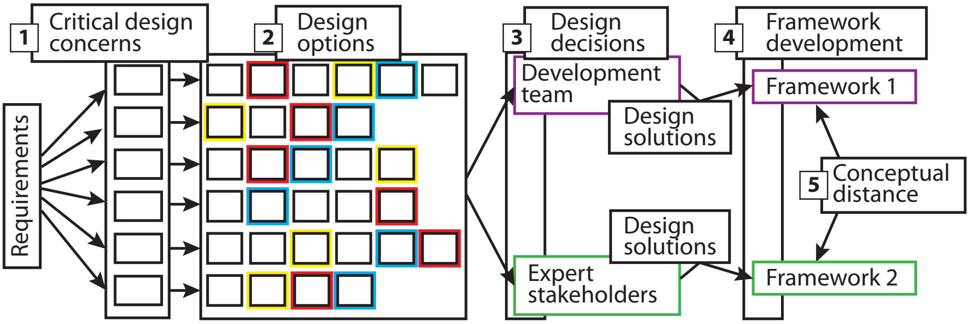


Figure 3.4. Process of framework development

technology environment: e.g. current organizational technologies and policies, and (vii) design: influence of the rest of the architecture.

Finding design options

To transfer the problems (or design concern) into solutions, there is often more than one solution. In this step, all possible options should be identified for each concern. This means that each concern should be individually investigated and possible solutions should be identified and listed. When a design decision is finally made, there will be a chosen design and may be some alternative designs. Alternatives are important because they are the evidence to show that the designers have considered more than one design option before making a decision, they also show the reason why these alternatives are not as appropriate as the chosen design.

Making design decisions and deriving design solutions

In this step, the task is to make decisions on what design options can best solve each concern, and to combine these best design options into a design solution. To make a decision, different design options may be considered, these alternative designs can help architects consider their relative pros and cons. Design rationale are used as basis to derive to make design decisions. These design rationale can have a qualitative or a quantitative nature. Qualitative design rationale can be (i) design issue: the issue to be dealt with in a decision, (ii) design assumptions, (iii) design constraints (of a technical or contextual nature), (iv) strengths and weaknesses, (v) trade-offs, or (vi) risks and non-risks. Quantitative design rationale on the other hand are defined by (i) cost: e.g. development efforts, platform support, maintenance cost and other intangibles costs such as potential legal liabilities; (ii) benefits: quantifies how well a design option may satisfy the requirements and the quality attributes; (iii) implementation risks, and (iv) outcome certainty risks.

The aimed design solution is generated by both the development team and the expert stakeholders in a participatory activity. On the one hand, continues the design team their decision making regarding their design rationale. On the other hand, the same decisions are requested from a team of expert stakeholders. Through a guided analysis, collective assessments are made on the different design decisions. Design outcome is the result of all design decisions, the chosen designs that are a part of the total solution. This chosen design either realizes the requirements of a system or it provides some design structures that are used in realizing the requirements. The design outcomes can be any design artifacts: e.g. architectural model, database model, design components and classes [4]. These design decisions can be characterized as follows [24]: a choice of an element, property, or purpose that addresses one or more concerns, and affects directly or indirectly the architecture; they may address more than one concern; they may specify existence of an architectural element, constrain the property of some elements, they provide a trace between architectural elements and concerns, and raise additional concerns [44].

Development of the functional framework

The development team immediately converts its conclusions into a first conceptual framework of the intended software. From the collaborative action of the expert stakeholder, many data is received, interpreted and converted. These conclusions are projected out in a framework of the system to consider how it will interact as a whole. Based on this received information from the expert session, an enhanced framework was conceptualized. As an indication and measure of concept validity of the framework, the conceptual distance among the features of the two versions of the framework was examined formally. For the sake of fairness, we have to mention that there is an epistemological problem. In theory, triangulation is achieved with two different researches and concepts that are not related to each other. The proposals of the focus group participants were theoretically-laden, because they did not start building up a new functional framework from scratch, but they internalized and commented on our proposed framework. This obviously means a knowledge independence problem in the research. This is somewhat relieved by applying the concept of semantic distance to express similarity or difference. If the semantic distance is low (i.e. the difference between the results of the two research activities), then the validation of the concepts is high.

3.4.3. Methods and techniques

The process of CCR involves the application of different methods and techniques to support the implementation. We identify methods to derive the design decisions, methods to do expert sessions, and methods to develop a framework.

Method to derive design decisions

The approach is to convert the requirements which are the formulations of the problem and the demand for the software into solutions. To acquire these solutions, we propose to use the method of morphological analysis [45], for which three steps must be accomplished: (i) Converting requirements in design concerns, (ii) Generating design options for each design concern, and (iii) making design decisions on the design concerns to identify one concept. This morphological analysis method is most used in hardware product design, but it is a problem-structuring and causal problem-solving technique. Nevertheless we also found it used for the design of modular systems [46]. The morphological graph should be used as instrument to visualize the design concerns with their solution options. An example of a morphological graph can be seen in Figure 3.5.

Methods for structured interrogation

The main demand of having a successful stakeholder involvement is that the participating stakeholders have the ability to discuss about solution possibilities. Therefor we assume that only expert stakeholders are eligible. Another important aspect is that the stakeholders discuss in one group in order to avoid impossibilities and contradictions between the different groups of stakeholders. Based on this reasoning, expert focus group sessions must be held to discuss the design decisions and framework development. In literature, a focus group is defined as: “a technique involving the use of in-depth group interviews in which participants are selected because they are a purposive, although not necessarily representative, sampling of a specific population, this group being focused on a given

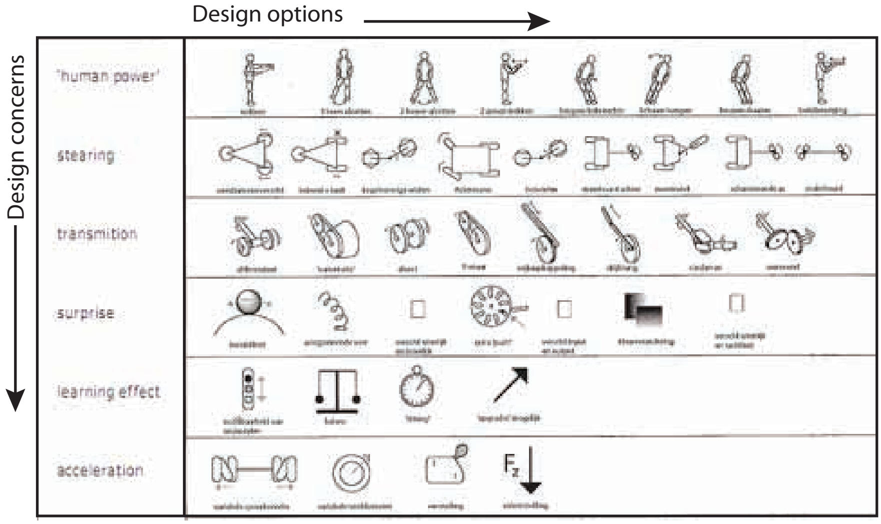


Figure 3.5. Example of a morphological chart (example from [47])

topic." [47] Participants in this type of research are, therefore, selected on the criteria that they would have something to say on the topic, are within the age-range, have similar characteristics.

Why other types of in-depth analyses are less appropriate is because the uniqueness of a focus group is its ability to generate data based on the synergy of the group interaction. The members of the group should therefore feel comfortable with each other and engage in discussion. Why experts are most appropriate is because they are comfortable talking to the interviewer and each other. For the reason to engage fully in the discussion, authors suggest the use of homogeneous groups [48, 49]. However, in this research we would suggest to invite all different types of stakeholders in the same session. Focus groups are valuable for obtaining in-depth understandings of the numerous interpretations of a particular issue of the research participants. By discussing the issue, they achieve a shared understanding and a collective assessment [50].

Methods to visualize frameworks

Since the early days of computer science, diagrams have been used to show the structure of programs. In these diagrams relations between program parts are visually encoded. Visualizations of software architectures typically deal with the structure at various levels of abstraction. A Web search with a search engine such Google for images related to the term software architecture reveals a wealth of different styles for drawing architecture (diagrams pipes and filters, layered systems, blackboards…). Most of these use ad hoc visual representations, and the semantics of the colors, nodes, icons, lines, and arrows is often unclear. To remedy this situation somewhat, one can follow general rules for the use of connectors, icons, text, color, etc. However, when it comes to building large systems with many developers, a common understanding of the architecture diagrams is a key issue, and standardized graphical notations such as UML promise to be the solution. Recently, three-dimensional visualizations of software architectures have taken real-world metaphors literally. The resulting visualizations may help in the future to convey architectural information to non-experts. The choice for visualization method is depending on the developers' skills, experiences and preferences and on the communication characteristics for stakeholders.

3.4.4. Criteria for goodness

The issue was to justify the CCR methodology. Justification means checking its logical correctness. Criteria of goodness are intended to justify the CCR methodology. In general, this logical correctness can be split up into: (i) Reliability, (ii) Consistency, and (iii) Cohesion. Converting these aspects into criteria, we concluded that: Reliability of feasibility can be measured if the methodology is executable. Consistency can be expressed by checking if there are no conflicts between the methodology components so if the methodology is internally contradiction free. Cohesion can be measured by its friendliness to other theories, if it is facilitating or enabling the implementation of other theories.

3.5. Application of the CCR to the test case

In this section, the operationalization of the CCR methodology in the application of the reference case is discussed. This application is used in the thesis as demonstration of the practical applicability of the methodology and to justify the CCR methodology. In next subsections, all procedural steps, as discussed above were executed and methods and techniques were used.

3.5.1. Deriving design concerns

As it was mentioned above, the objective of the first step was to identify the major design decisions, based upon the requirements. We could identify ten design concerns for the intended software tool:

1) The use of ubiquitous controllers to save energy: as the objective of the tool is to support smart energy saving using ubiquitous controllers, one of the most important considerations is to investigate how the controllers could be used to save energy and what information is needed to choose a particular controller.
2) The trade-off calculation: identification of applicable ubiquitous controllers is the first step, next the tool should support in the identification of the best controller. A trade-off calculation should be executed to know what controller of combination of controllers will bring the highest savings.
3) The application circumstances in which products can be: to support the designers, we have to investigate the context in which he works and find the best manner to support them. It should also be defined when the designer should start using the tool and what information should be available.

4) The required design support: a solution has to be found on how the designer will be supported by the software. Which actions are supported by the software tool, how do they support the designer, and what information is given.
5) The main functional support actions (product specifications, ubiquitous saving possibilities, control specifications, forecasting energy saving, documentation) have to be further detailed and decisions have to be made about the order and the content of the actions.
6) The case-based reasoning principle: from previous research, we found that experts often reason by comparing the current problem with their past experiences and extrapolate these to find a solution. We have to verify if this case-based reasoning principle might be applicable in this tool as well. In addition, we have to decide how this method could be used in the tool to support the designers.
7) The user-thinking/actions in the tool use: in addition to the tool actions, also the designers' actions have to be designed. What should the designer do while using the tool. What information should be inserted, what mental or physical actions should be done? How is the interaction between the tool and the user?
8) The database-concern: as we assume that the tool will be knowledge-based, we have to solve the data management.
9) The up-to-dateness: technological evolution of ubiquitous controllers is very fast. Consequently the up-to-dateness of the tool is very important. We have to identify how these improvements, changes, additions, etc. can be retrieved and converted into the tool.
10) The selection of electronic household products: on what products should the tool focus? As the tool should start with a small focus to test its efficiency before enlarging, the best products with highest potential should be identified.

3.5.2. Designers' decisions and first framework

We see the problem solving associated with the design process of energy-intensive household appliances as an integral process of retrospective analysis of past and existing solutions, their elements and performance, combined with the creative synthesis of new solutions with a view to the design requirements, and assessment of the solution opportunities against specific criteria. It has been found that neither the general functional specification, nor the energy-control component information alone is sufficient for a holistic development of energy usage aware household appliances [51]. The lack of information about context dependency of technological solution or the use style and habits of end users can be compensated for by reusing product knowledge and technical information that are embedded in past cases.

As explained above, the developed tool is supposed to provide designers with knowledge about the manifestation of past designs, and to extrapolate from this knowledge. For this reason two main functional constituents have been defined for the tool, namely: (i) the procedural information/knowledge processing components, and (ii) the various case base management components. They are shown in the left column and in the right column, respectively, in Figure 3.6. As shown, these are functionally integrated in the framework of the software tool through knowledge preparation and processing tasks. Accordingly, two different interfaces have been specified in the framework: one for the using the system functionality by the end-users (household appliance designers), and another for the case base management for the knowledge engineers. The functional framework reflects case-based reasoning oriented architecture, in which the various case bases play a central role. They enable forecasting the highest savings for each possible design, having different control solutions.

The functions of the procedural components of the system have been assigned to four modules, while the functions of the knowledge engineering components to three modules. Not only the past products, but also different ubiquitous control technologies and solutions will be represented as cases, and can be retrieved from the specific case bases. In addition to the product case base and the ubiquitous device case base, a knowledge base of waste preventing principles has been integrated into the functional framework. Neither meticulous qualitative information analysis, nor exact quantitative calculations are needed, but insightful estimations with the purpose to select and forecast from the best matching alternative. This indicates that the quality of the result depends on the number of cases, the relevance of the cases for the design task at hand, the descriptive and predictive quality of the cases, and the appropriateness of the software tool for the specific application.

The forecasting software tool will assist the end-users (designers) to select the best controller for the specific product in four steps Firstly, the product characteristics should be estimated in order to know how much energy the future product will consume, how much it will cost and were in the product and in its usage the waste could be located (Module 1 in Figure 3.6)). Based on the possible waste, principle savings must be searched and defined. In order to support the search for possible energy controllers, minimum and maximum parameters should be defined, e.g. the ubiquitous controller should be able to detect whether a person is in a range of maximum two meter. In the second step of the tool usage, these possible controllers are searched (Module 2 in Figure 3.6). The third step is to estimate the characteristics for each possible controller (Module 3 in Figure 3.6). These are needed to calculate the energy savings of each potential controller in a household appliance with a certain use scenario in the fourth step (Module 4 in Figure 3.6). All these pieces of information are needed to estimate trade-off. The results of the estimated trade-off for the particular solution alternatives should be ranked, and communicated to the designer, to facilitate the final decision on the best controller solution.

3.5.3. The expert session

Based on the decisions made by the development team we could derive following questions that should be discussed by the experts. We converted them into the following propositions: (1) Thinking of ubiquitous controllers is a good strategy for advancement in energy saving. (2) Trade-off calculation is the right approach for this knowledge intensive tool. (3) Application of different ubiquitous control functions in household appliances requires the

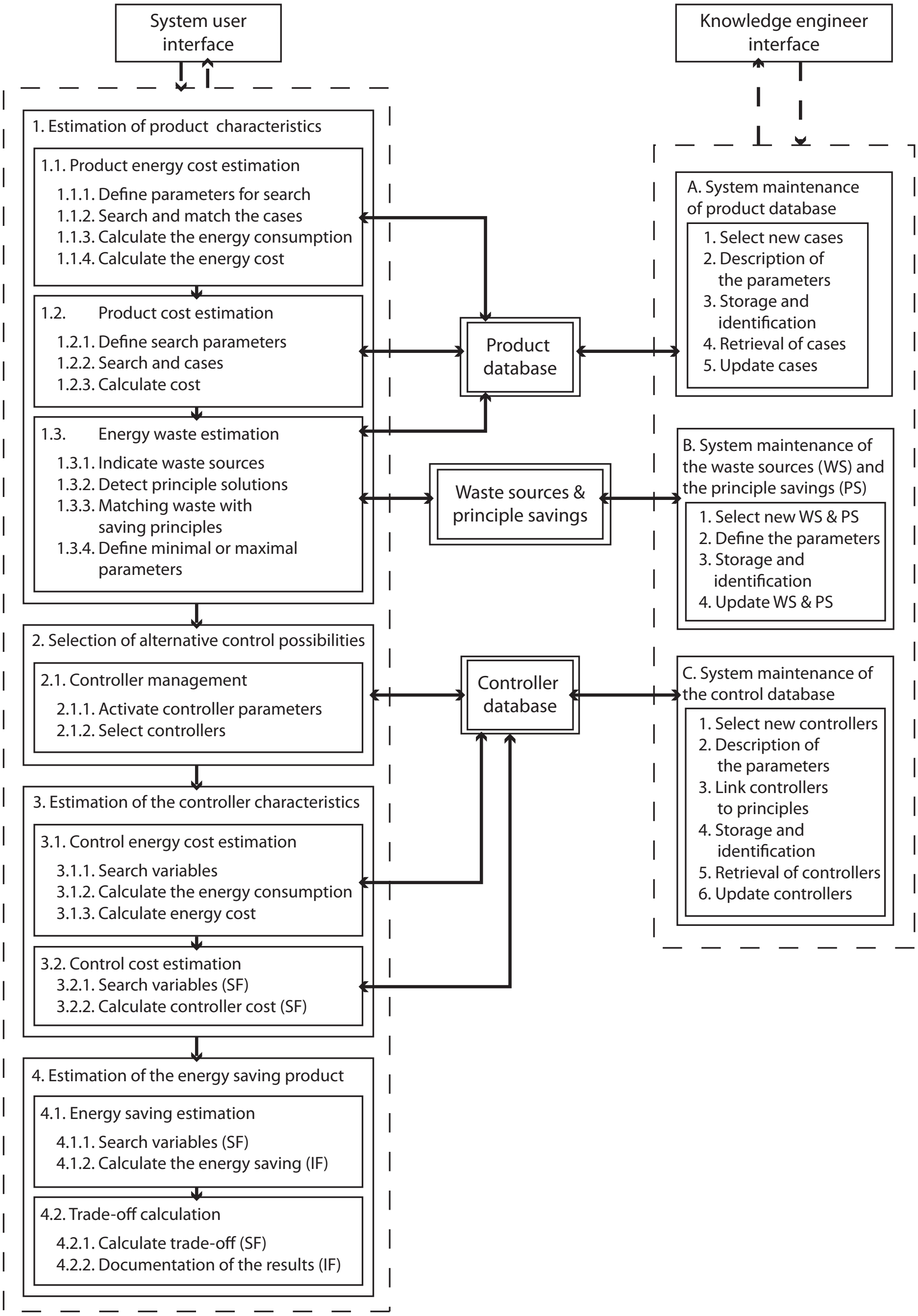

System user interface
Knowledge engineer interface
1. Estimation of product characteristics
1.1. Product energy cost estimation
1.1.1. Define parameters for search
1.1.2. Search and match the cases
1.1.3. Calculate the energy consumption
1.1.4. Calculate the energy cost
1.2. Product cost estimation
1.2.1. Define search parameters
1.2.2. Search and cases
1.2.3. Calculate cost
1.3. Energy waste estimation
1.3.1. Indicate waste sources
1.3.2. Detect principle solutions
1.3.3. Matching waste with saving principles
1.3.4. Define minimal or maximal parameters
2. Selection of alternative control possibilities
2.1. Controller management
2.1.1. Activate controller parameters
2.1.2. Select controllers
3. Estimation of the controller characteristics
3.1. Control energy cost estimation
3.1.1. Search variables
3.1.2. Calculate the energy consumption
3.1.3. Calculate energy cost
3.2. Control cost estimation
3.2.1. Search variables (SF)
3.2.2. Calculate controller cost (SF)
4. Estimation of the energy saving product
4.1. Energy saving estimation
4.1.1. Search variables (SF)
4.1.2. Calculate the energy saving (IF)
4.2. Trade-off calculation
4.2.1. Calculate trade-off (SF)
4.2.2. Documentation of the results (IF)
Product database
Waste sources & principle savings
Controller database
A. System maintenance of product database
1. Select new cases
2. Description of the parameters
3. Storage and identification
4. Retrieval of cases
5. Update cases
B. System maintenance of the waste sources (WS) and the principle savings (PS)
1. Select new WS & PS
2. Define the parameters
3. Storage and identification
4. Update WS & PS
C. System maintenance of the control database
1. Select new controllers
2. Description of the parameters
3. Link controllers to principles
4. Storage and identification
5. Retrieval of controllers
6. Update controllers

Figure 3.6. Scheme of the functional framework

consideration of the application circumstances. (4) The design support tool should operate in a context of information ambiguity and incompleteness. (5) The function structure is useful with general functions: product specifications, ubiquitous saving possibilities, control specifications, forecasting energy saving, documentation, (6) Case-based reasoning method is the best approach because it provides full product information based on past product cases. (7) The design support tool needs informal and subjective design decisions. (8) The knowledge intensive design support tool should be based on multi-functional databases. (9) The design support tool requires two kinds of users, the end-user and the knowledge engineer of the tool; and (10) A selection of the electronic household products that have the highest potential for energy saving, should be considered within this research

The objective of the focus group session was to obtain the opinion of the experts concerning the initial problem statement and solutions for the tool. People were selected by following stratified sampling, or, to be more concrete, the type of the expertise was the basis of inviting the experts. We invited four types of experts to take part in the discussion session: (1) energy experts and experts on sustainable design, (2) information system experts, (3) application design (electronic design) experts, and (4) system methodology experts. This variation of expertise was necessary to gain a comprehensive opinion on the tool. The focus group research comprised three phases: (i) it started with a preparation phase, (ii) then the discussion session was completed, and (iii) finally, the raw data were consolidated and processed. To prepare the session we invited the expert by e-mails, and prepared them for the discussion session by sending a description of the research topic. Since the session took place at an international symposium, we also invited them to take part in a presentation on the results of the previous literature study and the first concept of the functional framework. The two hour long focus group session was held as a special workshop session at the Tools and Methods of Competitive Engineering Symposium in Italy, in April 2010. In total, fourteen experts were recruited from all round the world to participate in the session: half of them came from Europe and there were two American, three African and three Asian experts involved.

The session was conducted according to the following scenario: At the start of the focus group session, all experts received a document with the most important discussion topics in the form of propositions. The first half hour was spent on a media assisted presentation of the functional framework, following the order of the written propositions. All experts were asked to write down whether they agreed or disagreed to the discussed propositions. The disagreements were noted and the number of disagreements was recorded. In the following one and half hour long discussion, the disagreed propositions were discussed in the order that was suggested by the ascending number of rejections. That is, the discussion started with the debate over the propositions that most experts disagreed upon. In the second part of the discussion, we shifted from the discussion of the contents of the research to the methodological approach that is shown in the Introduction. To facilitate after-event processing of the data, everything was recorded on video, as well as on voice tape, and has been transcribed after the workshop. The analysis of the data received from the experts followed the principles generally known from the relevant literature [48, 52]. The whole focus group session was typed and reorganized, first according to the propositions, and afterwards according to the topic. The text was also indexed by keywords.

3.5.4. Experts' decisions

The expert's decisions were retrieved by analyzing the focus group results. These results of the focus were both quantitative and qualitative. On the one hand, we recorded the numbers of agreements and disagreements, which are shown in Table 3.1. On the other hand, we have interpreted the outcomes of the discussions to obtain the meaning of the feedback for further reasoning. As we can see in Table 3.1, the concepts have been validation at a rather high level. The experts were not attacking the fundamentals, but suggested refining and enhancing the concept.

Table 3.1. Agreements and disagreements of the experts with the propositions

*AN	*+	*-	*0
1	11,6	2,4	0
2	12	1	1
3	11,5	1,5	1
4	10,4	2,6	1
5	9,25	4,75	0
6.	11	3	0
7	9,5	4,5	0
8	9	4	1
9	9	4	1
10	10	3	1

*AN = assumption number; *+ = agreed; *- = disagreed; *0 = no opinion

After the quantitative evaluation of the results related to the propositions (Figure 3.7), we completed a qualitative evaluation of the discussion part of the focus group session. The discussed topics are shown in the mind-map in Figure 3.8. To achieve a structured qualitative evaluation of the discussion topics, we established four semantic categories. These are: (i) agreements and confirmations (green-symbol), (ii) disagreements and explanations (red cross-symbol), (iii) extra information and refinements (gears-symbol), and (iv) out-of-context topics (red forbidding-symbol). All topics of the discussion were classified into these semantic categories. As a concise summary: Experts agreed upon the fact that using a control function for energy saving is a good strategy and that designers need "thinking through this". Concerning usability, they agreed that the behavior of the users of household appliances is very important, and advised to do further research on this particular topic to investigate how users behave and work with specific electronic household appliances. This is the same as their position regarding the designers and the controllers. They agreed with the propositions, but also suggested to do more detailed research to underpin the concept and define the contents of the tool. With respect to the tool, the opinions were more divided. On the one hand, the experts wanted to have a

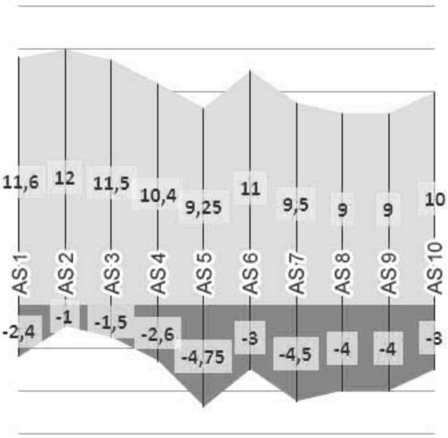


Figure 3.7. Comparison of agreements with the disagreements

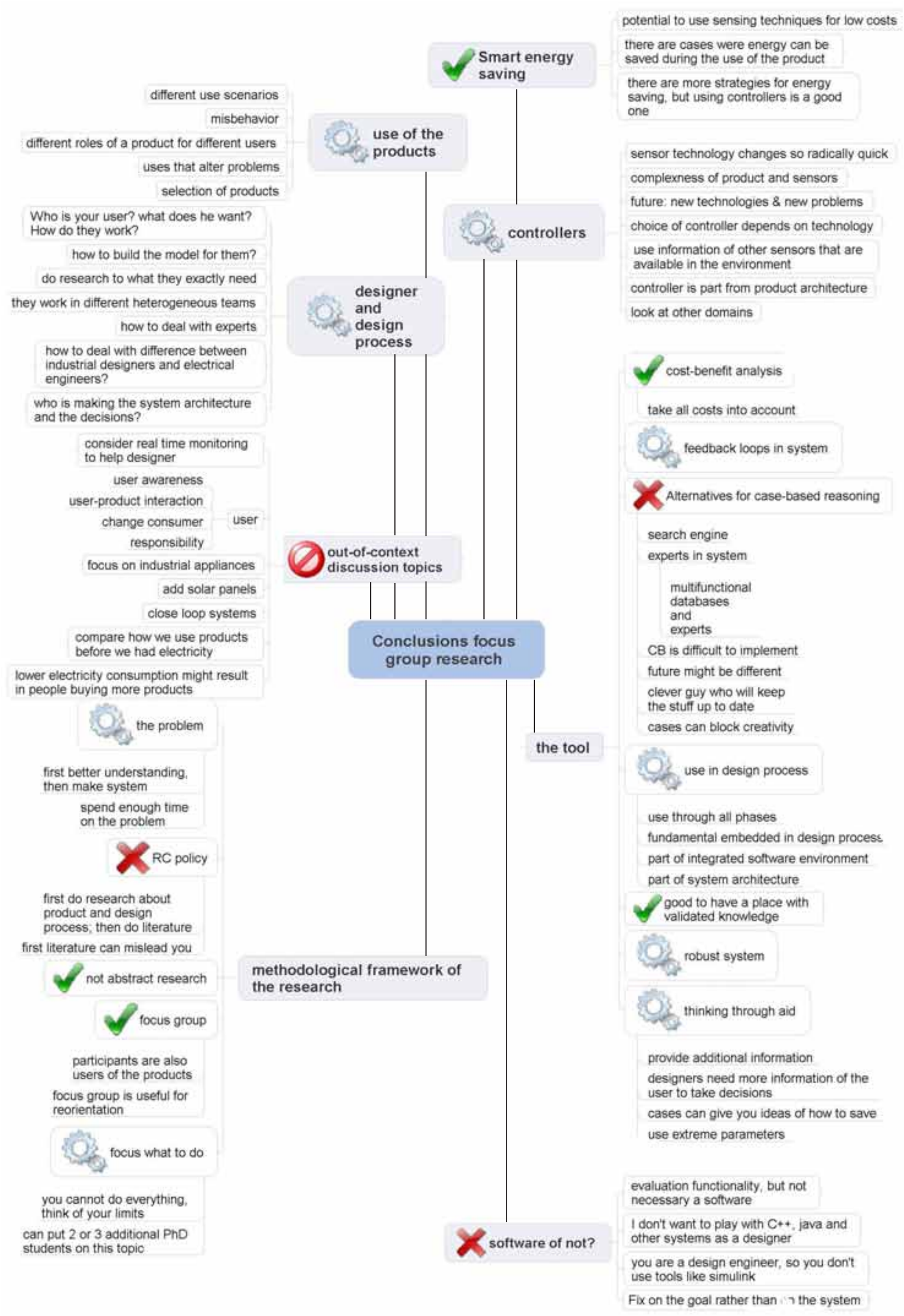

Conclusions focus group research
Smart energy saving
potential to use sensing techniques for low costs
there are cases were energy can be saved during the use of the product
there are more strategies for energy saving, but using controllers is a good one
use of the products
different use scenarios
misbehavior
different roles of a product for different users
uses that alter problems
selection of products
controllers
sensor technology changes so radically quick
complexness of product and sensors
future: new technologies & new problems
choice of controller depends on technology
use information of other sensors that are available in the environment
controller is part from product architecture
look at other domains
designer and design process
Who is your user? what does he want? How do they work?
how to build the model for them?
do research to what they exactly need
they work in different heterogeneous teams
how to deal with experts
how to deal with difference between industrial designers and electrical engineers?
who is making the system architecture and the decisions?
consider real time monitoring to help designer
out-of-context discussion topics
user awareness
user-product interaction
change consumer
user
responsibility
focus on industrial appliances
add solar panels
close loop systems
compare how we use products before we had electricity
lower electricity consumption might result in people buying more products
the tool
cost-benefit analysis
take all costs into account
feedback loops in system
Alternatives for case-based reasoning
search engine
experts in system
multifunctional databases and experts
CB is difficult to implement
future might be different
clever guy who will keep the stuff up to date
cases can block creativity
use in design process
use through all phases
fundamental embedded in design process
part of integrated software environment
part of system architecture
good to have a place with validated knowledge
robust system
thinking through aid
provide additional information
designers need more information of the user to take decisions
cases can give you ideas of how to save
use extreme parameters
methodological framework of the research
the problem
first better understanding, then make system
spend enough time on the problem
RC policy
first do research about product and design process; then do literature
first literature can mislead you
not abstract research
focus group
participants are also users of the products
focus group is useful for reorientation
focus what to do
you cannot do everything, think of your limits
can put 2 or 3 additional PhD students on this topic
software of not?
evaluation functionality, but not necessary a software
I don't want to play with C++, java and other systems as a designer
you are a design engineer, so you don't use tools like simulink
Fix on the goal rather than on the system

Figure 3.8. Conclusions of the focus group discussion

robust system that can be used through the whole design process, and, on the other hand, there were some experts who did not believe in the case-based

reasoning method. Moreover, some experts were not sure whether it should be a software tool, or not, at all. However, they did in fact not foresee or proposed any alternatives.

The post-event qualitative assessment pointed at the fact that the opinions of the experts on the research methodological framework were also divided. It seems that doing this research in a deductive way was rather unknown and unusual to some of the experts. Some suggested to complete first all needed empirical research studies, and afterwards to start the development of a tool. We should mention that we generally followed the design inclusive research methodology, as explained in Section 1.3, which suggests: (i) to start with explorative studies and to arrive at an explanatory theory, (ii) develop the concept and a testable prototype of the tool, and (iii) to do confirmative research through testing the tool with potential users and other stakeholders. Our internet based search and literature study

were the main elements of the explorative study. However, the expert suggested looking into how the designers work with ubiquitous automation and how they would work with or adapt the tool, before continuing with the development of the tool. A we explained earlier, this experiments served for early concept (functional framework) testing, before developing more detailed abstract or functional prototypes for comprehensive user testing. In fact, this has been scheduled for Research Cycle 3, but it most probably was not communicated clearly enough during the discussion session. We also have to mention as part of our conclusion that perhaps the most difficult part of doing a focus groups session was to keep the experts focused on the scope of the research and on the objective of the session. This might be for the reason that most of the participants also wanted to show how much they were experienced with adjacent themes

3.5.5. Changes and improvements introduced in the functional framework

After the quantitative and qualitative assessment of the outcome of the focus group session, we converted the obtained knowledge to improvement opportunities for the functional framework. The objective was to revisit the original framework again and to introduce functional and structural improvements before abstract or functional prototyping. As compared in Figure 3.7, a significant number of experts agreed on the initial functional propositions (blue/top), than of those who did not (red/bottom). Nevertheless, many of their critical statements inspired us to introduce the proposed changes or ideate further points for improvements. As shown, propositions 1 to 4, which are concerning the general idea of the tool, were confirmed also by the experts, who did not think of a complete paradigm change. For propositions 5 to 9, the experts made several comments, in particular concerning the transparency and simplicity of the tool. Based on the comments, structural improvements have been introduced, as shown in Figure 3.9. The generic working principle of the design support tool has not been changed, that is, the CBR methodology will be used to retrieve knowledge about past product cases and this will be availed for the designers of

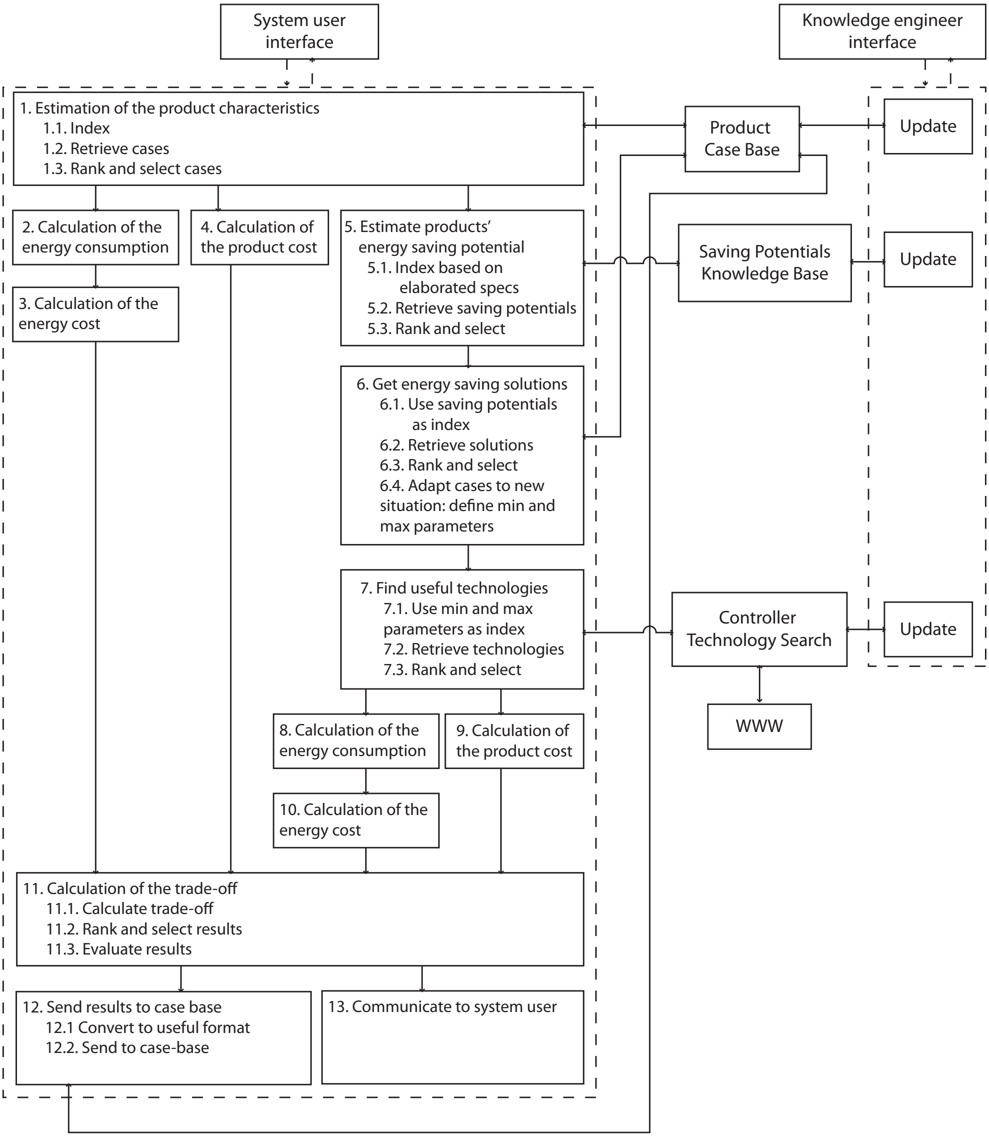

System user interface
Knowledge engineer interface
1. Estimation of the product characteristics
1.1. Index
1.2. Retrieve cases
1.3. Rank and select cases
2. Calculation of the energy consumption
3. Calculation of the energy cost
4. Calculation of the product cost
5. Estimate products' energy saving potential
5.1. Index based on elaborated specs
5.2. Retrieve saving potentials
5.3. Rank and select
6. Get energy saving solutions
6.1. Use saving potentials as index
6.2. Retrieve solutions
6.3. Rank and select
6.4. Adapt cases to new situation: define min and max parameters
7. Find useful technologies
7.1. Use min and max parameters as index
7.2. Retrieve technologies
7.3. Rank and select
8. Calculation of the energy consumption
9. Calculation of the product cost
10. Calculation of the energy cost
11. Calculation of the trade-off
11.1. Calculate trade-off
11.2. Rank and select results
11.3. Evaluate results
12. Send results to case base
12.1 Convert to useful format
12.2. Send to case-base
13. Communicate to system user
Product Case Base
Update
Saving Potentials Knowledge Base
Update
Controller Technology Search
Update
WWW

Figure 3.9. The enhanced functional framework

ubiquitous energy controllers.

3.5.6. Detailing the major structural components

Due to the functional complexity of the software tool, and the space limitation in this paper, we can discuss only one major (representative) structural components of the software tool here: estimation of the product characteristics (Module 1 in Figure 3.9), including: (ii) estimation of the product energy costs (Block 3 in Figure 3.9), (ii) estimation of the product costs (Block 4 in Figure 3.9), and (iii) estimation of the energy waste of the product (Block 5 in Figure 3.9). These components could be selected as representatives of the whole system, because they include the most cardinal and repetitive actions and interactions. They are also closely connected to the forerunning calculation for the trade-off estimation, discussed in Section 2.6.5.

The product energy cost estimation process involves four functionally different sub-components (Figure 3.10). With the help of the first sub-component, the end-user selects the relevant parameters to find cases that match his 'to-be-developed' product. These parameters are applied as search filters for the cases stored in the product database. When the best matching past products have been found, their functional parameters and values will be used to calculate the estimated energy consumption of the product. In order to provide a reasonable estimate of the energy costs of the product, the calculated energy consumption is combined with the information about the product lifecycle and use circumstances. These latter data can be retrieved from the selected cases in the product database, or obtained from the user based on direct input.

The product cost calculation process also involves four sub-components (Figure 3.11). The first sub-component deals with the case application strategy. This means that using this resource the user can choose to use the same cases which have been used to calculate the product costs, or new cases. The procedure of selecting new cases is similar to the procedure

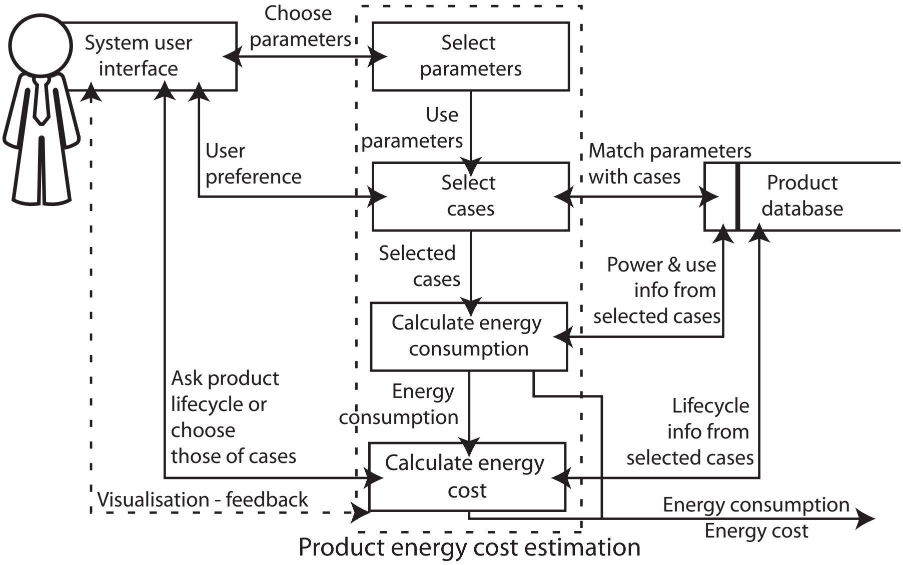


Figure 3.10. Estimation of the product energy cost module

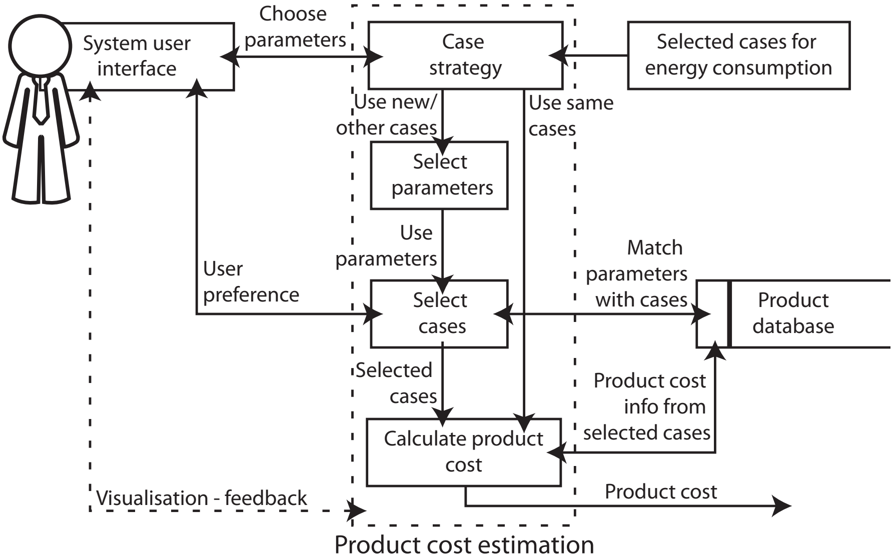


Figure 3.11. Estimation of the product cost module

that was explained in the case of product energy cost estimation. The information about the product costs of the selected cases is used to forecast the expectable product cost of the extended appliance.

The energy waste estimation component (Figure 3.12) has two sub-components. The firstly used sub-component generates a list of possible waste sources in the context of the product and offers it for the end-user. The ‘shortlist’ of possible waste sources contains general principles which are significant in the current design case. These principles limit the total amount of loss at the energy waste sources and can be considered by the designer to reduce energy consumption. This list is presented to the system user, who is supposed to identify the concrete energy waste sources in the conceptualized product.

3.5.7. Comparison of the result: conceptual distance

The main difference between the original and the enhanced framework is that it has been relocated to a new technology platform. Expert suggested to implement it as a web/hosted application, rather as an application package installed on workstations. This it can more easily an intelligent web-search application, and hence the contents of the tool can be maintained as up to date. This is needed by the fast evolution of the sensor and networking technologies fast. Another improvement is related to the case base. In order to keep this knowledge base up to date, the design support tool may have a learning function, and after completing the design task, the tool can automatically put the successful design alternatives into the case base, as a new case or as multiple alternative cases. We also

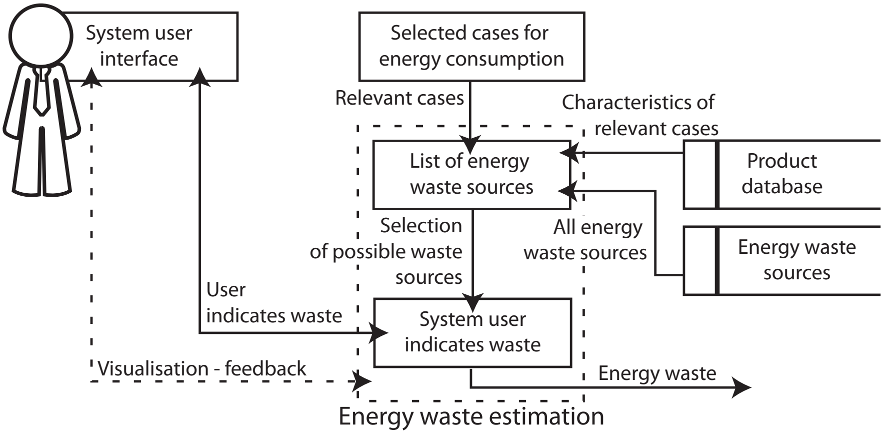


Figure 3.12. Estimation of the energy waste module

discovered that a case in a case-based reasoning system always contains two different things: the context (which are the product characteristics) and a solution of a problem (which are the energy saving possibilities). This denotes that we can use the same case base at two points in time during the design process. Though it was just explicitly discussed, the identification of waste sources can also be an enhancement of the forecasting tool, but it needs additional research. Based on the above recommendation of the experts, the task of the knowledge engineer is reduced. In an advanced implementation of the tool, the search for new control technologies can happen automatically based on an Internet browsing and a sophisticated filtering in the background. This means that only the waste source knowledge base should be kept up to date by the knowledge engineer. Finally, one of the proposals of the experts was to introduce some chronological order according to the natural flow of calculation of the product costs and the energy costs. After a critical analysis of the proposal we recognized that it may impose constraints on the designer and the preferred workflow. Therefore, we formed a different view on it, and decided to not apply any chronological or logical sequencing over the various modules and blocks

As the last action, we made a comparative analysis of the expectable performance indices of the initial and the enhanced framework, to identify the improvements from a computational point of view. The main issues are robustness and the complexity. The enhanced framework seems to be better because its structural complexity is reduced, and it can be assumed that it will have similar effect on the computational complexity. Although the number of components is higher, the number of flows is less. This indicates that the rearrangement of the functions resulted in a more effective architecture. It also means that the probability of failure of the forecasting tool is lower; hence the chance to accomplish the task is higher. Through these, the robustness of the tool can be higher.

3.6. Confirmative experiments and studies

3.6.1. Explanation on the general conduct of the confirmative research

The objective of the confirmative research was to test the effectiveness and efficiency of the CCR methodology. To achieve this, we applied the CCR methodology to our reference case. The method of empirical testing using a concrete application case was used as it is known to be the most effective way of testing methodologies; nonetheless it is a reasoning-with-consequences strategy. Our CCR-methodology has been applied and tested using the reference case that is introduced in Chapter 2. We observed how the process and methods of the CCR were applied, and considered if the methodology was supporting to obtain a desired outcome.

3.6.2. Organization of the experiment

The research was organized according to the procedural steps, explained in 3.4.2. The design concerns were identified, and different options were generated in advance. Based on these, design decisions were made and a framework was developed by designers separate from the expert session. During the expert session, design decisions were discussed with the intention to achieve a functional framework. The specific goal was to uncover noteworthy enhancements and suggestions that were different compared with the framework of the development team. By defining the main components and their characteristics, sufficient amount of information was shown to the experts. During structured decisions, the experts had to make a progressive assessment on the design decisions related to the software tool. Experts, rather than potential end-users, have been invited for the reason that we focus on pre-concept testing, and not usability testing, so providing information about user's attitudes, beliefs, desires, and their reactions was preferred [53]. For the sake of fairness, we have to mention that there is an epistemological problem. In theory, triangulation is carried out with two different researches and concepts that are not related to each other. The proposals of the focus group participants were theoretically-laden, because they did not start building up a new functional framework from scratch, but they internalized and commented on our proposed framework. This obviously means a knowledge independence problem in the research. In Figure 3.13., the applied

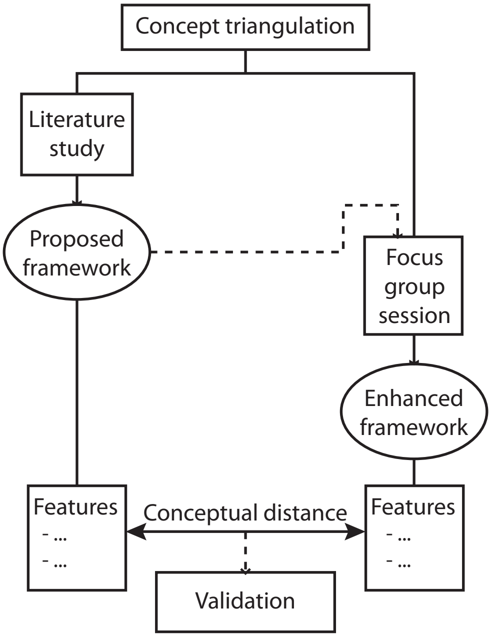


Figure 3.13. Applied triangulation

triangulation is visualized.

3.6.3. Raw data generated

The data generated during the expert sessions was all related to the test case. The focus group session was recorded with both a camera and a Dictaphone. Consequently, the complete development process was documented, so afterwards investigation could be done of how the session was organized and what the effectiveness and efficiency of the outcome was. We followed these principles in our research to assure reasonable software quality. First of all, we demonstrated the designed functional framework and the implementation concepts to experts active in some related engineering domains. The objective was to learn their reflections on the underpinning theory, and to discuss the fulfillment of the specified requirements and the methodology used in the software tool. The feedback of the expert will be taken into consideration before developing a detailed abstract prototype of the software tool, which will be demonstrated to stakeholders, such as the end-users (designers), software programmers, and knowledge engineers.

3.6.4. Coding, processing and interpreting data

The data that needed to be interpreted, analyzed, and processed is related to the outcome, process and methods applied in the reference case. We processed the data to derive an opinion on the justification, validation and consolidation of the developed CCR methodology. In general we were pleased with the amount and type of information generated by the stakeholders and the conceptual distance between the development team and experts was relatively small. Considering the process and the outcome, we had some remarks and considerations to improve the CCR methodology. More information on the confirmation of the CCR can be found in the next Section.

3.7. Confirmative research concerning the CCR methodology

3.7.1. Justification of the CCR methodology

The aim of the justification is to prove that the CCR methodology is logically error free. An indirect justification strategy was chosen, based on logical reflection, using the method of critical reasoning with consequences, because it was difficult to prove it directly. However, after executing the CCR methodology, we could discuss its criteria of goodness, defined in 3.4.4. The methodology was implemented in the development of the reference case. The application of the methodology was completed successfully in the reference case; however, we found some limitations, and constraints. To discuss the application experiment, we could only discuss the criteria of reliability and consistency. The facilitation of the methodology could not be discussed because it verifies how the CCR supports building another theory upon it. To discuss it we need the other theory. Therefore we cannot discuss it here, but we had to postpone it to the follow up chapter.

The CCR methodology could be considered as logically true, however there were some small limitations. The CCR is reliability and feasibility as we could execute it. The mapping of the design concerns, design options and design decisions was very useful to get an overview on the complexity that emerges in the software between the different parts and between the different decisions. However, we were considering what would happen if the complexity enlarges even more, because then support is necessary in visualizing the relationships and the effect of a decision. The CCR was also found internally consistent. Both the conduct of the expert session and the whole CCR process was good. Nevertheless, we also found some discussion items. Regarding the expert session with stakeholders, we concluded that the abstract level of details of information that is given regarding the software system both had positive and negative aspects. Positive was that due to the vague scope, stakeholders were not limited by made decisions. On the other hand, the vague scope also made it more difficult to empathize in the situation, and people got easily distracted. Therefore, it was important to clearly set the boundaries and to remind the stakeholders to the aim of the application. Unfortunately because of the abstraction level, there was a problem of interiorization and this made it harder to obtain a collective assessment. Regarding the whole process of CCR, we found that the total time and effort spent on CCR was promising, assuming that we have achieved a first level of stakeholder satisfaction, and an increased acceptability of both stakeholders and developers in the case of the given software application.

3.7.2. Internal validation of the CCR methodology

In this Sub-Section, the internal validity of the experiment was discussed. Validation may focus on multiple aspects, however we decided that construct validation, content validation and sampling validation were the most appropriate ones here. The method used for validating the methodology was reasoning upon the aspects that delivered the solution.

Construct validation

The first aspect is the construct validation: As it is important to validate if what had to be measured was really measured. Therefore, we investigated the different constructs or elements that were used during the operationalization of the CCR. The CCR is built upon different steps, using different constructs, having their input and output, and doing some data transformation, in order to retrieve a validated framework at the end. The identified constructs are: (i) the applied requirements, (ii) the critical design concerns, (iii) the design options, (iv) the design decisions, and (v) the conceptual framework. The methodology of CCR was developed to increase stakeholder involvement in the framework ideation phase, more specific in the decision making phase. In the process, expert stakeholders were involved in a group discussion to discuss the design options and make solutions for each concern, next to the development team's design solutions. We can conclude that the application-experiment was a valid approach as we could observe how the desired effect was achieved and how the different constructs of the methodology were needed to converge to a validated framework.

Content validation

In the content validation, we measured the extent to which the application case experiment represents all facets of the CCR methodology. The objective of the case experiment was to apply the methodology to the reference case and test1 the effectiveness and efficiency. The application was observed and reported in order to retrieve problems and difficulties that could be improved. We concluded that the CCR indeed supports the development of a rough conceptual framework. The selected reference case was valid to measure the applicability of the methodology, as it belongs to the operation domain.

Sampling validation

Lastly, we had to discuss the sampling validation. First we had to remind the fact that the exact types of stakeholders for the software are not known yet. As well their roles are also unclear. So during and after the development process, they may change, shift, extend, etc. however, this does not mean that the current information retrieved is not valuable. The stakeholder identification was based on the preliminary investigation. Further development of the software product will result in a better vision on the stakeholders, their roles, and their interactions with the system. Consequently, different stakeholders can be involved to deepen the problem domain versus those who are involved in making decisions regarding the solution. Regarding the whole development process, the effect of involving different stakeholders should be investigated. In this research, the expert stakeholders were selected using stratified sampling on the current number of stakeholders and their amount of interaction with the software. The experts invited had to be a purposive, not necessarily representative, sampling of specific types of stakeholders. The expertise was the basis of inviting the experts. We invited four types of experts to take part in the discussion session: (1) energy experts and experts on sustainable design, (2) information system experts, (3) application design (electronic design) experts, and (4) system methodology experts. This variation of expertise was necessary to get a comprehensive opinion on the tool. The strategy also dealt with the biggest issue of focus groups, namely the difficulty of getting people together at the same time and at the same place, as they were there for another (main) purpose, i.e. the session took place at an international symposium. The two hour long focus group session was held as a special workshop session at the Tools and Methods of Competitive Engineering Symposium in Italy, in April 2010. In total, fourteen experts were recruited from all round the world to participate in the session.

3.7.3. Consolidation of the CCR methodology

Consolidation has two aspects to discuss, the de-contextualization and the re-contextualization. The de-contextualization or generalization is not considered to be relevant for the CCR methodology. We could argue that it can also be used in other contexts such as hybrid systems and cyber-physical systems, where the to-be-handled-complexity is high as well. Nevertheless, further research efforts are needed into these directions to investigate the possibilities and to optimize this usage of the CCR methodology. In addition, as we do not want to use the CCR out of the context of the whole DSDM, we do not consider it to be important. The re-contextualization or specialization is more important here, regarding the reference case, but also regarding the information that is transferred to the next cycle in general. As CCR is the first methodology in the SD process, we have to consider how the information that comes out of this phase, will be used in the next phase, i.e. what the context and the objective of next phase is and how the current knowledge is useful for this. The aim was not to validate and verify the reference case, but it was necessary to verify and validate the application of the CCR methodology. Using the reference case, we could show that the delivered information can be used in the next cycle. The enhanced functional framework could be used as the foundation of the further research and prototyping. Moreover, its elements were taken to next phase. The conceptual framework can be divided into three large groups of elements that had to be detailed in the next research actions.

3.8. Concluding remarks

In the research context of the framework ideation phase, we considered a characteristic transition from problem description, manifested as requirements, into an abstract solution that is established as a functional and structural framework. Consequently, the most important aspects to achieve in this phase are (i) blending the knowledge of multiple domains into a consistent body of knowledge, and (ii) developing system-level understanding and conceptual framework of an abstract solution. Stakeholder involvement is an important added value in this first decision making process to achieve a relevant solution. They should be involved to model the expectations, needs and goals of the software and should discuss the critical design decisions. In this framework ideation phase, a methodology was essential that supports the just mentioned aims of stakeholder involvement. On the developed CCR methodology, we can conclude with the following propositions:

Proposition 1:

The CCR methodology enables better requirements and framework development by exploring expert-stakeholders' opinions

Proposition 2:

Complex systems deal with the problem of incomplete knowledge of the context and often ill-defined and conflicting ideas on the solution at the beginning of the development. Consequently, the requirements cannot be defined from the beginning but require an incremental and evolutionary strategy

Proposition 3:

SH involvement is crucial in the first decision making process to identify a relevant solution because the most important decisions are made here.

Proposition 4:

The CCR methodology enables better framework ideation by exploring expert stakeholders' opinion.

Proposition 5:

To handle the complexity, the design was split into manageable parts or concerns.

Proposition 6:

> During the guided expert discussion, a collective assessment was gathered on the design decisions, a shared understanding was created, and the acceptance was enlarged through interiorization.

Proposition 7:

> Based on the theory of triangulation, the design decision of the development team could be compared with those of the expert SH and the functional and structural framework could be generated and enhanced.

We concluded that the method of collective critical reflection is especially useful for software development. It might be useful in other contexts of complex systems development as well, where multiple kinds of stakeholder should be involved to clarify the problem and reason about a solution. Further research is needed in this direction as should also further develop the single-phase methodology CCR to increase the interiorization.

3.9. References

[1] Iivari, J., Isomäki, H., and Pekkola, S., (2010), “The user – the great unknown of systems development: Reasons, forms, challenges, experiences and intellectual contributions of user involvement”, Information Systems Journal, Vol. 20 (2), pp. 109-117.

[2] Kaindl, H., Constantine, L., Pastor, O., Sutcliffe, A., and Zowghi, D., (2008), “How to combine requirements engineering and interaction design?”, Proceedings of the 16th IEEE International Requirements Engineering Conference, RE’08, Barcelona, Catalunya, pp. 299-301.

[3] Garlan, D., and Shaw, M., (1993), “An introduction to software architecture”, Proceedings of the Advances in Software Engineering and Knowledge Engineering, Ambriola, V., Tortora, G. (Eds.), World Scientific., Singapore, pp. 1-39.

[4] Tang, A., and Vliet, H., (2009), “Software architecture design reasoning”, in: Software architecture knowledge management, Ali Babar, M., Dingsøyr, T., Lago, P., van Vliet, H. (Eds.), Springer Berlin Heidelberg, pp. 155-174.

[5] Zachmann, J.A., (1987), “A framework for information systems architecture”, IBM Systems Journal, Vol. 26 (3), pp. 276-292.

[6] Diehl, S., (2007), “Static program visualization”, in: Software visualization, Springer Berlin Heidelberg, pp. 35-77.

[7] Abrahamsson, P., Babar, M.A., and Kruchten, P., (2010), “Agility and architecture: Can they coexist?”, Software, IEEE, Vol. 27 (2), pp. 16-22.

[8] Vinekar, V., Slinkman, C.W., and Nerur, S., (2006), “Can agile and traditional systems development approaches coexist? An ambidextrous view”, Information Systems Management, Vol. 23 (3), pp. 31-42.

[9] Paetsch, F., Eberlein, A., and Maurer, F., (2003), “Requirements engineering and agile software development”, Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’03), p. 6.

[10] McMahon, P.E., (2004), “Bridging agile and traditional development methods: A project management perspective”, The Journal of Defense Software Engineering, p. 5.

[11] Giese, H., Rumpe, B., Schätz, B., and Sztipanovits, J., (2011), “Science and engineering of cyber-physical systems”, Dagstuhl Seminar 11441, Vol. 1, 2011, p. 22.

[12] Ferre, X., Juristo, N., and Moreno, A.M., (2005), “Which, when and how usability techniques and activities should be integrated”, in: Human-centered software engineering—integrating usability in the software development lifecycle, Springer, pp. 173-200.

[13] Penzenstadler, B., and Eckhardt, J., (2012), “A requirements engineering content model for cyber-physical systems”, Proceedings of the RESS 2012, Chicago, Illinois, USA.

[14] Zeng, Y., (2004), “Environment-based formulation of design problem”, Journal of Integrated Design and Process Science, Vol. 8 (4), pp. 45-63.

[15] Carroll, J.M., (2000), “Five reasons for scenario-based design”, Interacting with Computers, Vol. 13, pp. 43-60.

[16] Revilla, E., Prieto, I.M., and Prado, B.R., (2010), “Knowledge strategy: Its relationship to environmental dynamism and complexity in product development”, Knowledge and process Management, Vol. 17 (1), pp. 36-47.

[17] Sawyer, P., Pathak, A., Bencomo, N., and Issarny, V., (2012), “How the web of things challenges requirements engineering”, in: Current trends in web engineering, Grossniklaus, M., Wimmer, M. (Eds.), Vol. 7703, Springer Berlin Heidelberg, pp. 170-175.

[18] Hall, J.G., Mistrik, I., Nuseibeh, B., and Silva, A., (2005), “Editorial: Relating software requirements and architectures”, IEE Proc.-Softw, Vol. 152 (4), p. 2.

[19] Alfred, C., (2008), “Requirements vs architecture”, Charlie Alfred Weblog, http://charliealfred.wordpress.com/requirements-vs-architecture/, 2008.

[20] de Boer, R.C., and van Vliet, H., (2009), “Controversy corner: On the similarity between requirements and architecture”, J. Syst. Softw., Vol. 82 (3), pp. 544-550.

[21] Mahaux, M., Mavin, A., and Heymans, P., (2012), “Choose your creativity: Why and how creativity in requirements engineering means different things to different people”, Proceedings of the REFSQ 2012, Regnell, B., Damian, D. (Eds.), p. 16.

[22] Galster, M., Eberlein, A., and Moussavi, M., (2006), “Transition from requirements to architecture: A review and future perspective”, Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD’06), p. 8.

[23] Hall, J.G., Jackson, M., Laney, R.C., Nuseibeh, B., and Rapanotti, L., (2002), “Relating software requirements and architectures using problem frames”, Proceedings of the IEEE Joint International Conference on Requirements Engineering, IEEE, pp. 137-144.

[24] Avgeriou, P., Kruchten, P., Lago, P., Grisham, P., and Perry, D., (2007), “Architectural knowledge and rationale: Issues, trends, challenges”, SIGSOFT Softw. Eng. Notes, Vol. 32 (4), pp. 41-46.

[25] Jansen, A., van der Ven, J., Avgeriou, P., and Hammer, D.K., (2007), “Tool support for architectural decisions”, Proceedings of the Software Architecture, 2007. WICSA ‘07. The Working IEEE/IFIP Conference on, pp. 4-4.

[26] Jansen, A., and bosch, J., (2005), “Software architecture as a set of architectural design decisions”, Proceedings of the Conference on Software Architecture (WICSA’05), IEEE Computer society, p. 10.

[27] Waterman, M., Noble, J., and Allan, G., (2012), “How much architecture? Reducing the up-front effort”, Proceedings of the AGILE India (AGILE INDIA), 2012, pp. 56-59.

[28] Lee, L., and Kruchten, P., (2008), “A tool to visualize architectural design decisions”, in: Quality

of software architectures. Models and architectures, Becker, S., Plasil, F., Reussner, R. (Eds.), Vol. 5281, Springer Berlin Heidelberg, pp. 43-54.

[29] Kruchten, P., Capilla, R., and Dueas, J.C., (2009), “The decision view’s role in software architecture practice”, Software, IEEE, Vol. 26 (2), pp. 36-42.

[30] Montero, F., and Navarro, E., (2009), “Atrium: Software architecture driven by requirements”, Proceedings of the Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on, pp. 230-239.

[31] Broy, M., (2006), “The ‘grand challenge’ in informatics: Engineering software-intensive systems”, Computer, Vol. 39 (10), pp. 72-80.

[32] Mazalek, A., and Van Den Hove, E., (2009), “Framing tangible interaction frameworks”, Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, Vol. 23, pp. 225-235.

[33] Gerritsen, B.H.M., (2010), “Engineering frameworks: A bibliographic survey-based problem inventory”, Proceedings of the 1st IMS Summer School, Zurich, Switzerland.

[34] Berends, J.P.T.J., and Van Tooren, M.J.L., (2007), “Design of a multi-agent task environment framework to support multidisciplinary design and optimisation”, Proceedings of the Collection of Technical Papers - 45th AIAA Aerospace Sciences Meeting, Vol. 17, Reno, NV, pp. 11751-11772.

[35] Barreiro, J., Labarga, J.E., VizÃ¡n, A., and RÃos, J., (2003), “Functional model for the development of an inspection integration framework”, International Journal of Machine Tools and Manufacture, Vol. 43 (15), pp. 1621-1632.

[36] Fan, L.Q., Senthil Kumar, A., Jagdish, B.N., and Bok, S.H., (2008), “Development of a distributed collaborative design framework within peer-to-peer environment”, CAD Computer Aided Design, Vol. 40 (9), pp. 891-904.

[37] Romero-Hernández, O., Ponsich, A., Hernandez, S.R., Lascurain, M.d., and Aquino, J., (2010), “A multi-objective mathematical programming framework for a sustainability analysis of wastewater treatment processes”, Int. J. Environmental Policy and Decision Making, Vol. 1 (1), pp. 17-39.

[38] Easterbrook, S., (2004), “What is requirements engineering?”, in: Draft book chapter.

[39] Jingqiu, S., and Yingxu, W., (2003), “A new measure of software complexity based on cognitive weights”, Electrical and Computer Engineering, Canadian Journal of, Vol. 28 (2), pp. 69-74.

[40] Majid, R.A., Noor, N.L.M., Adnan, W.A.W., and Mansor, S., (2010), “A survey on user involvement in software development life cycle from practitioner’s perspectives”, Proceedings of the Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, pp. 240-243.

[41] Iivari, N., (2004), “Enculturation of user involvement in software development organizations - an interpretive case study in the product development context”, Proceedings of the Proceedings of the third Nordic conference on Human-computer interaction, ACM, Tampere, Finland, pp. 287-296.

[42] Webb, E.J., Campbell, D.T., Schwartz, R.D., and Sechrest, L., (1966), “Unobtrusive measures: Nonreactive research in the social sciences”, Rand Mcnally, Chicago, p. 240.

[43] Denzin, N.K., (1970), “The research act in sociology: A theoretical introduction to sociological methods”, Butterworth, (Publishers) Limited, Chicago: Aldine, p. 368.

[44] Alfred, C., (2008), “Complexity-driven”, Charlie Alfred’s weblog, http://charliealfred.wordpress.com/complexity-driven-1/, 2008.

[45] Roozenburg, N.F.M., and Eekels, J., (1995), "Productontwerpen, structuur en methoden (2nd edition)", Uitgeverij Lemma BV, Utrecht, The Netherlands.

[46] Yan, W., Chen, C.H., and Chang, W., (2009), "An investigation into sustainable product conceptualization using a design knowledge hierarchy and hopfield network", Computers and Industrial Engineering, Vol. 56 (4), pp. 1617-1626.

[47] Rabiee, F., (2004), "Focus-group interview and data analysis", Proceedings of the Proceedings of the Nutrition Society, Vol. 63, pp. 655-660.

[48] Krueger, R.A., (1998), "Analyzing & reporting focus group results", Sage Publications.

[49] Krueger, R.A., and Casey, M.A., (2000), "Focus groups: A practical guide for applied research, third edition ", Sage Publications, p. 215.

[50] Liamputtong, P., (2011), "Focus group methodology: Introduction and history", in: Focus group methodology: Principle and practice SAGE Publications Ltd p. 224.

[51] Winkelman, P., (2010), "A theoretical framework for an intelligent design catalogue", Engineering with computers, Vol. 27 (2), pp. 183-192.

[52] Puchta, C., and Potter, J., (2004), "Focus groups practice", Sage Publications.

[53] usability.gov, (2010), "Analyzing by using focus groups", 2010.

Chapter 4

Research cycle 3

Methodology of modular abstract prototyping

4.1. Introduction

4.1.1. Objectives of this research cycle

In this chapter we focus on research cycle 3. In this research cycle we investigated the concept development phase, which is in fact the most decision-intense phase of the whole software development process. A growing need of the industry was identified for new means to support testing of software concepts by stakeholders in the early phase of their development. In this early phase, the in-development software exists as functional and or procedural concept. At the moment, the available means are rather limited and more focused on the technical development of software tools, while early prototyping of software concepts is a challenging task due to the incompleteness and vagueness of the information that is available in this stage. The lack of proper modeling, simulation and demonstration means and the large opportunities, more than just supporting interface design were the biggest motivations towards the objective of this cycle. The objective of the research cycle is to develop and test an early software prototyping methodology to support testing with multiple stakeholders. By developing a rich and complete prototyping of the software concept, it is possible to aggregate stakeholders' feedback-through-demonstration as of the earliest phase of software development.

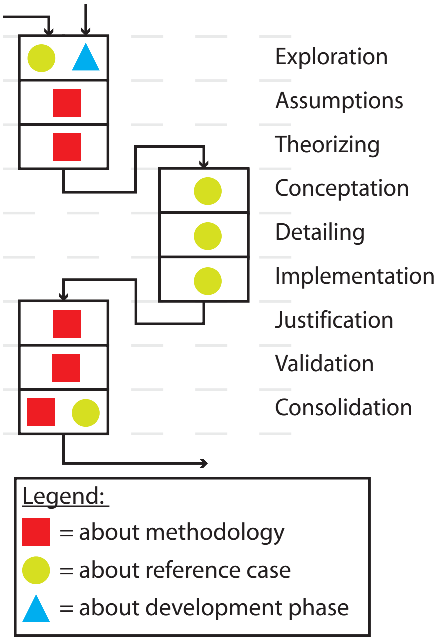


Figure 4.1. ***Approach RC3***

This prototype should include a real life manifestation of all characteristic operation and interaction/use processes, including the operation of the concept, the actions of the human actors, and the happenings in the surrounding environment.

4.1.2. Approach of research cycle 3

As explained in the introduction, the research executed in Research Cycle 3 is based on the framing methodology of design inclusive research. The approach of the research cycle is shown in Figure 4.1. In the exploration phases, we focused on the investigation of the concept development phase (Section 4.1.3), the opportunities for early prototyping (Section 4.2.1) and the need for a new methodology (Section 4.2.2). Based on these literature studies, assumptions were made (Section 4.2.3) and a theory (Section 4.3) was being developed for a specific early abstract prototyping methodology. During the design activities the methodology was applied in the development of the reference case (Section 4.4). We demonstrated the operational software concept to a multiplicity of stakeholders, using the novel methodology for modular abstract prototyping (Section 4.5). In the confirmative phase, justification, validation and consolidation were achieved based on the test data of the focus group experiment, towards the developed methodology (Section 4.6). Section 4.7 completes the research cycle with a discussion and conclusions.

4.1.3. Exploration of technical concept development

During the concept integration phase, solutions are generated for the different parts of the software applications. By starting with the most critical parts, inefficient and unsuccessful development tracks can be stopped earlier and less iteration is required. The output of this phase is a description of the operating principle, the subsystems and main components, and the materialization of the concept using drawings, bills of material and models[1]. In product development, the concept design phase includes two main aspects: (i) the design of the main components of the product, and (ii) the concept testing.

Designing technical concepts

Based on the framework developed in the previous phase, now the concept has to be designed. Within the limits defined in the framework, we find solutions for the most important design items. The primary function possibilities should be logically dissolved. Using prioritization, all open design problems are solved using creativity techniques. Then these alternatives are brought together in light of their consistency on the one hand and their potential for creating added value to the other side [2]. Second and third generation products are complex systems comprised of many interacting subsystems and components. The concept development phase considers the architecture of the entire system. During this phase, the system is broken down into subsystems and these further into many components. Teams are assigned to develop each component. Additional teams are assigned the special challenge of integrating components into the subsystems and these into the overall system [3]. The final assembly scheme for the production system is usually defined during this phase as well. The output of this phase usually includes a (geometric) layout of the product, a functional specification of each of the product's subsystems, and a process flow diagram for the final assembly process [3].

Concept testing

One or more concepts are tested to verify that the customer needs have been met, assess the market potential of the product, and identify any shortcomings, which must be remedied during further development. If the customer response is poor, the development project may be terminated or some earlier activities may be repeated as necessary [3]. Every stage of the concept development process involves various forms of models and prototypes. These may include, among others: early proof-of concept models, which help the development team to demonstrate feasibility; form-only models, which can be shown to customers to evaluate ergonomics and style; spreadsheet models of technical trade-offs; and experimental test models, which can be used to set design parameters for robust performance [3].

4.2. Knowledge aggregation and assumptions for abstract prototyping

4.2.1. Addressing the challenges and needs in concept development and testing

Importance to detect faults

Multiple papers have mentioned that many of the faults detected in existing software can be traced back to the problems of requirements specification, pre-implementation testing and user conformant evaluations [4-6]. It is well-known that the most influential decisions about the functioning, quality, features, properties and costs of are made in the early phases of product innovation/design projects. At the same time, the opportunities for developing and investigating alternative variants are the highest, as well as for introducing fast concept modifications without significant costs. Supporting the early phases of design projects is important. The problem is that the activities in different phases of software development, in particular in the design phase, do not scale to precisely match the underlying needs of the users [7]. Obviously, it is more costly to conceptualize something incorrectly and then to sort out the problems. Consequently it is cheaper to design and build the software right at the first time and to reveal all unforeseeable problems in an early phase.

Lack of prototyping means

In industry, there is an increased demand for new and effective means to enable rapid ideation, modeling and demonstration of software concepts to support software verification-through-demonstration in the early stage of product development. Early software concepts prototyping is a challenging task due to: (i) the incompleteness and vagueness of the information available at this stage, (ii) the emerging nature of the human ideas and the technical concepts, (iii) the difficulty to discuss just functional or procedural concepts [6, 8]. As for now, the available means are rather limited and more focused on the technical development of software tools, than on the demonstration of the operations and the impacts on the use environment [4]. There is also an inherent complexity involved owing to the fact that not only the operation of the system should be included, but also the human decision making, control and use actions, and the human-system interactions.

Comparison with other domains

While remarkable advancements have been achieved in the field of virtual prototyping and rapid physical prototyping, the progress is much less impressive in methodological and computer support of the inventive activities of product innovation [1]. Dedicated to pre-manufacturing modeling and testing, traditional virtual and physical prototyping technologies are used with the intent to create models of reasonably high fidelity. However, the effect of virtual and physical prototypes on the overall inspiration, creativity, and innovativeness is limited, and the technological and financial improvements that can be achieved by modifying these detailed models are often disproportional to the necessary efforts [3]. In addition, these technologies require detailed information about the form and functions of the product, do no offer means for capturing the operation and use contexts, and creation of virtual and physical replicas requires a lot of efforts, time and investment. Consequently, they cannot be considered in the fuzzy front end of product innovation and design [4].

User involvement

It has also been recognized that the involvement of representative software users in the development process is valuable, because it significantly improves the acceptance of the final product. Ultimately, usability comes from fitting the architecture and the content of the user interface to what the users are trying to accomplish [9]. In most cases, participatory design is mainly carried out with the involvement of limited number of potential users and low-fidelity prototypes, which are easy for the users to become familiarized with and which they can learn and employ by themselves. The most important aspect is to fit the communication modalities to the interests of the wide varieties of stakeholders [7].

Prototyping needs

Usually a systematic approach is desirable, because software errors are typically deeply and intimately embedded in the architecture of the software. In addition to the functional errors, the user and usability aspects should also be taken into consideration. The user aspects can be taken into consideration in software development by aggregating knowledge about the future users or by directly involving them in a participatory software development. This form of software co-development has been named participatory design [10-12]. Strong stakeholder modality and interest heterogeneity demands flexible yet effective means to demonstrate software concepts to dislocated stakeholders and focus-groups: using an application independent procedural framework covering both prototype development and demonstration, with modularly focused in-context content, concurrently to dislocated stakeholders, and through synchronized sessions for groups if needed, preferably in a virtual environment. This is an important aspect because mobilizing stakeholders for an on-site demonstration is often a complex and costly task. In customer-centered innovation processes, abstract prototyping can be the only viable way to rationalize information structuring for concept presentations.

Since end users have no knowledge or experience in reading source-codes of software tools, designers need to use efficient communication means. In addition to allowing hands-on experimentation, it is crucial that the demonstrative software includes more than just the interface. By providing the end users with clues about the structure and content of the developed software, we can support their understanding of the possibilities and limitations of the system, well beyond what is suggested by the interface [7]. For the reason that typically a lot of information is still missing, one of the major challenges of AP is that only low-fidelity ones can be developed at the beginning of the development process. The development of high fidelity prototypes would assume a much wider pool of information, which is only available in the later detailing phases of software development. However, in this case, the involvement of other software experts and end users is essential to support software conceptualization and exploration of errors [13, 14].

4.2.2. Knowledge aggregation on early prototyping

In software prototyping, two orthogonal dimensions of thinking exist: (i) one dimension of information content [14], capturing prototype functionality, and (ii) one of prototype fidelity [15]. They are visualized in Figure 4.2, which will be used as a reasoning model for our assessment. The first dimension is about the information content included in the early prototypes. The second dimension is the representation fidelity of the early prototypes. In terms of the information content that describes the software concept, we can identify explicit and implicit prototypes. Explicit prototypes typically implement the software on a testable level, for instance by coding and programming, to test the observable operations and behavior. As the opposite, implicit prototypes do not allow testing the functionality directly. They work with the expectations for and conditions of operation. They typically capture the requested content through a list or a structure of requirements or wishes.

In terms of the representation fidelity, both low fidelity and high fidelity prototypes are considered. Low fidelity prototypes apply strong simplifications in terms of modeling existing or imagined reality, while high fidelity prototypes seek to achieve the most thorough and comprehensive representation of reality [15]. It has to be noted that the dimensions formed by the information content and the representation fidelity of prototypes are not independent of each other. Low fidelity prototypes either apply a higher level of abstraction or use implicit contents to describe software concepts. On the other hand, high fidelity prototypes are implemented as a set of executable algorithms, which produces all important operations of the software, and can be tested for various criteria, such as reliability of instruction execution, data sensitivity, and computational performance [16].

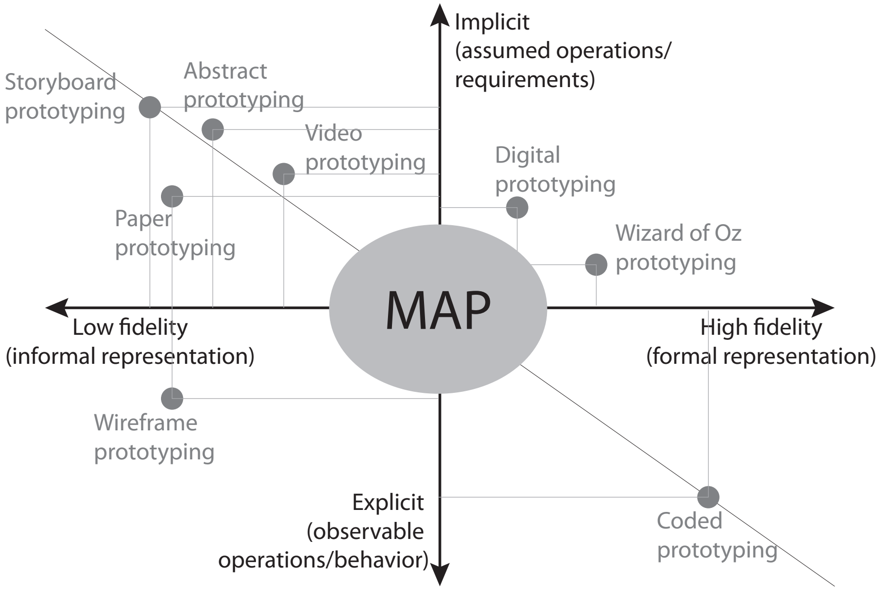


Figure 4.2. ***Prototyping of software products***

In Table 4.1, we address two essential aspects of early software prototyping, namely, (i) prototyping of the functionality of software tools, and (ii) prototyping of the user interfaces of software tools. We focus on the assessment of previously proposed methods, tools, and approaches. Interested readers should consult other detailed surveys, such as [14] and [15], which discuss the issues of early prototyping from various perspectives in more detail.

4.2.3. What is abstract prototyping?

Abstract prototyping (AP), also known under various names in the literature, such as pre-implementation prototyping [28], low-fidelity prototyping, [29], throw-away prototyping [30], rapid prototyping [5], early prototyping, low-cost prototyping [4], surrogate modeling, media prototyping [31], pre-implementation testing [32, 33], soft prototyping [34, 35], or paper prototyping is a testing approach in software engineering that supports demonstration and evolution of software concepts at an early stage. It allows designers to optimize the operation of the software and allows end users to understand how to work with the system, and can be manifested as an informal (such as mental or paper models) or as a formal manner (by using language models, animated models or interactive models). AP is used for various purposes in various contexts [36]. The common objectives of use are (i) aggregation of information, which cannot be obtained otherwise (i.e. without developing prototypes), (ii) to attain a comprehensive image on the operation and interaction possibilities, and (iii) to formalize the information inquiry and the entire of the software development process.

Table 4.1. Characteristics of low and high fidelity prototyping and application differences

	Characteristics	**User interface prototyping**	**Functionality prototyping**
Low-fidelity prototyping	strong simplifications; modeling existing or imagined reality; higher level abstraction; implicit contents; most often in the early stages	Usually throwaway prototypes by nature, and are produced by some quick-and-dirty prototyping techniques e.g. canonical abstract prototyping [17]	Shift from analogue paper prototypes (e.g. sketches, sticky notes, mock-ups, story boards) [18, 19] to digital software prototypes (e.g. PowerPoint, on-screen animations, live video streaming, motion simulations) [20]
High-fidelity prototyping	rather thorough and comprehensive representation of reality; set of executable algorithms; operations of the software; can be tested for reliability, data sensitivity, and computational performance, etc.; often used in the later stages of development	Different software tools have been developed to intuitively build interfaces for software prototypes that allow higher interaction. Examples are Denim[21], Silk [22], SketchWizard [23] and SUEDE [24]. Testing in real-life use environments and contexts is a new trend, inspired by mobile prototyping [25] and real-time prototyping in ubiquitous applications [20].	High fidelity prototypes are more robust testable implementations (evolutionary prototypes), which are produced using production-quality coding, and are designed for easy growth and frequent improvements [26]. High-fidelity functional prototyping is often referred to as agile software development, or human-centered extreme programming [27]

With regards to these objectives, the major roles what abstract prototyping can play in software development have been identified by researchers as follow. Namely, AP: (i) facilitates the communication to the end users in the early phase and the adaptation of the software-in-development to the user needs [12], (ii) makes the ideas tangible for the developers themselves [14], (iii) assists in clarifying the interface of the system [37], (iv) helps to identify the functional boundaries of the system [38], (v) facilitates making a forecasting on the required resources [19], supports making estimations on the desired system development capacities, money, time, infrastructure, etc., (vi) supports the exploration of errors and reduces the potential pitfalls [39], and (vii) provides means of process monitoring, and of systematizing the process [6]. Ultimately, it provides means for combining the relevant knowledge, procedures and methods into a comprehensive methodology, with the aim for abstract prototyping from the perspective of an application independent AP methodology. Considering the needs of the designers for easy to use support means and for a general applicability, our attention was orientated to a pragmatic methodology.

This methodology deals only with the minimally necessary information constructs in the process of abstract prototyping, and can be formulated with symbols as:

$$AP=M(N(P,S,C)) \tag{1}$$

Where,

AP is the abstract prototype,
P are the personas who participate in the process described by the abstract prototype,
S is the scenario of all operation and interaction sub-processes taking place in the process,
C is the context of the application and use of the software,
N is the narration of the story of the contents of the process, and
M is media-based staging and presentation of the contents of the process.

In fact, P, S, and C together constitute the information contents that are needed to describe the operation of and interaction with the developed software. They convey various chunks of information to the abstract prototype, such as: P => (type, sampling, characteristics, attributes), where type Î {end users, knowledge engineers, stakeholders}; S => (system functionalities, user behavior, system-user interactions); and C => (goal of system, tool environment, constraints). In the course of the AP process, first the specific information chunks are collected, structured and interrelated.

Towards the enactment of these contents, these information constructs are converted into and complemented by a narration N, i.e. with a story of the interactions and the autonomous operations happening, and by a media-based representation M, i.e. with an animation and visual presentation of the staging of the happenings. The narration and the visualization work together and strengthen each other. This mixed media representation of the software operation and interaction serves the purpose of demonstrations and assessment. This latter assumes criteria selection, knowledge aggregation from the stakeholders taking part in the early assessment of the software, and processing the feedback for both the software and the abstract prototype. This is important to be mentioned, because the assessment of the software is made through the abstract prototype developed. The narration, which is one essential component can be presented either textual or verbal or mixed format [40]. The textual information can be presented as static (as a book), as running (as the subtitling in a movie), or animated (appearing and disappearing when needed). The verbal communication can be classified according to having it from a single source (or from one narrator), or from multiple sources (or from a group of actors). Besides this narration, the visual presentation, also called staging of the abstract prototype, plays an important role in the communication. Based on the richness of information, it can be 2D symbol or script-based, 3D model or picture-based, and 4D time-animation (dynamic) based.

The usual first step in testing abstract prototypes is defining the criteria, which should be made separate for the demonstrated software and for the demonstrating abstract prototype. As a measure of goodness of the abstract prototype exactness, completeness, fidelity, etc. can be used. The goodness of the developed software however should be evaluated in terms of the operational requirements and usability requirements of the users and the stakeholders [41-43]. In general it means that the quality of abstract prototyping interplays with the observed quality of the software presented. A poor abstract prototype may suggest that the quality of the presented software observed to be less than it essentially is. On the other hand, an attractive and perfect-looking prototype can overshadow some quality deficiencies of poorly developed software, because the software itself is not available for demonstration and the designers strive after presenting their concepts as perfect, this paradox situation cannot be avoided. Nevertheless it is important to keep in mind that the quality of the abstract prototype does not have anything to do with the quality of the real system.

The next step of testing the abstract prototype is information gathering based with differently sampled user groups in repeated sessions. For usability testing different methodologies can be used in different contexts; (i) direct experimentation with single or multiple testers at the same time; (ii) active information processing, also called creative AP, which counts on the creative contributions of the users in the process; (iii) passive information processing, also called demonstrative AP that happens without giving the chance for the participants to intervene or change; and (iv) executing the test in a surrounding which is familiar for the testers (in the real world or on the web) or in an unfamiliar lab environment. As discussed in [11, 42], popular methods for information gathering are focus group sessions, field observations , interviews, logging actual use, proactive field study, and questionnaires. The last step involves the evaluation of the test results and making conclusions on the necessary changes. The necessary changes may concern the content of the software and the abstract prototype. What we found in the literature was that the methods used for information processing were in concert with the methods chosen for information gathering.

4.2.4. Abstract prototyping and the need for adapting the methodology

As discussed above, software concepts may evolve in the phase of conceptualization, but may also change dynamically during an interactive demonstration session. In addition, demonstration to various stakeholders usually requires different contents to be demonstrated in the form of abstract prototype. Since the latter two issues together pose additional challenges for abstract prototyping, we have studied the literature to see what scientific and practical research questions have been addressed, and what approaches and solutions have been proposed to support emergent and dynamic early prototyping of software. However, we restricted our attention to the development of early prototypes with varying contents and to rapid adaptation of prototypes to varying demonstration contexts, and ignored the technical issues of how changes can be incorporated in an early prototype.

Development of software prototypes with varying contents belongs to the domain of extreme programming or evolutionary prototyping [27]. As we found, the results in this domain are very scarce and limited, and studying the phenomenon of emergent prototyping is still in its infancy, contrary to the potentials that research in this domain may have. In the current practice, typically the same prototypes is presented to all stakeholders, they are not changed in the course of the demonstration and testing sessions, and the requested changes are discussed and introduced off-line. This causes repeating communication of changes to all concerned stakeholders afterwards [30].

Nevertheless, the necessity for dynamic reconfiguration in prototyping has already been recognized. Gray, P.D. et al. argued that it is a crucial factor for rapid prototyping to indeed be able to rapidly reconfigure prototypes (optimally, with a time delay of at most a few seconds) [44]. A second factor is that, ideally, the dialogue support system should allow the user to break off execution, make changes to the specification, and resume execution from the original position in the dialogue, which they also call suspended time editing. As a rule, reconfiguration must be easy and fast to achieve maximum benefits. Dynamic reconfiguration also ensures that changes can be made whenever the need is perceived, without any loss of the problem context. The reconfiguration cannot be the responsibility only of the (interface) designer or the design team. Since they prefer to be in control, solutions are required that are able to make it possible for end users to adapt software programs. From the aspect of emergent prototyping, we have to differentiate the situations when any part, including the application code, of the software can be reconfigured and the situations when only the interface specifications are modifiable. It seems that both are challenging and this gives the reason why only a very few researchers has touched upon the problem of dynamic and evolving prototyping of software concepts.

We use a more comprehensive interpretation of the term in our research, arguing that AP affords more than interface design [36]. We define a low-fidelity prototype as a functionally unresponsive prototype of simulated configuration, visuals and interactions. A medium-fidelity prototype is a (partly) complete functionally responsive or operable prototype of approximate configuration, approximate visuals and accurately simulating interactions to the evaluator. Definitions are a further generalization of fidelity classes given in McDonald IV [45] and conforming to the tools and methods described in Bowles and Box [46]. We re-conceptualized AP towards an all-embracing concept presentation and demonstration of proxy functionality and implementation of software tools to stakeholders: modular abstract prototyping (MAP). MAP modularizes demonstration content and session planning and supports animated demos to stakeholders anywhere online, through different media and scenario plays, and with rich facilities to capture their feedback. There are three major issues related to conducting effective participatory research: (i) the amount of information to be provided for the participants without causing large perceptive and cognitive biases, (ii) the manner of providing information with respect to the mental models of the participants, and (iii) extraction and aggregation of research data and interpretation of them in context.

Abstract prototyping is about designing realistic scenarios without really producing a workable tangible prototype. Three approaches have been identified, namely: (i) generic abstract prototyping (GAP), (ii) modular abstract prototyping (MAP), and (iii) interactive abstract prototyping (IAP). By definition, GAP is an implementation of the underpinning information structure in a demonstration means of a monolithic architecture. It presents the foreseen real-life process, or processes, in their embedding environments, including the involved humans and their actions. GAP assumes that the demonstration is orientated towards one stakeholder, or a homogeneous group of stakeholders. As opposite, MAP assumes that the demonstration is for multiple stakeholders with different professional background, mind sets, interests, and demands, and that they are interested in the assessment and improvement of the demonstrated concepts and implementations from multiple different perspectives. Including all pieces of information in one single abstract prototype would be unpractical either from a cognitive, or from a practical point of view. Hence, MAP reflects a context-dependent dissecting of the information structure into functional modules that can be combined according to the stakeholders and their demands.

From an information technological point of view, a MAP is a modular architecture, which allows an independent development and flexible adaptation of the modules according to multiple stakeholders and demonstration sessions. The information sub-structures encapsulated in the modules necessitates both a semantic and a technical analysis. The modules are supposed to contain complementing, rather than overlapping information sub-structures. A demonstration session can be broken down into a series of separate sub-sessions, in which the dedicated MAP contents are presented to the different stakeholders. This way, (i) the development of the demonstration materials can be more efficient and flexible, (ii) the information overload of the stakeholders can be reduced in comparison with that of generic APs, and (iii) and the demonstration sessions can be more intensive and stakeholder oriented. It is also an advantage that, by taking into consideration the stakeholders' opinions, the number of necessary iterations can be reduced and iteration cycles are made shorter. The number of modules (i.e. the resolution of the MAP) is closely related to the number and interests of different stakeholders involved in the assessment process. A specific combination of the modules should ensure that the optimal amount and pieces of information are provided to each of the stakeholders, in a cognitively controlled way. In addition, other principles, for instance, template-based development or maximal reusability can also be considered in the methodology of modular abstract prototyping.

A MAP has been defined as a comprehensive (self-contained), content-wise dissected information structure, which is operationalized for demonstration as a combination of multimedia enabled, digitally recorded narrations and enactments [42]. Eventually, the module development involves an analysis of the necessary information from the perspective of efficient informing, structuring the contents, and designing the narration and enactment. An advantage of MAP is that different narration and enactment parts can be produced by different abstract prototype developers or knowledge engineers, or by using significantly different media. In addition, a MAP allows concurrent content development for and parallel implementation of multiple modules, and, if needed, enables an easier change of the modules. Certain kernel modules can be repeatedly used in demonstrations for different stakeholders. Another practical advantage of using MAP is that decomposing a complex problem to more manageable modules reduces the challenges and the risks, (but raises the necessity for a careful structural design). The modules may have volatile relationships with the complete MAP and with each other. In principle, they may be arranged according to various structural patterns, such as a linear chain, tree, a loop, or even a web. In practice, the structure of the modules is determined by the body and flow of information required for an efficient and comprehensive informing.

4.2.5. Assumptions on modular abstract prototyping

We re-conceptualized AP as a comprehensive methodology of early and low fidelity, but rich-in-information, prototyping [36]. As an underpinning theory, *generic abstract prototyping* (GAP) has been developed based on the assumptions in [47]. By definition, GAP targets comprehensive capturing and demonstration of foreseen real-life processes in their embedding environments, including all humans (stakeholders) and their actions. GAP produces abstract prototypes of a monolithic architecture; see Table 4.2 for a comparison with MAP. Our main assumptions regarding MAP are as follows:

Assumption 1:

Modularly-organized abstract prototypes are presented to stakeholders in dedicated focus group sessions. MAP serves as an instrument to explore and aggregate feedback about software concepts from stakeholders, permitting for measuring and assessment of their requirements fulfillment.

Assumption 2:

The information structure of the to-be demonstrated software concept can semantically be separated into complementing information sub-structures that are encapsulated in the difference modules of the MAP.

Assumption 3:

From an information technological point of view, a MAP prototype is a hierarchically organized information structure, entailing modular content dissection. This way, MAP allows for modularization of developments and flexible modification in response to feedback from demonstration sessions.

Assumption 4:

The total demonstration session may be broken-up into a series of sub-sessions, in which dedicated MAP contents are presented to different stakeholders. In practice, sessions are organized per stakeholder group. This way, stakeholder information overload is reduced compared to GAPs, and the development of demonstration material renders more flexible and higher efficiency.

Assumption 5:

With the MAP methodology the number of iterations required may be expected to drop, as well as their typical cycle times, as a result of better informed decision making and faster development convergence.

Assumption 6:

The optimal number of modules (i.e. the AP resolution) is related to the number of

Table 4.2. Comparison of GAP and MAP main aspects and characteristics

Main aspect / characteristic	GAP	MAP
demonstration	real life process, based on: functionality, human actions, and application environment	medium-fidelity, rich information software process, based on: functionality, human actions, and application environment
content	scenario (bundle) captured	focused, modules-embedded semantic information sub-structures
stakeholder communication	perceptive and cognitive channels of human intellect	APM-based cognitively-controlled communication
process stages	four-stage process: 1. proposed software concept technical info aggregation 2. AP demo content compilation 3. field demo testing 4. data/findings assessment and conclusions on improvements	four-stage process: 1. conceptualization: software concept technical info aggregation 2. design of all modules 3. execution with different groups of stakeholders 4. data evaluation and conclusions (more information: section 3)
result produced	monolithic architecture	modular architecture containing modules with a specific content and modality adjusted for the intended stakeholders
targeted audience and demonstration mode	session series of single content demonstration to mixed audience of stakeholders	multi-session modular content demonstration to focus stakeholder groups

stakeholders and their modalities and interests, but may also be based on different criteria. A well-chosen combination of modules provokes adequate stakeholders' informedness and robust change management in a cognitively controlled way.

4.3. Theory and realization of modular abstract prototyping

The impact of these assumptions on the theory and realization, i.e. implementation process and aspects and criteria for goodness, are further explored below.

4.3.1. Underpinning theory

In conceptualization of MAPs, the formal theory of GAP, published in [47], has been used as a platform of departure. The information structure of MAP has been derived taking into consideration the abovementioned assumptions. We have adapted all formal definitions from [47] that are appropriate for MAP. The adapted notional concepts and information constructs of GAP are listed below: the to-be demonstrated process (Π), the software scenario (Ξ), the involved human actors (H), the surrounding environment (Σ), the content demonstration media means (M), the concerned stakeholders (SH). The process Π is modeled as a finite set of process objectives (Ω), process states (S), process transitions (T), locations (L) and durations (D). The scenario Ξ is specified by a finite set of resources (function / operation carriers) (K), operations (O), affordances (A), and indicative actions (IA), also called signals or messages. The human actors H are described by the set of human individuals (P), their competences (K), their roles (R), the set of performing actions (PA), and the set of manipulative actions (MA). The surrounding environment Σ is specified by a finite, non-empty set of entities (E), attributes (Γ), relations (Δ) and conditions (X)

As a consequence of the specific assumptions discussed in Section 4.2.5, we may consider the following: In general, any kind of AP is constructed from four major pools of information: (i) the technical concept information that describes the technical (functional, structural, and implementation) concepts related to a new software, (ii) context information which help transform the technical concept information into the possible technical content of the MAP, (iii) the demonstration content of MAP, which is derived from the technical concepts by considering the presentation context, and which is, in fact, built into the modules of a MAP, and (iv) the presentation context, which informs about the preferable way of structuring and presenting the modules. As represented in Figure 4.3, these bodies of information can symbolically be defined as follows.

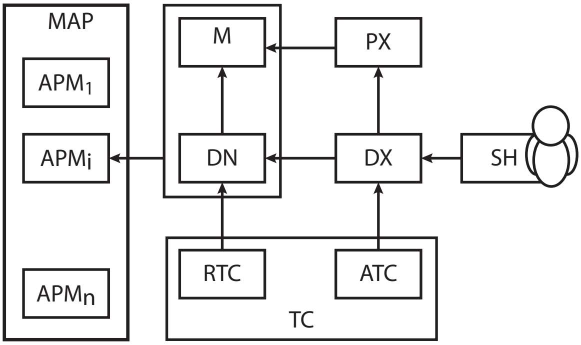


Figure 4.3. Visual representation of the information model of MAP

Let the technical concept information be denoted by TC. This includes a sub-set of information, representative technical concepts information (RTC), which is processed in the different abstract prototype modules (APM). The other sub-set of the technical concept information is not directly encapsulated in the APM. This however should also be aggregated in order to understand the technical concepts in a broader view and deeper. We refer to this as auxiliary technical concept information, and denote it by ATC. From the aspect of APM development, RTC should be processed explicit (built in) information, and ATC can be exploited as implicit information, guiding the proper definition of the technical contents of APMs. This additional information should in fact be considered by knowledge engineers at making decisions on the information contents and demonstration of MAPs.

Symbolically, we can write:

$$TC = RTC + ATC \tag{2}$$

Using the definitions provided in [47], TC can be formally defined as a four-topple:
$TC = \{\Pi, \Xi, H, \Sigma\}$, and (3)
$RTC = \{\{S, T, L, D\}, \{K, O, IA\}, \{P, PA, MA\}, \{E, \Gamma, \Delta, X\}\}$ (4)
$ATC = \{\{\Omega\}, \{A\}, \{K, R\}\}$ (5)

The technical content information that is eventually embedded in a MAP is a sub-set of the RTC. This is actually the demonstration content of the APMs. Let's denote it by DN. Using the above introduced symbols:
$DN = RTC'$ (6)
Where
$RTC' \subseteq RTC$ (7)

Considering the above specifications, we can write:

$\{S, T, L, D\}' \subseteq \{S, T, L, D\}$ (8)
$\{K, O, IA\}' \subseteq \{K, O, IA\}$ (9)
$\{P, PA, MA\}' \subseteq \{P, PA, MA\}$, and (10)
$\{E, \Gamma, \Delta, X\}' \subseteq \{E, \Gamma, \Delta, X\}$ (11)

This means:
$DN = \{\{S, T, L, D\}', \{K, O, IA\}', \{P, PA, MA\}', \{E, \Gamma, \Delta, X\}'\}$ (12)

With these, the total demonstration content information embedded in a MAP is:
$DN = \cup_{i=1}^{n} DN'_{i}$ (13)

Where
n represents the number of modules

This embedded demonstration content information is semantically appended with two other bodies of information, namely: A) the auxiliary demonstration context information, DX, which is composed of context information from two sources: (i) the auxiliary technical concept information (ATC), and (ii) the stakeholders (SH) related demonstration context information, SHX, (defining the technical content of APMs); and B) the presentation context information, PX, (which guides and influences the way of development and delivery of the media-based presentation). The stakeholder related demonstration information, SHX, can be defined as:

$SHX = \{\{\Lambda, P\}, \alpha\}$ (14)

Where $\{\Lambda, P\}$ are the stakeholders perspectives and demands, and α is interpreted as a module selector operator, which is playing role in expressing the context of interest of the stakeholders. Considering these, DX can be symbolically defined as:

$DX = \{ATC, \{\Lambda, P\}, \alpha\}$ or (15)

$$DX = \{\Omega, A, \{K,R\},\{\Lambda,P\}, \alpha\} \qquad (16)$$

As mentioned earlier, PX is a complement of DX. This set of information is used to select the most appropriate media (M) according to the stakeholders (SH) and the demonstration content (DN). The demonstration content of an abstract prototyping module (APM$_i$) consists of a subset of DN and is represented by means of some content demonstration media, M$_i$. With this:

$$APM_i = \{DN_i, M_i\} \qquad (17)$$

In Figure 4.4, we visualized the relationships between the above introduced information sub-structures and the different stakeholders. The bottom part of the figure shows the stakeholders. The middle part indicates the MAPs, which are presented to n-different stakeholders. The upper part indicates the configuration of m-number of APM, which are included in a particular MAP. As shown in Figure 4.4, a particular MAP may consist of different numbers of modules, and the total information content of a specific MAP in a known demonstration context, DX = {ATC,{Λ,P}, α}, can formally be defined as:

$$MAP = \otimes APM_i = \otimes\{DN_i, M_i\} \qquad (18)$$

In the next Section, we will elaborate on the procedural aspects of the MAP methodology. In addition to describing the prototyping procedure, we also touch upon some of the methods that can be applied to generate and process the information structures put forward by our theoretical considerations.

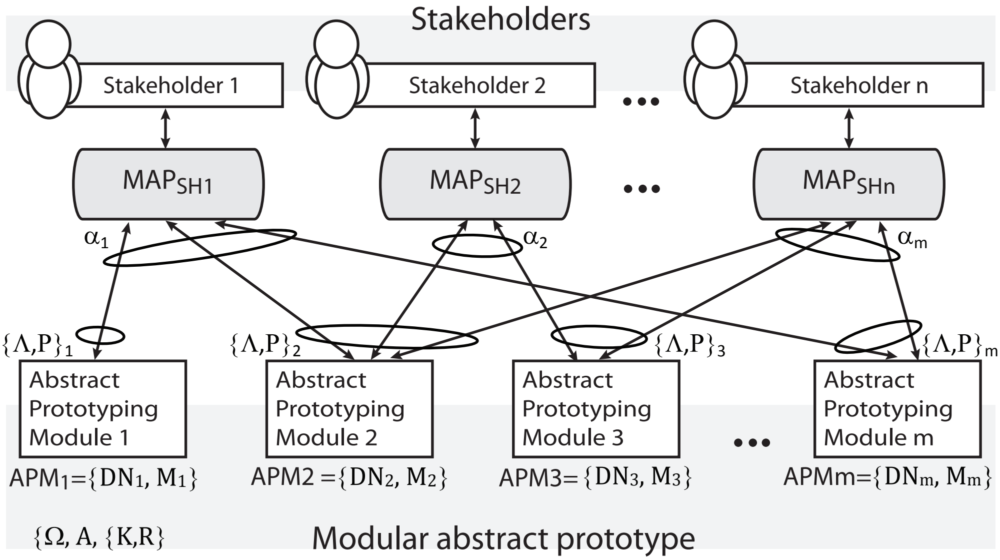


Figure 4.4. ***Assembling the APMs into specific MAPs for different stakeholders***

4.3.2. Procedural aspects

Assuming that the conceptualization of the software design has been sufficiently elaborated and enhanced to embark on an evaluation and discussion, and that these results are available for abstract prototyping, the whole execution process of MAP can be split up into four steps, aiming for informed decision making and strongly funneling change proposals. An overview of the MAP process is shown in Figure 4.5. Elaborated below, different actions are needed to operationalize the MAP methodology, along with fitting methods to generate and process information structures, as implied by our theoretical considerations.

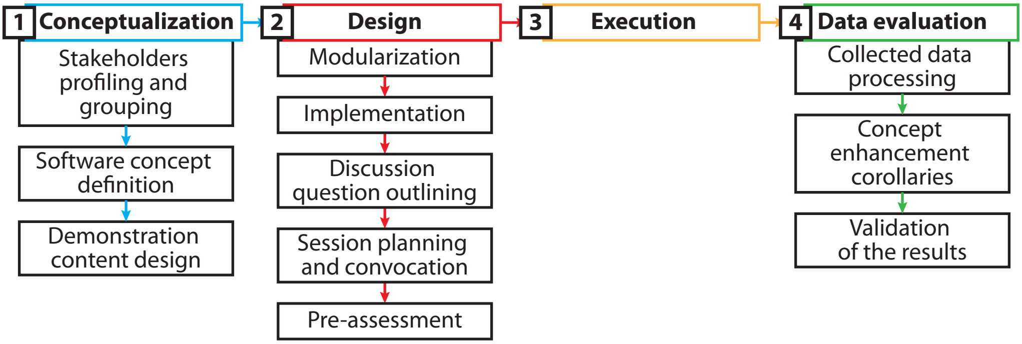

1 Conceptualization
Stakeholders profiling and grouping
Software concept definition
Demonstration content design
2 Design
Modularization
Implementation
Discussion question outlining
Session planning and convocation
Pre-assessment
3 Execution
4 Data evaluation
Collected data processing
Concept enhancement corollaries
Validation of the results

Figure 4.5. Process of MAP towards testing concept integration

Phase 1: Conceptualization

The first phase of the MAP process concentrates on software concepts' technical information aggregation. The phase involves activities that are necessary in any user-centered software development project (structuring and prioritizing of information, persona creation …). This preliminary phase is necessary to design the prototype, successful execute the tests and achieve valuable data evaluation.

Stakeholders profiling and grouping: identification and description of stakeholders and decision making on whom to invite for demonstration sessions. Although this is context dependent, in general all profile descriptions are qualitative descriptions that include the following information about the stakeholders: (i) their relationship towards the software concept and necessary context information (users, developers, …), (ii) the necessary technical contents for the modules of the prototypes (e.g. the technical functions, the user actions, the human computer interaction, …), and (iii) favorable modalities and media (flow charts, block diagrams, UML, text, animations, visuals, photos, …).

Software concept definition: technical concepts aggregation, based on the best of available technical information. Information should be structured and prioritized according to their significance (criticality). For optimal stakeholders feedback to software engineers, careful technical content design is critical, as well as a careful articulation of critical aspects.

Demonstration content design: demonstration content is the total of the chunks of information that should be explicitly built into the modules: (i) an explicit definition of the demonstrated process, (ii) functions of the developed software, (iii) actions of human actors involved and their interactions with the software, and (iv) the environment (including constituting entities, their characteristics and relationships). Demonstration content definition includes three interwoven activities: (i) conceptualization and specification of end-users as personas [48], (ii) modeling and specification of software concepts as a functional and structural system, and (iii) modeling and specification of the operational environment of software and end-user interaction. These touch points (human interaction, sensor inputs, output devices, etc.) are crucial to have a complete scenario of system operation, human actions, human-system interactions, and environment is thus obtained.

Phase 2: Design

This phase involves the compilation and testing of module technical contents:

Modularization: activities in this phase are (i) AP decomposition into demonstration modules, (ii) allocation of (parts of the) demonstration contents to specific modules. The decision on the number and the contents of the modules is guided by stakeholders profiling and grouping.

Implementation: demonstration contents of abstract prototype modules are embedded in narration and in enactment, two interrelated constituents of abstract prototypes. Narration is a simplified synthetic description of outline and highlights of a foreseen real-life process. Narrations are operationalized in the abstract prototype by appropriate media selection (e.g. animated text, human voice, synthetic sound, etc.). Enactment is the actual staging, performing, and media-enabled visualization of significant arrangements, happenings and contributors to the process. The narration part of an abstract prototype module is typically developed and edited in textual form. Narration texts permit prototype developers to identify logical units in each and every module, called episodes. Furthermore within each episode, they can identify and mark terms and phrases as keywords. Keywords are points where enactment elements are linked. An appropriate media composition and selection is needed for each enactment, fitting the related episode of narration.

Discussion question outlining: MAP developers should prepare questions prompting stakeholders' feedback for informed decision making. Enough information must be conveyed in the demo to empower stakeholders accordingly. Therefore discussion questions and prototype modules should be developed in parallel.

Session planning and convocation: a plan for demonstration sessions should be developed: (i) choose a test method and procedure (an efficient approach is to organize focus group sessions with on-site or videoconferencing-based stakeholder participation, but other approaches may work as well), (ii) group participants, (iii) set up testing, and (iv) invite stakeholders. Heterogeneity makes focus group sessions challenging, homogeneity of the participants' background improves shared understanding and awareness, enhancing discussion and decision making effectiveness.

Pre-assessment: a trial run prior to full-scale real-life demonstration sessions pre-assesses harmony of contents and presentation. A convincing carefully constructed technical information content can easily be destroyed by poor structuring or implementation of the MAP modules. Criteria focusing on functional and utility qualities towards effective feedback should prevail over media types, whilst exactness, transparency, clarity, accessibility, completeness, contrast of options, and fidelity prevail as measures of goodness. The goodness required for effective feedback may differ from focus group to focus group. Indicators for this may be embedded and managed in the profiles as well as the modules. Their match can also be assessed in this Pre-assessment.

Phase 3: Execution

The objective of this phase is to collect stakeholders' feedback on the demonstrated software concepts functionality, interfacing, and performance, so as to come to full-scale assessment, as-is. Interrogation of individuals and focus group sessions [49] are the two most frequently applied techniques to achieve this [11, 42]. Several stakeholder participation modes can be applied:

- Passive (observant) stakeholder participation, with opportunity to reflect and raise questions in a follow up discussion

- Active (interactive) participation and having the opportunity for immediate reflections, interrupting questions and follow up discussion, and

- Executing the assessment in an inventive form, providing opportunity for the stakeholders to propose changes to software concept and to see the effects of the proposed changes immediately during the session

The latter option will probably provide the best clarification, nevertheless, it would require a software tool supporting immediate on the spot module re-engineering. Further research is needed to develop such real-time adjustment tools.

Phase 4: Data evaluation

A well-designed and executed MAP process may empower stakeholders to formulate change proposals directly during session. Some reworking or clustering is commonly needed, however, and proposals should be evaluated and ranked in conjunction. During this phase, stakeholder feedback is being assessed and possible changes to or refinements of software concept are defined:

Processing of collected data: recorded data from (i) video-recordings (with the permission of group participants), or (ii) note-taking by a fellow researcher (not taking part in actual discussions), are processed. Recordings should be transcribed, integrated and checked, so as to cater for adequate quantitative and qualitative analysis material [50].

Concept enhancement consequences: results are converted into change proposals to improve and optimize the software concept. Data reduction techniques may be needed to rank the data, and semantic interpretation to extract their meaning. Changes are evaluated for impact, dependencies, effort, feasibility and priorities. The process may be supported using software change control and software configuration management tools [51].

Validation of the results: results obtained should be validated. This can be conducted using a control group, or by comparing different stakeholder responses. The theory of theoretical saturation can be used to confine the number of focus groups [52]: data saturation is obtained when no significant new information emerges in a focus group session.

4.3.3. Implementation aspects (narration and enactment)

Demonstration contents of abstract prototype modules are embedded in narration and in enactment, two interrelated constituents of abstract prototypes. Technically, narration is a simplified synthetic description of the outline and highlights of the foreseen process. It may demonstrate the process from multiple perspectives, such as its manifestation, its embedding in the real life environment, or its impacts on the embedding environment.

Enactment is the actual performing and media-enabled visualization of the episodes of the process, including all conceived arrangements and happenings of significance. The relationships between the narration contents, narration media resources, the enactment contents, and the enactment media resource is graphically shown in Figure 4.6. From the viewpoint of the applied presentation media, narration can be delivered either in textual form, verbal form, or a mixed form. The media for textual delivery can be structured text, such as blocks of texts, moving text lines, and animated text fields. In specific cases, even handwritten text can be included. The media for verbal communication is human voice or machine voice. Verbal narration can be produced either in a single narrator setup, or in a multi-narrator setup. The explanatory story of the narration is normally broken up into logical blocks, called episodes. As a result of this decomposition, the narration can be developed in a modular way, which might be extremely useful in case of long narrations. The length of the episodes in time is vaguely determined by the human memory capability and comprehension in a frame of discourse. Therefore, the textual description of an episode is not supposed to be longer than 400-500 words.

It is supposed that narration primarily works in the cognitive domain of human communication, while enactment works in the perceptive domain. They are complementing and enhancing each other towards an optimal impression and largest impact. What it means in the implementation practice is that the units of the enactment, called segments, are

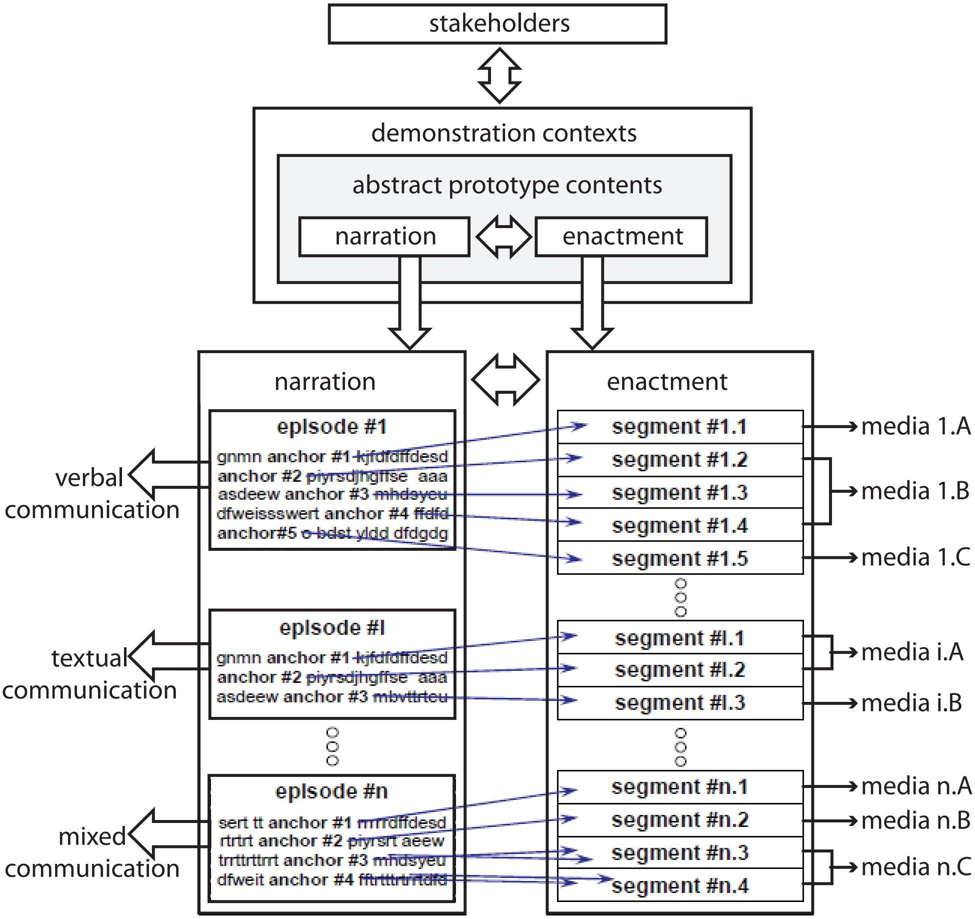


Figure 4.6. The relationships between contents of narration and enactment (based on [47])

connected to the narration text/speech at certain semantic anchors. A semantic anchor is a word, term, or phrase within a block of text/speech that needs and allows virtual articulation. Before designing the enactment, the abstract prototype developer should identify a limited number of semantic anchors in the narration. As a matter of fact, the number of the anchors is defined by the content and the length of the block of text/speech. We have considered 4 – 6 in an episode of average length. The action performed in a segment of enactment may be visualized by using various media forms. Normally the duration of the narration and the enactment are the same, and they start and finish at the same time. However, the duration of the enactment can also be shorter, or even longer.

The narration can be used to provide an early introduction of the objectives and main considerations, without any visualization. If the enactment is longer, then breaks can be included in the narration as needed, and vice versa. If the narration is longer, then the enactment will comprise disjoined segments, which turn up in the order and time defined by the semantic anchors. A segment can be visualized by using one media, such as common symbol diagram, animated symbol diagram, graphical sketch/drawing, slide/photo show, cartoon/digital animation, motion picture, stereoscopic visualization, holographic visualization, interactive visualization, or immersive visualization, or any combinations of them. These usually appended with human talks or sounds. Narration reflects an external perspective on the demonstrated process, while enactment presents an internal view point. As far as the media used to express and demonstrate the content information of the AP is concerned, its expressiveness and objectiveness are more important, than its attractiveness, though Aps should be convincing also from this aspect. These features of the demonstration media form a confounding variable in the demonstration. Therefore, the demonstration media must be carefully selected not to interfere with the technical contents to be demonstrated. Otherwise, the incorrectly chosen media may become exaggerating, overwhelming and misguiding. This indicates that a vague optimum is to be targeted.

The contents of the narration should create a sufficiently deep insight and a comprehensive awareness, while the enactment should modestly, but convincingly visualize all features of the foreseen process. Towards these ends, the enactment should also cover what cannot be included in the narration explicitly, and should complete what could just partially be included. As a methodological issue, it worth noting that abstract prototyping must obey the principle of parsimony, that is, it should strive for achieving a trade-off in terms of the investments and the return on the investments (feedback or approval). In practice it means that the AP should convey all necessary pieces of information about the process using the most appropriate media form, but not more than that is sufficient. As a simple solution, for instance, white-boarding can be used to show operations of the design tool and the related human mental and physical actions, and designed screen shots can be used to showing the human-tool interaction. The recording of a verbal narration and this sort of enactment can be animated in photo movie making tools or by combining digital models with live video streams.

There are also multiple possible forms of demonstration sessions, with different benefits and deficits. The demonstration of APs for stakeholders in presentation sessions may happen in: (i) reflexive, (ii) interactive, and (iii) constructive forms. In a reflexive demonstration, the AP is presented to the stakeholders without interruption, and the discussions and the processing of the presented knowledge happens afterwards. The two advantages of this form are: (i) that guarantees that the full AP is presented in the timeframe allocated for the demonstrations, and (ii) that it makes possible to separate the presentation and the assessment in space and time (e.g. as a dislocated idea review). A shortcoming is that it does not support the formation of the shared awareness through immediate reactions, though a high level of shared understanding can be achieved by a well-constructed abstract prototyping.

An interactive demonstration allows the stakeholders to make comments and ask questions at any moment in the course of the presentation of the AP. What it requires are a stoppable and resumeable AP design, and a strong moderation. The interaction supports the rapid formation of the shared awareness among the designers and the stakeholders. This may lead to a better assessment of the proposed concept on the side of the stakeholders, and collecting more information for enhancements on the side of the designers. An interactive demonstration does not support real time changes in the contents of the demonstrated AP. This is however an explicit goal of a constructive demonstration session. Modifications can be introduced by substituting certain modules of the constructive (modular) prototype by pre-prepared modules, or by providing computer based means for a real-time and fast modification or regeneration of certain information constructs according to the requests and advises of the stakeholders. The advantage of this approach is that the creative interaction may result in appropriate and innovative solutions. The pitfalls are that it: (i) requires higher level involvement of the stakeholders, (ii) significantly extends the time of the demonstration and assessment, (iii) the procedure may be hanged on by lack of information, and (iv) the constructive manipulation of the AP requires sophisticated editing tools.

4.3.4. Criteria for goodness

To justify the MAP methodology, which means checking its logical correctness, criteria of goodness were identified. In general, this logical correctness can be split up into: (i) reliability, (ii) consistency, and (iii) cohesion. Converting these aspects into criteria, we can conclude that: Reliability of feasibility can be measured if the methodology is executable (e.g. by checking the amount of information to communicate, time and effort to build the prototype, achieved fidelity and clarity, level of abstractness in thinking ,…). Consistency can be expressed by checking if there are no conflicts between the methodology components so if the methodology is internally contradiction free (by checking if the chosen demonstration means is according to the SH: adapted rapidly in terms of their contents, depending on the viewpoint, knowledge and demands of the stakeholders optimal amount and pieces of information are provided to each of the stakeholders, flexibility, modular development of abstract prototypes also increases productivity because different modules may be developed by different researchers and designers,...). Cohesion can be measured by its friendliness to other theories, if it is facilitating or enabling the implementation of other theories.

4.4. Application of the MAP methodology to the test case

The application case, as discussed in Chapter 2, has a dual objective. On the one hand we want to demonstrate how the MAP methodology can be applied in practice and demonstrate its usefulness and usability. On the other hand, by developing an application case that could be taken as a reference case, we may verify the efficiency and effectiveness of the methodology.

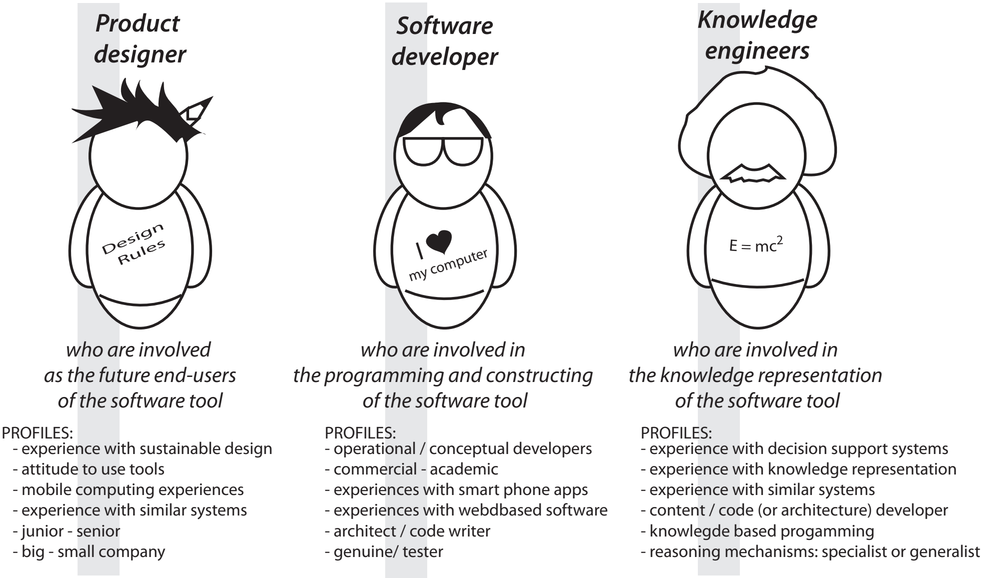
Empirical testing in concrete application cases is known to be the most effective way of testing methodologies, although it is a reasoning-with-consequences strategy. We applied this strategy to verify the suitability of our methodology in use context. No other theoretical or non-experimental strategies could be considered for this confirmative testing. We choose this approach also for pragmatic reasons, as no other medium-fidelity prototyping methodologies were available to compare with at the time. In this Section we will provide information about how the methodology has been used in the concrete application reference case, which involved conceptualization and development of a software tool to support designers in smart energy saving. Sub-sections from 4.4.1 until 4.4.5 discuss the taken procedural steps, and provide an overview of the applied methods.

4.4.1. Conceptualization

Identification of the stakeholders

Three different groups of stakeholders identified for this software tool are, based on the information gathered from the problem definition and framework ideation as discussed in [53]: (i) product designers, (future end-users of the application); (ii) software developers (programmers of the application); and (iii) knowledge engineers (for knowledge processing about cases from the past in the case-base).

In the pre-testing phase, not only the special characteristics of the particular groups of stakeholders were identified, but tentatively also a body of information that they need about the functionality and application of the software tool in order to be able to judge the merits and pitfalls of the proposed concept. For example, product designers must be practicing designers, must have sufficient experience with electronic application design, and should have experience with applications of design software tools. Further characteristics of their



who are involved as the future end-users of the software tool

PROFILES:
- experience with sustainable design
- attitude to use tools
- mobile computing experiences
- experience with similar systems
- junior - senior
- big - small company

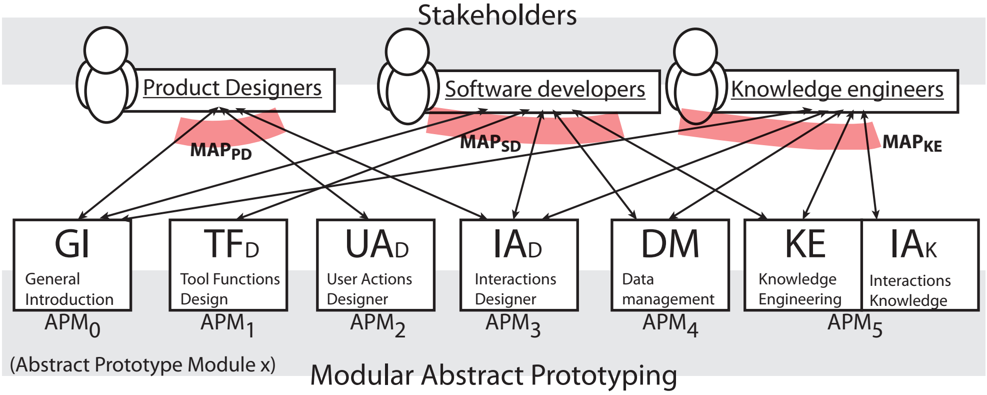
who are involved in the programming and constructing of the software tool

PROFILES:
- operational / conceptual developers
- commercial - academic
- experiences with smart phone apps
- experiences with webdbased software
- architect / code writer
- genuine/ tester

who are involved in the knowledge representation of the software tool

PROFILES:
- experience with decision support systems
- experience with knowledge representation
- experience with similar systems
- content / code (or architecture) developer
- knowlegde based progamming
- reasoning mechanisms: specialist or generalist

***Figure 4.7.** Identification of the stakeholders*



Formulas:

$MAP_{PD} = APM_0 + APM_2 + APM_3$

$MAP_{SP} = APM_0 + APM_1 + APM_3 + APM_4 + APM_5(IA_K)$

$MAP_{KE} = APM_0 + APM_3 + APM_4 + APM_5$

Figure 4.8. Overview of the modules developed for different stakeholders for our application case.

competence profile have also been taken into consideration at sorting the designers into groups according to their experience with ecodesign, their attitude to use interactive and knowledge-intensive tools, their initial knowledge on smart and ubiquitous technologies, and their experiences with designing smart controls. An overview of the identified stakeholders and their profiles are shown in Figure 4.7.

Defining the demonstration context and content

Having identified the characteristics of the stakeholders and their possible demands and requirements for the to-be-demonstrated software concept, we collected and structured the technical information concerning the concept of the software tool. Based on this, the technical information content to-be-embedded in the MAP and the technical information contexts were defined. Furthermore, information about the demonstration context was also collected. Based on these, the contents of the various modules have been determined, and the composition of modules into stake-holder oriented MAPs has been defined (Figure 4.8).

We decided to have six modules:

0. A general introduction module that introduces the concept of the application. This module should be universal for all stakeholders and should contain all basic information needed for the understanding of other modules. Important in this module is the use of a shared terminology, modality and goodness of demonstration. A selection of screenshots from this module is shown in Figure 4.9.

1. The second module demonstrates the functionalities of the software application. This

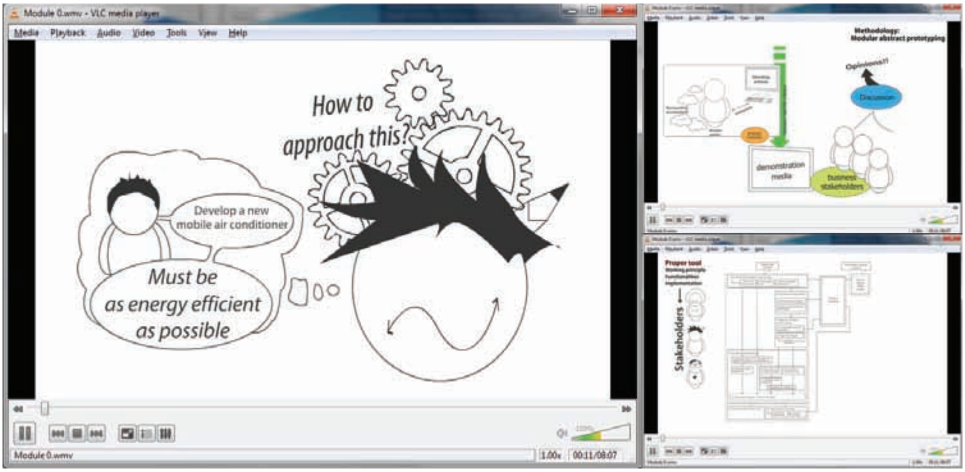

Module 0.wmv - VLC media player
Media Playback Audio Video Tools View Help
How to approach this?
Develop a new mobile air conditioner
Must be as energy efficient as possible
Module 0.wmv
1.00x 00:11/08:07
Methodology: Modular abstract prototyping
Opinions!?
Discussion
demonstration media
business stakeholders
Proper tool
Stakeholders

Figure 4.9. *Screenshots from the general introduction module*

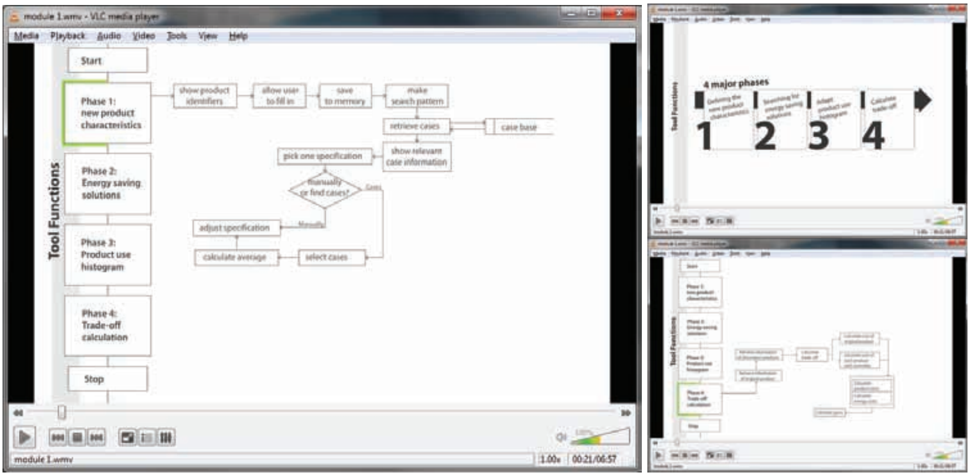


Figure 4.10. *Screenshots from the tool functions for designers module*

information is particularly important for the software developers. A few screenshots are shown in Figure 4.10.

2. The third module informs about anticipated user actions and cognitive thinking processes, while interacting with the application. While not of prime interest to software developers, it is critical to product designers, who need this kind of information to check whether it fits their logic and reasoning processes. In Figure 4.11, a brief overview is shown of the module using some specific screenshots.

3. The fourth module represents a use scenario of the human-computer interaction and is

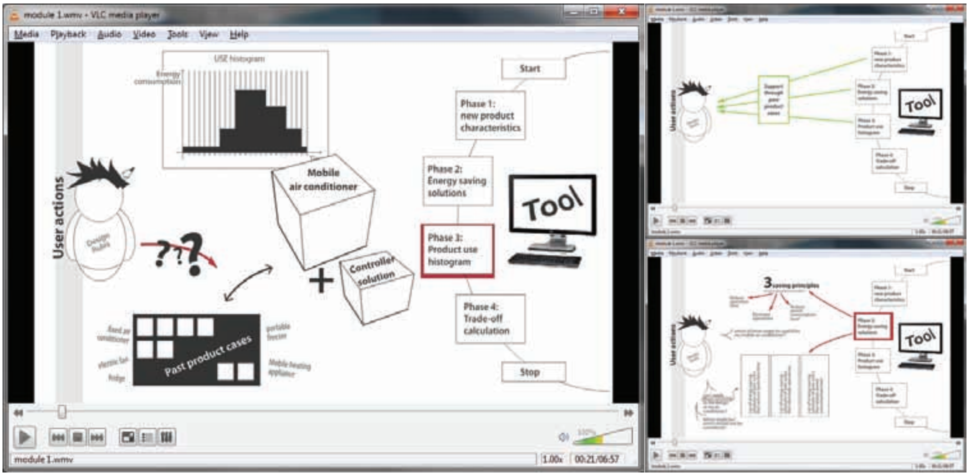


Figure 4.11. Screenshots from the designers' use actions module

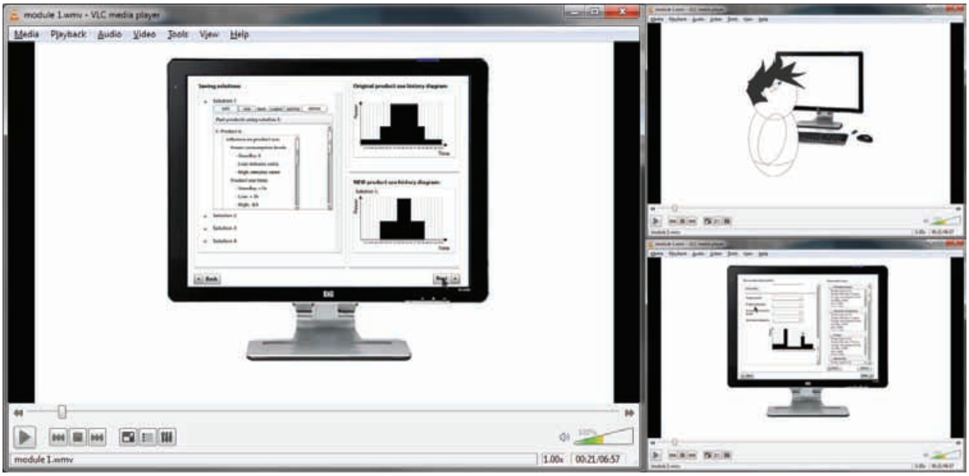

module 1.wmv - VLC media player
Media Playback Audio Video Tools View Help
Saving solutions
Original product use history diagram
NEW product use history diagram
Time
Back
Next
module 1.wmv
1.00x
00:21/06:57

Figure 4.12. Screenshots from the designers-computer interactions module

of interest to all parties; it communicates how the application is used, what information had to be inserted and what information has been retrieved. In Figure 4.12, a selection of screenshots is shown to give an impression of the realization of the module.

4. The fifth module is related to the application data management. This information is important for both knowledge engineers and software developers. The latter should acquire an impression of how data can be stored and retrieved, while the former has to know how data to be delivered will be processed, and how the product designers can retrieve it. Figure 4.13 shows a selection of screenshots to give an impression of the realization of the module.

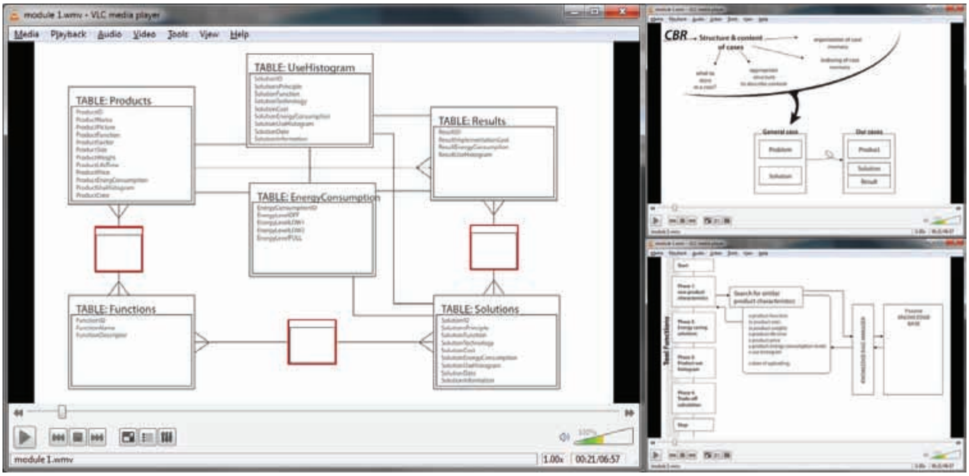


Figure 4.13. ***Screenshots from the data management module***

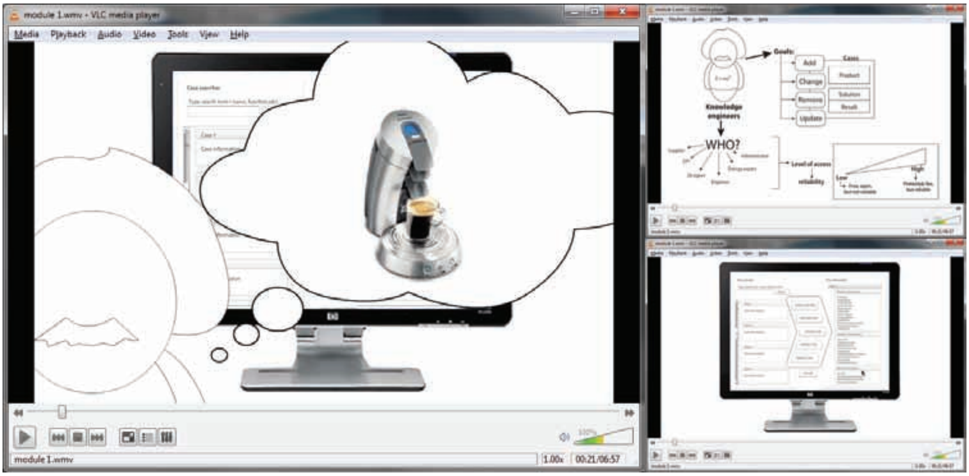


Figure 4.14. Screenshots from the knowledge engineering interaction module

5. The last module is related to the knowledge engineering part of the software. It should communicate how the knowledge base will be kept up to date and what opportunities are offered to knowledge engineers for that purpose. Figure 4.14 displays some screenshots of the module.

4.4.2. Development of the MAP

The prototype building

To develop the application's prototype, animated movie with different forms of visual representation was chosen for the enactment, depending on the contents of the modules.

For module 1 and 3 we used storyboarding; for module 2 and 5, flow chart animations were selected; module 4 and a part of module 6 were presented using simulation of user interfaces. For the remaining part of module 6, storyboarding has been selected again. Regarding narration, we used personas [48] adopting the roles of end-users. The narration was scripted on paper and recorded orally afterwards. For the realization of the enactment, line drawings and hand sketches were combined with product images, screen shots, and movie clips; animated and recorded in one single digital demonstration media using Adobe Flash Professional CS5 and Adobe premiere CS5 (See Figure 4.15). In Figure 4.10-4.14, the selection of screenshots provide an impression of the resulting prototype modules visualization.

Outlining the discussion

In addition, the discussion for each of the focus group sessions had to be developed. Information conveyed through the MAP modules should support educated discussions and informed decision making, resulting in strongly funneling change proposals. Based on the work of Krueger and Casey [52], we defined following groups of questions for each group of stakeholders: (i) understanding questions about the complete application, (ii) general questions on each module, (iii) specific questions on specific aspects in each module, (iv) concluding questions for cross-verification, and (v) methodological questions on the MAP methodology itself. In Figure 4.16, the questions that were discusses with the designers-stakeholders are shown.

4.4.3. Working with the MAP in focus group sessions

Execution technique

To learn stakeholders' reflections, we used passive (observant) stakeholder participation in focus group sessions. During the execution of the concept confirmation, per profile focus group sessions were organized, hosting relatively small groups of people (6-12 participants), addressing specific topics, which are either matched to the characteristics of the participants, or varied according to the specific interest of the researcher [52]. The reason why the method of focus group sessions was chosen is that focus group sessions are generally relatively easy to assemble, and the experimentation is inexpensive. Furthermore, focus group sessions provide rich data through the constant interaction [50].

Number of sessions

Taking into consideration the different profiles of stakeholders, we organized four sessions with product designers, two sessions with software developers, and two sessions with knowledge engineers. The number of focus group sessions for the most important group of stakeholders (product designers = end-users) was decided based on the rule of thumb of Krueger and Casey [52]. They proposed to plan three or four focus group sessions in a queue. They advise that once you have conducted these, determine if you have reached saturation, which is the point where no new information can be expected to emerge from sessions with additional groups.

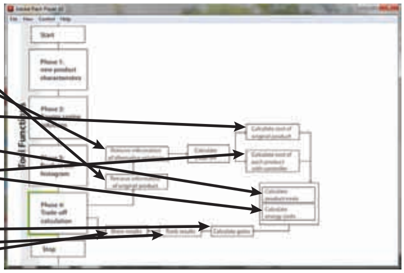
Narration

Enactment

Module tool functions
Episode 1.6: phase 4

The tool offers various actions to support the calculation of the trade-off. In the fourth phase of the design process, the tool should retrieve the information of the original product characteristics and of the alternatives that were generated to save energy. It does it in order to calculate the cost of the original product, based on the consideration of the actual product cost and the cost of the consumed energy, and to do the same for the new product equipped with ubiquitous controller. Actual the tool is able to do it for each alternative solution. Based on the calculated costs, the tool determines the gains. As a final action in this phase, the tool ranks the result of the trade-off calculation and visualizes this information on the

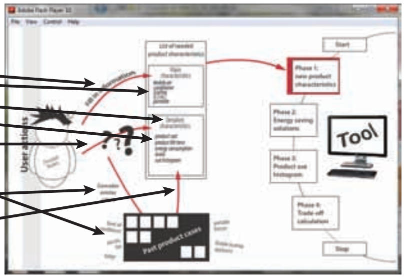
Enactment 1.6:
animated flow chart diagram


Tool Functions
Start
Phase 1: new product characteristics
Phase 2:
Phase 3:
histogram
Phase 4: Trade-off calculation
Stop
Calculate cost of original product
Calculate cost of each product with controller
Calculate product costs
Calculate energy costs
Calculate gains
Rank results
Show results

Module user actions design
Episode 2.3: reasoning about the new product specifications

In the first phase, Harry will use the tool to specify the intended characteristics of the new product. He must be ready to fill in the slots of the inquiry window by providing information about the main functions, the size and weight. This is not difficult for Harry, but giving an estimate on other characteristics such as energy consumption, product cost, product life time etc. are much more challenging to be described, in particular in this early stage of development. To support him to cope with this difficulty, the tool offers past cases that Harry can bring through, search for the relevant ones and can select the most relevant cases as guides. He can compare the latter cases and when the tool provided him with the average values of the characteristics, he should make a decision. He has to take into consideration at forming a decision how much the individual cases fulfill the specifications.

Enactment 1.6:
animated schematical picture


Adobe Flash Player 10
File View Control Help
User actions
List of needed product characteristics
Fill in information
Start
Phase 1: new product characteristics
Phase 2: Energy saving solutions
Phase 3: Product use histogram
Phase 4: Trade-off calculation
Stop
Tool
Past product cases

Module tool-user interactions design
Episode 3.5: interaction on the screen of the product use histogram

The appearing of the window with the product use histogram just needs Harry pressing the 'Next'- button. Harry can see all the selected solutions on the left side of the screen. In the upper right corner, the use histogram of the original product is representing the energy consumption levels and usage time. In the lower right corner, Harry can see an empty use histogram that must be filled in for each of the selected solutions. To know how the technological solution effects the products' energy consumption, examples of other application cases of the energy saving solution are represented. Harry can edit the diagram for each and every solution. Otherwise, at any time he might delete the solution if he realizes that it is not applicable. He can also obt for combining solutions, in order to achieve higher energy savings. As he wants to edit the diagram, he can start with a blank diagram or can use the one of an existing product as a starting point. When Harry has completed this for each solution, he may continue with the next task.

Enactment 1.6:
screenshot animations

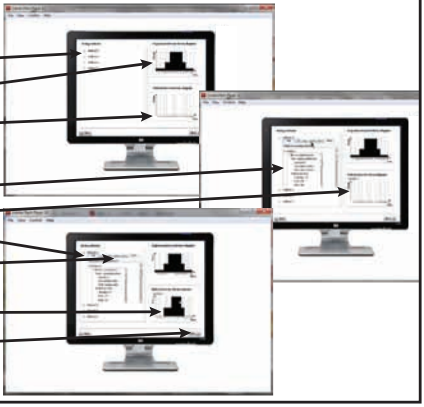


*Figure 4.15.	Examples of the elements of the narration and the enactment parts of the AP*

General Questions

	TF$_D$ Tool Functions Design	UA$_D$ User Actions Designer	IA$_D$ Interactions Designer
Module/ functional unit	GQ3	GQ3	GQ5
Phase 1: New Product Characteristics	GQ1	GQ6	GQ4
Phase 2: Energy Saving Solutions	GQ1	GQ6	GQ4
Phase 3: Product Use Histogram	GQ1	GQ6	GQ4
Phase 4: Trade-Off Calculation	GQ2		GQ4

Specific Questions

	TF$_D$ Tool Functions Design	UA$_D$ User Actions Designer	IA$_D$ Interactions Designer
Phase 1: New Product Characteristics	SQ3	SQ1	SQ2
Phase 2: Energy Saving Solutions		SQ1	SQ2
Phase 3: Product Use Histogram	SQ4	SQ1	SQ2
Phase 4: Trade-Off Calculation	SQ5	SQ1	SQ6

Questions for designer stakeholders

Understanding questions:

UQ 1: What is your first impression about the software tool?

UQ 2: Does it seems to be easy & transparant to use the tool in practice?

UQ 3: Have you become convinced that the tool can be used for other products as well?

General questions:

GQ 1: Is case-based reasoning a good strategy to support designing for energy saving?

GQ 2: Do you think that the proposed trade-off estimation approach is an effective one?

GQ 3: Are the tasks properly assigned to user actions and system functions?

GQ 4: What is your opinion about the number of necessary interactions?

GQ 5: Which implementation of the tool would you prefer?

a) software on computer or laptop,
b) webhosted
c) smart phone application

GQ 6: Would your design creativity and freedom be reduced by considering past product cases?

Specific questions:

SQ 1: Does the structural procedure offered by the tool fits into your daily design practice?

SQ 2: What alternative representation can be used to visulaize the cases for the user?

SQ 3: Does the tool provide sufficient means for describing the new product characteristics?

SQ 4: Does the product use histogram provide sufficient information about the product usage?

SQ 5: Should the trade-off calculation be visualized?

SQ 6: Is there any better way to present the end results of the trade-off forecasting?

Concluding questions:

CQ 1: What would you improve?

CQ 2: Can you identify missed opportunities?

CQ 3: What is the most important point we discussed?

Figure 4.16. Discussion questions for the designer-stakeholder

Inviting participants

Stratified homogeneous sampling was applied, c.f. Patton [54]: participation in a group interview about major issues might differently affect them. The participants were selected based on the above mentioned profiles. The minimum sample size: n = 7 people per session. Anticipating no-show ups we over-recruited each group by inviting 10 persons. The sessions were held in Belgium and in the Netherlands.

Conduct of the sessions

Each session was organized as follows: First, the participants got a brief introduction, explaining the goal of the session, and how it would be structured. They were also informed that, in case of their unanimous confirmation, everything would be recorded for the data processing. After this introduction, the modular abstract prototype was presented in approximately 30 minutes for the participants. The digital demonstration material was composed of two general modules (General introduction and Design tool functions) and different number of specific modules dedicated to particular stakeholders. After the demonstration, a reflexive discussion was held based on a list of predefined questions. During this discussion, the questions were projected by using PowerPoint slides, together with some explanatory images taken from the digitally recorded demonstration material. The latter served as reminders to critical argumentations and explanations embedded in the modules of the abstract prototype. Because of the large number of aspects and questions to be discussed, exactly five minutes have been allocated to each question. It should be noted that the sessions were facilitated by two moderators. One of them, the owner of the research project, guided the session and safeguarded the proper tracks of discussions. The other moderator helped with session administration, time-keeping, and making notes on the discussions.

4.4.4. Data evaluation and conclusions

In order to be able to interpret and structure the data of the focus group sessions, the following plan was made:

1. All sessions were recorded, and as a first step in the analysis, all sessions were transcribed.

2. Next data exploitation was carried out to identify how much the data is in focus of each question. We started with congruence mapping to dissect relevant and irrelevant information, based on the distances between answers and questions (scale 1-10: 10 = highest semantic congruence). Here, we observed that non-congruence does not necessarily indicate irrelevance.

3. As a next step, semantic data re-coding was conducted, to handle cases in which congruence with the question was imperfect. Several semantic content-implied keywords were identified to re-code and restructure the data. Again, congruence mapping was conducted to eliminate irrelevant information, now the semantic distance was a base to define the concept relevance.

4. Further data reduction was completed, all information of different focus group sessions was merged, with the objective to strengthen the semantic meaning contained, eliminate duplicates and convert results into information structure. We considered using tools such as discussed in [55] to automate this semantic mapping, however we preferred not to use them in our case.

5. Defining the operation field was the next action; we created an overview of all grouped results of the sessions. Each result was assigned an impact and a frequency indicator. The impact number on a scale of 1 to 5 (5: impact on software basics, 4: impact on extended software; 3: impact on software implementation; 2: impact on commercialization; 1: other impact) and the frequency number being the number of times a topic was mentioned over all sessions. The impact of the different semantic results was weighted by a factor of each semantic topic (same impact indicator). The result of the operation field is visualized in a diagram shown in Figure 4.17. We have to mention that, to cap peaks, a ceiling and a threshold have been applied.

6. Next the operating domain was defined. By specifying different operating domains, we focused on those results having highest impact and frequency, offering the best opportunities for improvement. These are visualized in Figure 4.17 as iso-curves.

7. Based on these different domains of interest, the results were ranked. To continue the data processing, we eliminated all data that is below the third curve (Curve C in Figure 4.17), since either their impact and or their frequency is too low to be taken into account.

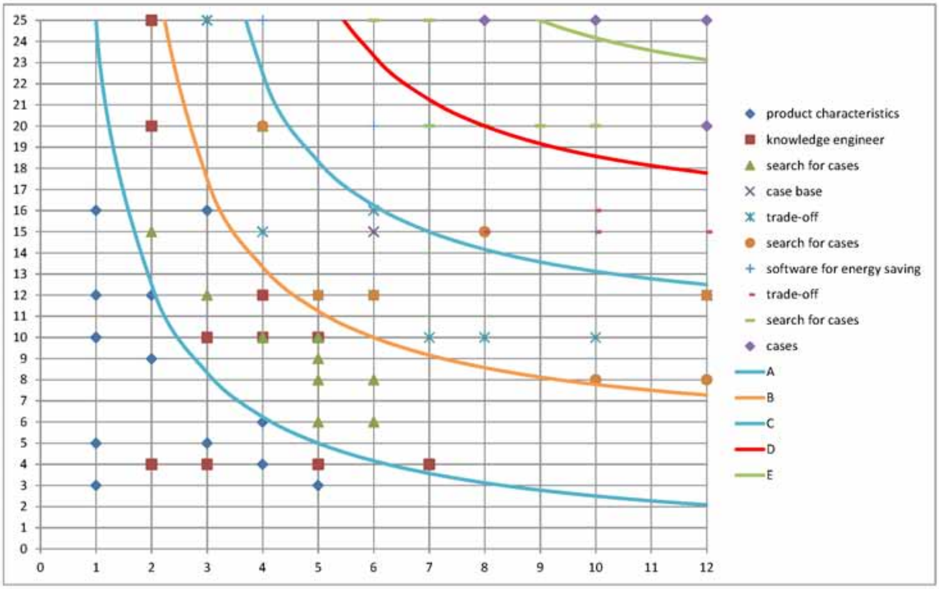


Figure 4.17. Operation field and search curves.

8. Lastly, these results were regrouped according to their conceptual level before they were interpreted. These results are shown in Table 4.3.

Results of the pre-testing

All experts agreed on that the abstract prototype facilitated the presentation of the functions and the implementation details and added a lot of value to demonstrate the concept of the software tool. At the same time, they also indicated that the total length of the prototype

had to be reduced to a maximum of 20 minutes. This meant that there was a need to

Table 4.3. Results of the data analysis organized according to conceptual level (continued on next page)

Semantic sub-topic	*CL	Related to
It should be considered when the tool should be used and to focus on one phase in the process: (i) inspiration tool in the design process (guiding, directing) or (ii) as decision making tool (exact comparison)	4	context
The decision making process of the designer is not a linear process but an iterative	4	context
Regarding the implementation we should focus on a web-application	4	context
We should consider the vagueness of the trade-off result and the value of the outcome. (Or it should be included in a way that the user does not achieve the impression that if they put uncertain information in to the system, they receive "certain" information in return)	4	trade-off
Database structure should be considered	3	case base
Different kinds of information can be in a case: closed case, an incomplete: only an idea, a principle, solution, technology, product use case, result …	3	case base
How will the database handle the different levels of information: product – function – solution – component – technology - …	3	case base
Products are complex containing different sub-functions, each having other saving potentials	3	complexity
What is the relationship between the computational part of the software and the knowledge base, where do they interact and what kind of information is exchanged?	3	system
If product have multiple sub-functions and sub-functions might have multiple solutions, these should all be combined and the influences of the solutions on each other should be taken into consideration	2	complexity
What is the exact task of the knowledge engineer, the most repeated proposal: one administrator knowledge engineer, and different other people who have an interface to insert information, which is checked by the administrator before uploading them, to watch over the reliability.	2	knowledge engineering

Table 4.3. Results of the data analysis organized according to conceptual level (continued from previous page)

Semantic sub-topic	*CL	Related to
Searching cases should start from the principles and then further be split up	2	system
The search for cases will be a further detailing and focusing unfolding technique	2	system
Searching solutions must be possible by searching (sub-)functions, product characteristics, components, technologies, …	2	system
The number of steps a designer must go through should be minimized	2	system
Principles of saving energy can be product-dependent and product independent: the independent should be suggested as probably interesting if not applied in a product	1	case base
Part of the task of the designers during the identification of the product characteristics is to identify the functional parts of the product	1	complexity
It would be interesting to know which sub-function of the product is the most consuming and has the largest potential for energy saving	1	complexity
Consider the ontology (vocabulary) of the items in the cases and in the case base to create consistency and to simplify search	1	complexity
The complexity (sub-functions, components, multiple solutions) should be inserted in the use histogram	1	complexity
The complexity (multiple functions and multiple solutions) should be inserted in the trade-off calculation	1	complexity
Examine what case information will be shown in each step	1	system
Reconsider the parameters of the trade-off calculation: should comfort be included, should they all have a load/weight, do external/context influences be included …		trade-off

*CL = Conceptual level

rearrange and reorganize the full-scale research sessions. One group of stakeholders, the participating product designers argued that the “Design tool functions” module was not really useful for them to answer the questions. They also proposed a restructuring and bundling of the chunks of information concerning the up-date of the knowledge embedded in the system. Originally, these were scattered over multiple modules, but they have been compiled into one module in the final version of the MAP.

Results of the full-scale research

The objective in our confirmative experiment was to collect the opinions of stakeholders in order to be able to enhance and optimize the concept of the software tool. By using a list of prepared questions during the focus group discussions, the findings could be structured according to the keywords, they were referring to. The full-scale research provided three major groups of results. These are: (i) data on the technical functions and implementation of the software, (ii) data on the utility of the software, and (iii) data on the used demonstration method. In general, the data related to (i) and (ii) included directly usable enhancement proposals. These could be used to develop a refined version of the software tool. On the other hand, some of the propositions required further investigations, for instance, discussions with other stakeholders, or expert interviews, or literature review.

In short, these are the findings of the research concerning the technical functions and implementations of the software. On the highest conceptual level, the context of software tools use was discussed in three topics: (i) the phase in the design process is a crucial aspect for the software use. Here there are two options: as inspiration tool in the design process (guiding, directing) or as decision making tool (exact comparison). The focus can be on both but a good differentiation is crucial. (ii) The decision making process of the designer is an iterative process instead of a linear. The software should allow the designer to iterate. (iii) Regarding the implementation, a web-based application is the most interesting. Another important aspect that was discussed on this conceptual level is the vagueness of the outcome and consequently the value of the trade-off result. Further discussion is needed to consider how to deal with this, and to see the added value of the trade-off calculation.

On the next conceptual level, topics related to the case base and to the complexity of the software emerged. Concerning the case base, the database structure is very important, different kinds of information can be in a case: closed case, an incomplete: only an idea, a principle, solution, technology, product use case, result… So we must consider how the database will handle the different levels of information: product – function – solution – component – technology. In addition to these different levels of information, the complexity is enhanced more because the products in themselves are also complex since they have different functions, each having other saving potentials that probably influence, boost or limit each other. Furthermore on a lower, system level, of the concept, the participants were suggesting how to implement this into the system, how decomposition of the search for solutions should be performed, when the system should communicate with the database and what information is needed. In addition also the other side of the case base was discussed. To specify the exact task of the knowledge engineer, they suggest to have one administrator person who is verifying all information before uploading in the system and who should maintain the database, in addition multiple other people, such as suppliers should have the possibility to insert their data.

4.5. Confirmative experiments and studies

4.5.1. Explanation on the general conduct of the confirmative research

The objective of the confirmative research is to test the effectiveness and efficiency of the MAP methodology. To achieve this, we applied the MAP methodology to our reference case. The method of empirical testing using a concrete application case was used in the reasoning-with-consequences strategy. We applied this strategy to verify the suitability of our methodology in use context. No other theoretical or non-experimental strategies could be considered for this confirmative testing. We choose this approach also for pragmatic reasons, as no other medium-fidelity prototyping methodologies were available to compare with at the time. In our application case, the MAP methodology has been applied and tested in the design and use context of developing the reference case, namely the proposed concept of a knowledge-intensive software tool. This tool supports product designers in decision making on ubiquitous computing augmentation of energy intensive products [56], in which MAP was used to improve (optimize) product software concepts before implementation. Observations and recordings contain results belonging to the use context of the application, but in addition, additional observations have been collected for the empirical verification of our MAP methodology itself. The aim of the verification was to assess the methodological and functional accuracy and applicability of the MAP methodology in a use context of software tool concepts design, and to uncover hidden methodological and application limitations and restrictions, vulnerabilities and necessary operational conditions on use context as well as on users of the methodology. Due to its relative complexity, the proposed software tool is an adequate testing case, and MAP could be applied without any constraints or limitations.

4.5.2. Organization of the experiment

Research was organized according to the procedural steps, explained in 4.3.2. The conceptualization and design of the prototype were carried out and focus group sessions were organized to test the software concept of the case. In these sessions nothing was mentioned about verification and validation of the methodology, in the introduction we explained that the goal was to test the concept of the software case. Next, the prototype was demonstrated and a reflexive discussion was held based on predefined questions. Five groups of questions were discussed, of which the last one was related to the MAP methodology.

4.5.3. Raw data generated

Most data generated is related to the test case, apart from the question in which the stakeholders were asked to judge the used methodology. All focus group sessions were recorded both with a camera and with a Dictaphone, to make sure that all information would be available afterwards. The camera recordings had the extra advantage that body language sometimes gave additional information on how people felt and behaved during the sessions.

4.5.4. Coding, processing and interpreting data

The data that needed to be interpreted, analyzed and processed is related to the outcome, process and methods applied in the reference case and to the answers stakeholders gave on the methodology-question. Useful in validating the goodness of the methodology is by the criteria of goodness in 4.3.4. We found that the MAP methodology was an effective means supporting the harvesting of directive feedback in educated discussions in focus groups. In general, a fair amount of data could be generated during focus group sessions for our application case study, clearly displaying its effective role as a stakeholder-tailored, information-rich demonstration means. The experiment closely followed the MAP methodological steps, explained in Section 3: conceptualization and design of the prototype were achieved and focus group sessions were organized to test the software concept of the case, and the results were analyzed and processed into an improved software concept. Having outlined the methodology explicitly, we found that focus groups could easily adopt the methodological steps; confusion about the steps was not observed. We also found that, this way, attention can fully go to the informed decision making. It also renders results from different sessions compatible, so that results can be merged later on. We found that in each of these steps, stakeholders, even from different groups, were able to discuss and merit their input. This led us to believe that this underpins our conjecture that the MAP methodology breeds medium-fidelity prototyping, and MAP prototyping obtains its medium fidelity from explicit-ness in content, in focus, but also from the embedding MAP methodology. During the focus group discussion we asked participants an additional, confirmative question regarding their impression of the used MAP methodology.

Participants' opinion on the used method

As the last question we asked the participants what they think about the demonstration methodology and means (MAP) that was used in the focus group sessions. In addition to addressing the points of the question, the participants offered some other remarks, which were not directly related to the MAP methodology. In general, all stakeholders were positive about the use of the methodology in the early phase of appliance and software. As main advantages they mentioned that using MAP: (i) provides a structured support for the discussions, (ii) guides the thinking of the people in the same direction, and (iii) does not requires the participants to generate an all-embracing complete picture, which is often a problem. The participants also commented on that extra research is still needed to optimize both the content and the framework of the software tool.

4.6. Confirmative research concerning the MAP methodology

4.6.1. Justification of the MAP methodology

An indirect justification for asserting the empirical statement of truth was chosen, as there is no means for bringing into existence a direct justification. By reasoning with the consequences of the theory I was able to scope its properness and to identify the limits of the MAP's applicability. The result could convincingly be expressed by the execution in the reference case. Reasoning with the consequences of the test result, we concluded that the MAP was logically error free by discussing its reliability, consistency and cohesion.

Reliability

The methodology was implemented in the development of the reference case, to examine if it was executable. The application of the methodology was completed successfully; here we discuss its most important aspects:

- Homogeneity of the focus groups: stakeholder homogeneity is important when working with focus groups, and a reasonable, but hard to quantify level of homogeneity is required for effective MAP results. To achieve this, in addition to a careful selection of participants, we also balanced their knowledge about the objectives and conduct of the sessions. Together with an invitation letter, a questionnaire was sent to each participant in support of identifying their profiles. However, the homogeneity intended could not be achieved in all cases and compromises had to be made as a result of availability of participants. Despite, the achieved level of homogeneity was reasonably high, which was evidenced by the fact that there were almost no misinterpretations.

- Language: another relevant finding is that the language used in the MAP and practiced during sessions matters. Our estimate was that the language should typically be English, both for the MAP and for the sessions. In all modules we used English. However, focus group sessions took place in Flanders and in the Netherlands, and most stakeholders preferred Dutch for the sessions. Apparently, it was easier for participants to discuss in their native language, even though questions guiding the discussions were presented in English. Thanks to the (agreed) recording, all raw data provided in Dutch could be transcribed into English text with excellent fidelity.

- Duration of MAP demonstration: it is known from literature documented experiments that people can listen attentively at most 20 minutes [57]. Therefore, we restricted the duration of digitally recorded demonstration material. Obviously, the density of information conveyed to the human intellect through the perceptive and cognitive communication channels is another relevant factor. Our experience was that the above duration was appropriate for most participants, but the intensity of information transfer posed a challenge to some of them.

- Number of modules: the information relevant for stakeholder groups is embedded in different modules. The larger the number of modules, the larger MAP composition flexibility can be expected. On the other hand, an extreme number of modules would lead to a combinatorial explosion. In our application case experimental sessions, different compositions of modules for different stakeholders were explored, in intense information elicitation processes. A near-optimal resolution of modules was found when we applied six modules, yielding appropriate content articulation to the three involved stakeholder groups. Here, a good trade-off was found between flexibility required, reusability of the modules, adaptability, and complexity. This number is an important conclusion from our experiments, putting us on a track of an important rule-of-thumb. Further experiments are needed to assess this rule, however.

- Structured discussion: by structuring the discussions with pre-formulated questions, we noticed that the participants were (re-)activated every time we showed a new question. This observed effect is considered to be very positive since it maintains the attention and keeps the focus of the participants.

- Number of required iterations: by using the MAP methodology the number of required iterations could be reduced. We assume that this is caused by the clear information demonstration and deep questioning and discussions that lead to strong change requests.

- Fidelity: the level of fidelity has been a constant issue as we described the prototype; we designed our methodology such that it can set forth MAP prototype of low- and medium-fidelity. We reason that, to that end, the methodology has to support generating medium-fidelity prototypes. Next, all phases discussed have to be executed so that, indeed, a medium-fidelity prototype is generated.

Our methodology dissects stakeholders, their decision making information, their decisions and their change proposals purposely and educated, with the communication attributes that control communication process across the methodological steps. The methodology supports and promotes this consistently, offers guidelines on how produce (input, skills, methods, tools…), to document it for evaluation and reproduce-ability, and permits any fidelity (“right-fidelity”; [58]) level of doing so. A module as such is a self-contained information structure supporting the contribution by one focus group; adding or removing groups can be accomplished at a module-by-module basis, whilst the MAP methodology preserves the consistency across the whole. We induce from this reasoning that the methodology supports low- and medium-fidelity prototyping. In the application case, we chose evaluators (stakeholders) to be observant. We did this to prevent complexity from being stacked up. Although not strictly necessary, we reason that for medium-fidelity, evaluators should be active, not observant. Main functionality, interactions, configuration and visuals can be presented to stakeholders in an operable, approximately final fashion. This is in line with our concept of medium-fidelity. We may argue that tools such as PowerPoint, Mockingbird, and Adobe Fireworks are in support of medium-fidelity [58]. Nevertheless, we prefer to further investigate the fidelity issue in later research work, and feel that, as of yet, we cannot convincingly claim medium-fidelity to have been demonstrated for the MAP prototype. We applied the complete methodology in the application case and found no obstacles or inconsistencies. Therefore we do think it is justified to say that the methodology supports medium-fidelity. As such, the MAP methodology supports a full-scale assessment of the intended software, which is not considered to be achieved by low-fidelity prototypes. We argue that MAP requires demonstration resources typical of low-fidelity prototyping, but offers medium, perhaps even high-fidelity demonstrative capacity because it supports demonstration of the complete software concept, supporting developers in their development, testing and verification activities.

Consistency

The MAP was also found internally consistent. In the process of the MAP, two important aspects emerged regarding possible conflicts:

- Relevant information: here, a decision making issue popped up regarding stakeholder information relevance: we were not able to present them a single overall picture, because they have different mind-sets and relationships to the application, and were consequently interested in different aspects of the application. To remedy, we consulted some stakeholders in advance. In these informal and open discussions, we managed to obtain hints and clues on what to embed in the modules, and what feedback to expect from our questioning.

- Representation of information: tailoring enactment to stakeholder views showed another issue in this research. We had to pay attention to how stakeholders visualize their information (their communication modality): whereas product designers prefer visuals and images, software programmers think in flow charts and algorithms, other stakeholders may not be able to comprehend these modalities. Different modalities and media had to be applied in the modules, therefore. We also observed that the overall influence of media selections was large in our experimental work.

Cohesion

Cohesion can be measured by checking the facilitation of one methodology to other theories. As we could not discuss the cohesion of the previous methodology in the previous chapter because no other theories were discussed, we have the opportunity to check here the cohesion between the CCR methodology and the MAP methodology. We concluded that by using the CCR in the framework ideation phase, all needed information was available to start go further with the MAP in the concept integration phase. The enhanced framework was received as outcome of the CCR process, and was taken over to the next phase. To be more specific, the conceptual framework consists of different modules that had to be developed. These modules and sub-modules were the starting point to detail the concept in the MAP.

4.6.2. Internal validation of the MAP methodology

In this Sub-Section, the internal validity of the experiment was discussed. Validation may focus on multiple aspects, however we decided that construct validation, content validation and sampling validation were the most appropriate ones here. The method used for validating the methodology was logical reasoning on the aspects that delivered the solution.

Construct validation

The first aspect is the construct validation: As it is important to validate if what had to be measured was truly measured. Therefore, we investigated the different constructs or elements that were used during the operationalization of the MAP. In our MAP methodology, we could identify following main elements: (i) the content of the to-be demonstrated concept, (ii) the context of the to-be-demonstrated concept, (iii) the different modules, (iv) the narration and enactment, and (iii) the focus group organization and execution. All aspects were considered during the application of the MAP to the reference case. The methodology of MAP was developed to validate the software concept in the concept integration phase. In the process, the concept was prototyped and groups of stakeholders were involved in a discussion to provide change proposals to improve the concept. We can conclude that the application-experiment was a valid approach as we could observe how the desired effect was achieved and how the different modules of the MAP methodology were used to derive the change proposals and to improve the concept.

Content validation

In the content validation, we measure the extent to which a measure represents all facets of the MAP methodology. The measure used in this research was the applicability of the MAP methodology in a reference case. The methodology of MAP was developed to increase stakeholder involvement in the concept development phase, as by a concept-demonstration prototyping means, stakeholders were involved in the validation of the concept. Using the reference case, an experiment was set up to test if the MAP indeed supports towards the concept validation. We can conclude that the application-experiment was a valid approach as we could observe how the desired effect was achieved plus how the methodology that supported towards it was performing. The selected reference case can be considered valid to measure the applicability of the methodology, as it belongs to the core of the operation domain.

Validity of sampling

Lastly, we discuss the validity of sampling. Stratified homogeneous sampling was applied as sampling strategy. Our impression has been, and this is also proven by the results, that this was an adequate strategy, because the participants were selected based on the needed profiles. Furthermore, in the qualitative research the aim was to obtain good change proposals in a short time range. The participants were selected based on previously defined stakeholder-profiles. Three different groups of stakeholders were identified: (i) product designers, (future end-users of the application); (ii) software developers (programmers of the application); and (iii) knowledge engineers (for knowledge processing about cases from the past in the case-base). Sessions were organized per group of stakeholders, each with a minimum sample size: n = 7 people per session. We organized four sessions with product designers, two sessions with software developers, and two sessions with knowledge engineers.

4.6.3. Consolidation of the MAP methodology

Consolidation has two aspects to discuss, the de-contextualization and the re-contextualization. De-contextualization or generalization is not considered to be relevant for the MAP methodology. We could argue that the method of modular abstract prototyping, next to its value for software development, can also be used in other contexts such as appliance development, or service specification, or product-service combinations, where the real-time processes should be optimized to multiple criteria. Nevertheless, further research efforts are needed into these directions to make good statements regarding the usage of modular abstract prototyping. As we do not want to use the MAP out of the context of the entire DSDM, we do not consider it to be important here.

The re-contextualization or specialization is more important here, regarding the reference case and also regarding the information that is transferred to the next cycle in general. We have to consider how the information that comes out of this phase will be used in the next phase, i.e. what the context and the objective of next phase is and how the current knowledge is useful for this. The aim was not to validate and verify the reference case, but it was needed to verify and validate the application of the MAP methodology. The concept of the reference case is taken over to next phase, in the form of a complex function structure. The findings of the focus group sessions were interpreted and design decisions were marshaled towards an improved software concept. Interpretation was especially needed for the following issues: (i) whether the tool will be used for inspiration, information, navigation, and or calculation, (ii) how to deal with complexity in multi-level databases, (iii) how to deal with complexity in multiple solutions and functions and technologies, (iv) vagueness of the trade-off result, (v) role of the knowledge engineer. The results were implemented in new function sets, as shown in Figure 4.18, that are useful in the next research cycle.

4.7. Overall discussion and conclusions

4.7.1. Discussion

As discussed in [47], abstract prototyping enables rapid ideation, modeling, and demonstration of concepts in early phase of development with the objective to receive extra information about the functioning, quality, features, properties of the demonstrated concept to modify or improve the initial concept. In general, the major benefits of abstract prototyping are that it (i) influences the most creative phases the development process, and (ii) opens the way towards intelligence that cannot be obtained otherwise. One recognized limitation of using general abstract prototypes is that they are structurally monolithic and offers no real time opportunities for content changes. Therefore, they are not the most suitable to help find answers to ‘what-if’ or ‘why-not’ type of questions. Contrary to these observed limitations, generic AP can be useful in many application fields as a very useful demonstration and thinking stimulation means.

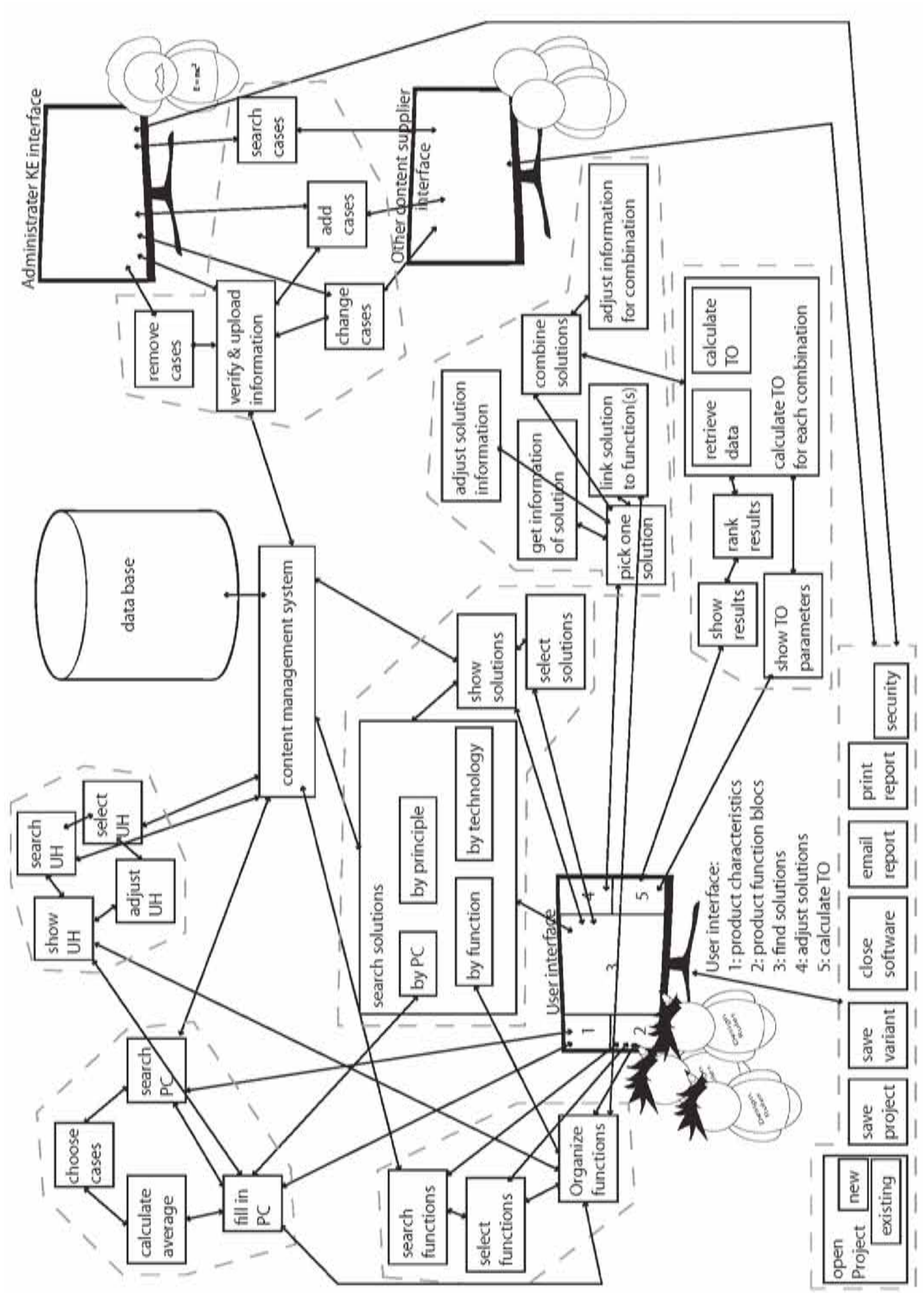


Figure 4.18. *Updated function sets*

Since the limitations of generic AP can be traced back either to issues associated with its constrained flexibility and the lack of sensitivity to multiple stakeholders, or to the amount of effort needed to handle the complexity inherent in this form of abstract prototyping and the amount of work needed to implement the narration and enactment parts in an integral, fluent and attractive way. In order to increase flexibility and eliminate some of the typical bottlenecks we have worked out the concept of modular abstract prototyping, and developed a methodology that can supports its application in various academic research and industrial development projects. The major objective was to offer a possibility for a quick development of contents in the context of multiple stakeholders by allowing a flexible combination of different modules into one specific abstract prototype. Modularization also lends itself to a higher level of reusability, that is, different combinations of abstract prototype modules can be combined according to the views and demands of stakeholders. This can also reduce the complexity of the content development efforts, as well as the incurring costs.

Evidently, modular abstract prototyping is a significant step towards a flexible and efficient early prototyping methodology, but it still can be further developed. MAP does not allow to introduce changes in the technical information content and to adapt the demonstration contents accordingly. This would need a fully interactive, real-time emergent approach, which has been called interactive abstract prototyping (IAP). The methodology of MAP has been developed without considering this particular objective. It has to be also mentioned that IAP needs a specific computer tool that supports not only the presentation, but also collecting content and context information. Our research is going in this direction and a functional framework and an implementation plan are being developed for this tool, which has been conceptualized as a web-hosted information hub, equipped with means for real-time editing of abstract prototype contents. The benefit of this would be a much shorter feedback loop between the prototype developers and the stakeholders, and a dynamic feedback to the concept developers based on a more comprehensive investigation of the stakeholders.

4.7.2. Concluding remarks

We concluded this research cycle with following propositions on the MAP methodology:

Proposition 1:

MAP offers the possibility of rapid development of modularly configurable and presentable content, supporting focused demonstration to stakeholder groups and their decision making process , by using an operational abstract prototype.

Proposition 2:

We sought to pair the advantages of high fidelity prototyping with the low cost of low-fidelity prototyping at early stages in the development.

Proposition 3:

The MAP methodology facilitates demonstration and early validation of software concepts with stakeholders.

Proposition 4:

Modularization is the key to fit different stakeholder groups with communication modalities of their preferences and interests

Proposition 5:

FGS is an efficient way to evolve educated and informed opinions on requirements fulfillment of a group of SH in the form of prompted feedback during demonstration sessions.

Proposition 6:

MAP is also useful for software developers because it reveals how a future tool would work in a real-life environment per SH focus.

Proposition 7:

Using a modular prototype structure enhances content development flexibility, criticality needed to serve multi-focused SHs and break down complexity. A (near-) optimal resolution allows sufficient and adequate content articulation, fewer modules hinder flexibility and reusability, more modules increase complexity, driving feedback beyond the information saturation point.

Proposition 8:

Since modular abstract prototypes are working both in the perceptive and the cognitive channels of human communication, they contribute to a rapid formation of a shared awareness and understanding among software designers and stakeholders. This leads to a significantly deeper and more rigorous assessment of the proposed concepts by stakeholders, and to a more consolidated feedback and enhancement proposals to designers.

We finally conclude that the method of modular abstract prototyping is especially useful for software development, although at the same time we argue that it can also be used in other contexts, such as appliance development, service specification, or product-service combinations, where real-time processes are to be optimized for multiple criteria. Further research efforts are needed into these directions to optimize the usage of modular abstract prototyping.

Future work should also address the following aspects: our current MAP does not support instantaneous technical information content modification in real time (i.e., during sessions). This would require a fully interactive, real-time emergent approach, known as interactive abstract prototyping (IAP). IAP needs a more extended tool that supports not only the presentation, but also collecting content and context information. Our research is going in this direction. Furthermore, decision making and change proposal elicitation and evaluation could be rationalized, based on decision theory. Criteria and measures of effective decision making can be collected and embedded in the modules and profiles, to guide questions to present to stakeholders during the sessions. . This would provide us with full control over the fidelity of the demonstrative capability, and consequently stakeholder feedback quality. The expected impact of this is a further increase of development convergence, as a result of stronger change proposal funneling. Future research into this direction is being considered.

4.8. References

[1] Baelus, C., (2003), "Methodologie van het ontwerpen 1", Productontwikkeling, departement ontwerpwetenschappen, Hogeschool Antwerpen, Antwerp.

[2] Verhaert, K., (2009), "Het verhaal achter nieuwe producten - toegevoegde waarde door productontwikkeling en industrieel design", Stichting kunstboek, p. 240.

[3] Ulrich, K.T., and Eppinger, S.D., (2008), "Product design and development", McGraw-Hill p. 368.

[4] Hakim, J., and Spitzer, T., (2000), "Effective prototyping for usability", Proceedings of the IEEE professional communication society conference, IEEE Educational Activities, Cambridge, Massachusetts, pp. 47-54.

[5] Constantine, L., (1998), "Rapid abstract prototyping", software development, Vol. 6 (10).

[6] Dow, S.P., Heddleston, K., and Klemmer, S.R., (2009), "The efficacy of prototyping under time constraints", Proceedings of the C&C'09, ACM, Berkely, California, USA.

[7] Kyng, M., (1995), "Making representations work", Communications of the ACM, Vol. 38 (9), pp. 46-55.

[8] Derboven, J., Roeck, D.D., Verstraete, M., Geerts, D., Schneider-Barnes, J., and Luyten, K., (2010), "Comparing user interaction with low and high fidelity prototypes of tabletop surfaces", Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries, ACM, Reykjavik, Iceland, pp. 148-157.

[9] Constantine, L., (1995), "What do users want? Engineering usability into software", Windows tech journal, Vol. 4 (12), pp. 30-39.

[10] van Notten, P.W.F., Sleegers, A.M., and van Asselt, M.B.A., (2005), "The future shocks: On discontinuity and scenario development", Technological Forecasting and Social Change, Vol. 72 (2), pp. 175-194.

[11] Brandt, E., and Messeter, J., (2004), "Facilitating collaboration through design games", Proceedings of the Conference on Participatory design: Artful integration, ACM, Toronto, Ontario, Canada, pp. 121-131.

[12] Irestig, M., Eriksson, H., and Timpka, T., (2004), "The impact of participation in information system design: A comparison of contextual placements", Proceedings of the Conference on Participatory design: Artful integration, ACM, Toronto, Ontario, Canada, pp. 102-111.

[13] Rettig, M., (1994), "Prototyping for tiny fingers", Commun. ACM, Vol. 37 (4), pp. 21-27.

[14] Lim, Y.-K., Stolterman, E., and Tenenberg, J., (2008), "The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas", ACM Trans. Comput.-Hum. Interact., Vol. 15 (2), pp. 1-27.

[15] Yang, M.C., (2005), "A study of prototypes, design activity, and design outcome", Design

Studies, Vol. 26 (6), pp. 649-669.

[16] Walker, M., Takayama, L., and Landay, J.A., (2002), “High-fidelity or low-fidelity, paper or computer choosing attributes when testing web prototypes”, Human Factors and Ergonomics Society Annual Meeting Proceedings, Vol. 46 (5), pp. 661-665.

[17] Constantine, L., (2003), “Canonical abstract prototypes for abstract visual and interaction design”, Interactive Systems. Design, Specification, and Verification, pp. 1-15.

[18] Sauer, J., Franke, H., and Ruettinger, B., (2008), “Designing interactive consumer products: Utility of paper prototypes and effectiveness of enhanced control labelling”, Applied Ergonomics, Vol. 39 (1), pp. 71-85.

[19] Snyder, C., (2003), “Paper prototyping: The fast and easy way to design and refine user interfaces (interactive technologies)”, The Morgan Kaufmann Publishers, p. 408.

[20] Liu, L., and Khooshabeh, P., (2003), “Paper or interactive?: A study of prototyping techniques for ubiquitous computing environments”, Proceedings of the CHI ‘03 Human factors in computing systems, ACM, Ft. Lauderdale, Florida, USA, pp. 1030-1031.

[21] Lin, J., Newman, M.W., Hong, J.I., and Landay, J.A., (2000), “Denim: Finding a tighter fit between tools and practice for web site design”, Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, The Hague, The Netherlands, pp. 510-517.

[22] Landay, J.A., (1996), “Interactive sketching for the early stages of user interface design”, School of Computer Science, Computer Science Division, Carnegie Mellon University, p. 262.

[23] Davis, R.C., Saponas, T.S., Shilman, M., and Landay, J.A., (2007), “Sketchwizard: Wizard of oz prototyping of pen-based user interfaces”, Proceedings of the symposium on User interface software and technology, ACM, Newport, Rhode Island, USA, pp. 119-128.

[24] Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N., and Wang, A., (2000), “Suede: A wizard of oz prototyping tool for speech user interfaces”, Proceedings of the 13th symposium on User interface software and technology, ACM, San Diego, California, United States, pp. 1-10.

[25] de Sá, M., and Carriço, L., (2009), “A mobile tool for in-situ prototyping”, Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services, ACM, Bonn, Germany, pp. 1-4.

[26] Wood, D.P., and Kang, K.C., (1992), “A classification and bibliography of software prototyping”, Requirements Engineering Project, Carnegie Mellon University, Pittsburgh, 1992.

[27] Memmel, T., Gundelsweiler, F., and Reiterer, H., (2007), “Agile human-centered software engineering”, Proceedings of the People and Computers XXI – HCI, Linden J. Ball, M.A.S., Corina Sas, Thomas C. Ormerod, Alan Dix, Peter Bagnall, and Tom McEwan (Ed.), pp. 167-175.

[28] Opiyo, E.Z., Horvath, I., and Vergeest, J.S.M., (2001), “Knowledge representation and processing in abstract prototyping of design support tools”, Proceedings of the Design research - theories, methodologies and product modelling, 13th international conference on engineering design, Culley, S. (Ed.), Bury St Edmunds: Professional Engineering Publishing ltd.. Glasgow, pp. pp. 469-476.

[29] Sefelin, R., Tscheligi, M., and Giller, V., (2003), “Paper prototyping - what is it good for?: A comparison of paper- and computer-based low-fidelity prototyping”, Proceedings of the CHI ‘03 Human factors in computing systems, ACM, Ft. Lauderdale, Florida, USA, pp. 778-779.

[30] Wiegers, K.E., (1999), “Software requirements”, Microsoft Press.

[31] Tideman, M., van der Voort, M.C., and van Houten, F.J.A.M., (2008), “A new product design method based on virtual reality, gaming and scenarios”, International Journal on Interactive Design and Manufacturing, Vol. 2 (4), pp. 195-205.

[32] Krishnan, V., and Ulrich, K.T., (2001), “Product development decisions: A review of the literature”, Management Science, Vol. 47 (1), pp. 1-21.

[33] Opiyo, E.Z., Horvath, I., and Vergeest, J.S.M., (2001), “Developing software tools for pre-implementation testing of design support tools”, Proceedings of the MicroCAD 2001 international scientific conference, Icalmar, L., Patko, G. (Eds.), University of Miskolc, Miskolc, pp. pp. 1-8.

[34] Cooprider, J.G., and Henderson, J.C., (1990), “Technology-process fit: Perspectives on achieving prototyping effectiveness”, J. Manage. Inf. Syst., Vol. 7 (3), pp. 67-87.

[35] Shao, G., and Hanna, W., (1990), “Soft prototyping in the design of military electronics”, Proceedings of the Aerospace and Electronics Conference, 1990. NAECON 1990., Proceedings of the IEEE 1990 National, pp. 729-735 vol.722.

[36] Du Bois, E., and Horváth, I., (2011), “Abstract prototyping in software engineering: A review of approaches”, Proceedings of the ICED11, Technical University of Denmark, Copenhagen, p. 10.

[37] Biddle, R., Noble, J., and Tempero, E., (2001), “Role-play and use case for requirements review”, Proceedings of the twelfth australasian conference on information systems, p. 10.

[38] Buhr, R.J.A., (1998), “Use case maps as architectural entities for complex systems”, IEEE transactions on software engineering, Vol. 24 (12), pp. 1131-1155.

[39] Hardgrave, B.C., (1995), “When to prototype: Decision variables used in industry”, Information and Software Technology, Vol. 37 (2), pp. 113-118.

[40] McCrary, N.E., and Mazur, J.M., (2010), “Conceptualizing: A narrative simulation to promote dialogic reflection: Using a multiple outcome design to engage teacher mentors”, Dducation techn research dev, Vol. 58, pp. 325-342.

[41] Robertson, S., (2001), “Requirements trawling: Techniques for discovering requirements”, International Journal of Human-Computer Studies, Vol. 55 (4), pp. 405-421.

[42] Maguire, M., (2001), “Methods to support human-centred design”, International Journal of Human-Computer Studies, Vol. 55 (4), pp. 587-634.

[43] Anastassova, M., Mägard, C., and Burkhardt, J.-M., (2007), “Prototype evaluation and user-needs analysis in the early design of emerging technologies”, Proceedings of the int. conf. on Human-computer interaction: interaction design and usability, Springer-Verlag, Beijing, China, pp. 383-392.

[44] Gray, P.D., Kilgour, A.C., and Wood, C.A., (1988), “Dynamic reconfigurability for fast prototyping of user interfaces”, Software Engineering Journal, Vol. 3 (6), pp. 257-262.

[45] McDonald IV, H.E., (2010), “How mock-ups, a key engineering tool, help to promote science, technology, engineering, and mathematics education”, NASA USRP-Internship Final Report, Johnson Space Center, NASA, 2010.

[46] Bowles, C., and Box, J., (2011), “Undercover user experience design”, New Riders Publishing, Berkeley, CA, p. 191.

[47] Horváth, I., Rusák, Z., Vegte, W.v.d., Opiyo, E.Z., and Koojman, A., (2011), “An information technological specification of abstract prototyping for artifact and service combinations”, Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Vol. 1, Washington, DC, USA, p. 17.

[48] Constantine, L., (2005), “Users, roles, and personas”, in: The persona lifecycle, Pruitt, Aldin (Eds.), Morgan-Kaufmann, San Francisco, p. 18.

[49] Liamputtong, P., (2011), “Focus group methodology: Introduction and history”, in: Focus group methodology: Principle and practice SAGE Publications Ltd p. 224.

[50] Bertrand, J.T., Brown, J.E., and Ward, V.M., (1992), “Techniques for analyzing focus group data”, Evaluation Review, Vol. 16 (2), pp. 198- 209.

[51] Galin, D., (2004), “Software quality assurance: From theory to implementation”, Addison-Wesley.

[52] Krueger, R.A., and Casey, M.A., (2000), “Focus groups: A practical guide for applied research, third edition “, Sage Publications, p. 215.

[53] Du Bois, E., and Horvath, I., (2012), “An easy-to-use methodological proposal for considering ubiquitous controllers in energy use optimization”, in: Design for innovative value towards a sustainable society, Matsumoto, M., Umeda, Y., Masui, K., Fukushige, S. (Eds.), Springer Netherlands, pp. 344-349.

[54] Patton, M.Q., (2001), “Qualitative research & evaluation methods, third edition”, Sage publications, Inc., p. 688.

[55] Crossman, A., (2013), “Statistical software programs for use with qualitative data”, http://sociology.about.com/od/Research-Tools/a/Computer-programs-qualitative-data.htm, 2013.

[56] Du Bois, E., and Horvath, I., (2011), “An easy-to-use methodological approach for considering ubiquitous controllers in energy use optimization”, Proceedings of the EcoDesign, Kyoto, Japan.

[57] Van Petegem, P., (2009), “Praktijkboek: Activerend hoger onderwijs”, Lannoo campus, p. 287.

[58] Bowles, C., and Box, J., (2011), “Undercover user experience design”, New Riders Publishing, Berkeley, CA, p. 191.

Chapter 5

Research cycle 4
Methodology of surrogate-based prototyping

5.1.Introduction

5.1.1. Objective of research cycle 4

In this Research Cycle 4, we investigated the third critical phase in the software development process, which is the system development phase of the software development process, also called detailed development phase, with the objective to increase, support, or refine the stakeholder involvement. In the phase of detailed design, all aspects of the concept are further developed to make the innovation ready for production and commercialization. This phase of system development is characterized by the further detailing and elaboration of the product concept and its validation. To achieve this industrial product, detailed and testable prototypes are necessary. Typically, actions to validate, test and qualify the product are performed, to correct the last design faults and to synchronize the different components and subsystems. Low- and medium-fidelity abstract prototypes were used in the earlier phases and these should be transferred into higher-fidelity testable (tangible) prototypes in this detailed design phase. This raised the need for high-fidelity prototype as a last validation step before fully developed software, which can be achieved fast and at low costs, but which is feasible, detailed, integrative, and facilitating system testing. The growing number of tools, modules and components are becoming available to enable rapid testable prototype development and the necessity to reduce functional or structural modifications at the end of the development process, were the starting points for this research cycle.

5.1.2.Approach of research cycle 4

Referring back to Chapter 1, this research cycle is based on the framing methodology of design-inclusive research, which means that design methods are used to build a testable prototype to validate the theory. The approach of the research is visualized in Figure 5.1. We start this Chapter with the exploration of system design and how it is carried out in both software and product development (Section 5.1.3). Next, we examined the trends in software development (Section 5.2.1.); with a focus that is put on component-based design (Section 5.2.2). Based on this exploration assumptions are made (Section 5.2.3) and a theory for testable tangible prototyping is created (Section 5.3). This theory presents the methodology of surrogate-based prototyping. To validate and verify the methodology, it was applied to the reference case in Section 5.4. In the confirmative research phase, the justification, validation and consolidation were achieved of the methodology (Section 5.6). Section 5.7 concludes this chapter with an overall discussion and conclusions.

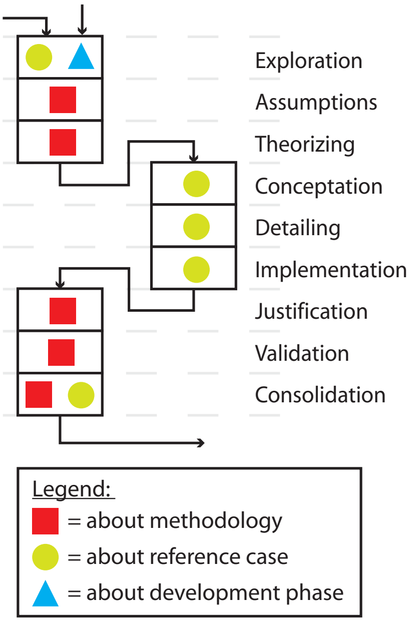


Figure 5.1. Approach of RC4

5.1.3.Explanation on system development or detailed design

The phase of detailed design or system development embraces the development a completely defined product design that is fully documented for manufacturing. This detailed design phase can be defined as [1]: A core engineering process, detailed design transforms concept alternatives, preliminary physical architectures, design specifications, and technical requirements into final, cross-disciplinary design definitions. These designs are further refined and all accompanying documentation required for manufacturing is completed in order for timely delivery to the customer of a fully defined, complete product. We want to clarify the necessity for full system development before code production, because it provides the link for integrating all cross-disciplinary conceptual and preliminary data into a complete, finished digital product definition. Accordingly, today's detailed design process is characterized by highly sophisticated designs and an ever-increasing demand for knowledge sharing.

Along the way, engineers must continually manage change and design complexity. They need to assess risks and balance trade-offs while rapidly delivering high quality designs that work reliably and offer customers value. Hence, software quality which concerns on usability, understandability, learnability, operability, attractiveness and compliance of the software system, should be tested [2] . Balancing changing requirements with cost and quality pressures further complicates matters. Changes to requirements are frequent, and incorporating those changes into the design process in a managed and controlled way is vital. An optimized, formalized, and flexible detailed design process enables companies to rapidly deliver competitive, high quality designs that offer customers real value. Typical benefits of improving the detailed design process may include [1]: (i) improve design productivity (control and management of design data, enable concurrent design of interrelated components, meet key requirements), (ii) increase design process efficiency (enable a formalized, automated, and repeatable design process, improve project execution and visibility into team progress), (iii) optimize design reuse (reduce design cost by supporting part reuse and eliminating component duplication, improve ability to quickly and easily find appropriately classified designs), (iv) improve design collaboration (manage global product development involving external suppliers and customers, provide for secure distributed team and customer design collaboration, encourage early and frequent cross-discipline communication; visualize heterogeneous design data).

We stated that software development is considered to be applying a reflective practice, in which the development is a phased implementation with in each phase having a software prototype, generated with different and increasing functionality level. Consequently, this phase should focus on testable high fidelity prototyping. The previous developed modular abstract prototyping was limited due to its lack of details, and its missing real, active testing possibilities. Consequently, in this phase we had to find a way to develop and execute testable software prototypes before production. The conventional software development methods are also inadequate due to their required knowledge and complexity of programming languages and the appearance of bugs, which increase the time, efforts and costs. The identification of a kind of in-between stage between abstract and fully developed software, needs a prototyping method which can be achieved fast and at low cost, but which is feasible, detailed, integrative, and can be used for system testing. Besides, the methodology must also support the conversion of previous generated and received information (of abstract prototyping) which was given by the different stakeholders to increase the software quality.

5.2.Knowledge aggregation and assumption for testable tangible prototyping

In order to properly evaluate the current situation, we need to be aware of the existing trends in the software industry. For this reason, we focused in the first Section of the explorative literature study on the evaluation of the four most dominant paradigms in software development. In order to be able to narrow the gap between the early abstract prototypes and the publicly tested pre-commercialization prototypes, we formulated the need for testable prototyping. From the perspective of the required prototyping methodology, we investigated the approaches of component-based software prototyping and testing in the second part of the literature study.

5.2.1. Overview of the trends of software development

During the examination the four most dominant software development paradigms (function-oriented development, object-oriented, component-oriented, and service oriented development) [3], we observed a trend of increasing evolvability and increasing complexity of software products [4-6]. To respond to these trends, a need emerged to split the software concept into manageable parts. The concept of separation into concerns is a matter to handle this complexity [7]. Software concerns are combinations of functionalities, which are logical, structural and from user perspective separable. Reacting upon these evolving needs of both flexibility and complexity, an important trend in software development (SD) can be perceived in the shift of the conceptual resources. Several paradigms came up in the last fifty years. The concepts proposed by these programming paradigms try to support developers in the process of improving separation of concerns in a different manner [6, 8-10].

More details on the evolving paradigms can be found in Figure 5.2. We conclude that these paradigms evolved over time to achieve more flexibility and from processing-based to utility-based SD [11]. The most recent paradigms adopt a rather pragmatic approach that believes business system development is an incremental process [12], so changes are inescapable aspects of software design and are expected to occur in every stage . The evolution of programming is tightly coupled with reuse in two important ways: (i) by reusing ever larger grained programming constructs from ones and zeroes to assembly statements, subroutines, modules, classes, frameworks, etc. [13], and (ii) the language is evolved to be closer to human language, more domain focused, and therefore easier to use [14].

For the time being, the majority in industry still uses conventional (function-oriented and object-oriented) software development [15]. However, considering the growing flexibility and complexity that must be dealt with [16], the conventional methods are not ideal. The

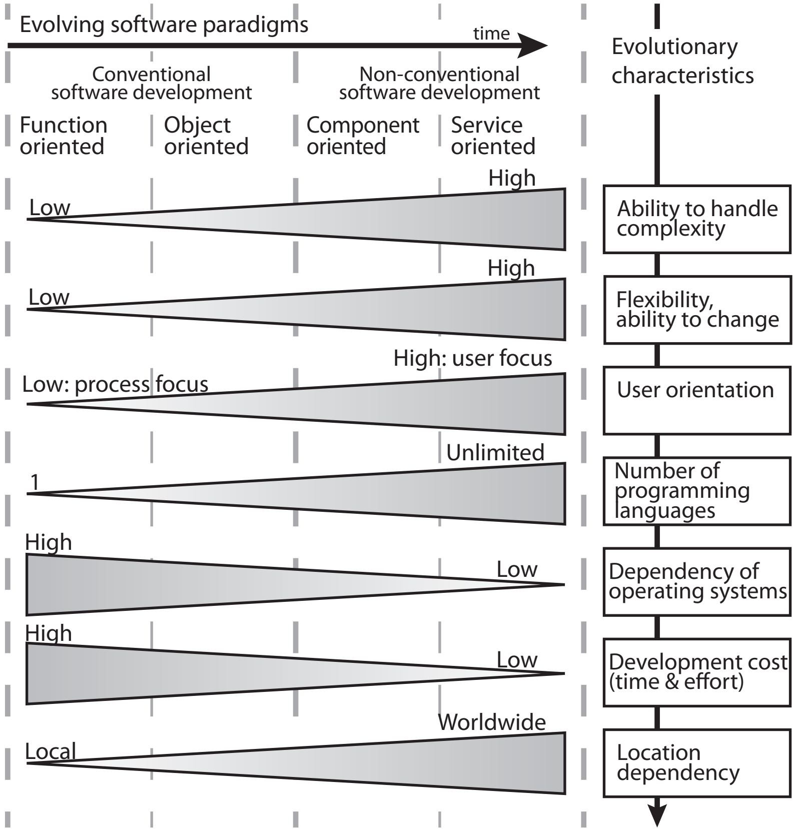

Evolving software paradigms
time
Evolutionary characteristics
Conventional software development
Non-conventional software development
Function oriented
Object oriented
Component oriented
Service oriented
Low
High
Ability to handle complexity
Low
High
Flexibility, ability to change
Low: process focus
High: user focus
User orientation
1
Unlimited
Number of programming languages
High
Low
Dependency of operating systems
High
Low
Development cost (time & effort)
Local
Worldwide
Location dependency

Figure 5.2. *Evolving software paradigms*

conventional software development methods are also inadequate due to their required knowledge and complexity of programming languages and the appearance of bugs, which increase the time, efforts and costs [17]. The identification of a kind of in-between stage between abstract and fully developed software, needs a prototyping method which can be achieved fast and at low cost, but which is feasible, detailed, integrative, and can be used for system testing. Besides, the methodology must also support the conversion of previous generated and received information (of abstract prototyping) which was given by the different stakeholders to increase the software quality.

Consequently, we focus on the non-conventional paradigms (component-based and service-oriented software development) to see how prototyping can be conducted in the present and in the future. We have to mention that we will not focus on service-oriented software development further as it is still in infancy , while component-based development is already more wide spread and growing.

5.2.2. Knowledge aggregation on component-based prototyping and testing

In component-based software development (CB SD), a compositional approach, similar to those realized in the hardware products industry [18, 19], is used instead of generative building. The advantages of this reuse-based development are lower costs, faster delivery & increased quality [16]. An overview of the characteristics of CB SD can be found in Table 5.1.

Table 5.1. ***Characteristics of CB SD***

Approach	use of pre-built components
Target	from large, rigid systems, which are not easily modified to smaller, more portable, independent, and flexible systems
Arguments [4]	reuse, portability, flexibility implicit: it saves time and money, minimize bad builds, and fatal errors, minimize need for key personnel
Component [22]	= piece of executable software with interface commercial, open-source, or in-house developed context independent
Developers	two types of developers: components developers, and applications assemblers
Process [23]	component design and testing component's use by application builders component search, satisfying requirements combining components in a frame interaction building
Auxiliary demands [21]	component interactions interaction rules
Problems in implementation	architectural interface mismatches interoperability incompatibilities [22]
Prototyping opportunities	fast development opportunities existing pre-built components

Although the CB SD is rather similar to manufacturing goods, ensuring the quality of component-based systems is much more difficult than is the case with manufacturing goods [22], as the raw material (software components) may be of uncertain quality and their uses and behavior may be only partially known, hindering the effectiveness of possible quality assessment processes [23]. In CB SD, two parallel software development tracks could be identified; equivalently there are also two separate testing actions for validation and verification: (i) Component testing, and (ii) Application testing. An overview of these testing approaches can be found in Table 5.2.

Table 5.2. Component-based testing

Component testing	comparable to the traditional unit testing [24] largest difference: components can be used by many assemblers for myriad uses in multiple applications [18]
Application testing	
Integration testing	to ensure that components work in unison [18] emphasizes the interface code focused on detecting integration faults [25]
System/ functionality testing	without reference to the code details specification-based testing evaluates both functional behavior and quality requirements [7]
Acceptance / utility testing	by users for validation alpha/beta tests

Discussion and some conclusions

Regarding the necessity for a fully testable high fidelity prototype, we interpreted the findings from the literature study and concluded the following. The most important conclusion is that reusability has been an important concern for industry as research means for software testing. Next to additional advantages such as lower cost and time, there is just no significant advantage from research point to develop software from scratch if similar utilities were readily available in other existing software packages, and can be reused in the new software product. However, there are also some technically lacking issues. The main challenge of compositional SD is interaction between the different components. Often a glue-code is needed between the components to initialize intelligible communication. The principles of non-conventional prototyping help to develop testable prototypes easily, in short time and with low cost. They allow offering a real-life experience for the testers to criticize and improve the functionality and utility of the software in development. Nevertheless, CB SD is only focused on the development of detailed final software products, without considering the opportunities for in-development prototypes, characterized by its limited functionalities and the aim to involve stakeholders in the development process in an earlier phase. In industry, however, there is a need and opportunity for such a software prototyping methodology that is in line with the component-based design approach.

In this research, we focus on the development of second and third generation complex interactive systems, which are characterized by their large functional and structural complexities, self-learning and -reasoning capabilities, partial autonomy, and context-driven adaptability. Regarding these systems, however, we experienced a lack of dedicated prototyping methodologies and means that fit to the characteristics of complex systems and are meaningful in the detailed design phase. Pre-implementation prototyping of such software products and systems is complicated and does not seem to be fully solved by the conventional methodologies. However, there is a major opportunity to use prototyping to avoid the need for functional or structural modifications when production of software programming code is started. However, we concluded that it is assumed that a high-fidelity rapid prototype can be created by a compositional methodology, which: (i) complements the conventional technologies, (ii) enables the investigation of dependability, functional integrity, technical feasibility, accuracy, etc. , and (iii) reduces development time and costs. We extensively surveyed the literature to explore the current state of the art in testable software prototyping. We identified the need for and the possibility of developing a novel prototyping methodology.

5.2.3.Assumptions on testable tangible prototyping

Based on the findings and our conclusions drawn from the exploration, we could formulate following assumptions for the required methodology:

Assumption 1:

The methodology should use the principles of component-based software development as enabler. This reduces the efforts and time needed for original code development in the prototyping phase, while it offers the opportunity for faster functionality and utility testing. Its major objective is to provide a relatively high-fidelity realization of the intended software functionality and support testing.

Assumption 2:

We assume that surrogate software can be used as a means of simulating or prototyping different application parts or concerns and to simulate the function sets of the intended software product. We define surrogate software as existing commercial, in-house, or open source software with certain functionalities that are similar or match function sets of the intended software.

Assumption 3:

Our hypothesis has been that functionally testable software prototypes can be created with purposeful combination of surrogate software. We assume that the advantage of using these surrogates as components is that only a minimal amount of programming is necessary and functionality and usability testing can be conducted earlier. To ensure a working system, these surrogates must communicate with each other. This might bring up a problem of interfacing.

Assumption 4:

In order to be efficient, we state that the methodology should capitalize on simplification possibilities offered by functional and structural similarities, extent of behavioral influence, and abstraction opportunities of sub-systems and components.

Assumption 5:

Considering the availability of software products in the market, we can assume that there are enough software surrogates available on which to base for building the prototype.

Assumption 6:

To handle complexity in software development, literature identified concerns to split the projected software in manageable parts. This can be achieved by functional decomposition into sets of functions. These function sets should be the base to identify the different useful surrogates to prototype the software product.

Assumption 7:

Generally spoken, software can be built in two manners: using a generative approach or by applying compositional construction. Surrogate software can be used directly (compositional) or using the functionalities of the surrogate as a programming language (generative). We use surrogates to realize rapid high fidelity prototyping, because we assume that composition work takes less time than generative software building, since these components only need interfaces to be built.

5.3.Theory and realization of surrogate-based prototyping

Based on the defined assumptions, we developed a methodology that is called surrogate-based prototyping (SBP). In this Section, we deepen the theory of SBP. According to our interpretation, to discuss all aspects of the theory of the proposed SBP methodology, we had to investigate (i) the underpinning theory, which explains the principles of the procedural execution, the method selection, and the criteria and way of testing, and (ii) the implementation aspects, which include the procedural aspects and the methods and techniques. The underpinning theory is explained in the next Sections 5.3.1 and 5.3.2. Below we will explain the other constituents of the methodology, respectively in Sub-Sections 5.3.3, 5.3.4 and 5.3.5.

5.3.1. Theoretical objectives

We discussed all objectives that should be known before formulating the theory of SBP. We found three critical objectives: (i) the trade-off of time optimization, (ii) the balance in creative composition, and (iii) the resolution issue.

Trade-off in time optimization

Focusing on the time-aspect, we must notice that two kinds of time can be identified: (i) time to find the surrogates, and (ii) time to build the prototype. Nonetheless, they are not necessary inversely related, moreover they can be even complementary. This leads to the fact that a high finding time can be together with a high building time, if lots of interfaces are needed. This time constraint is determined by two parameters. The optimal number of surrogates (S) and functions (F) must be found in order to be able to minimize the time:

Opt T (F, S). Consequently, the *minimal* time requests an optimization problem of both the surrogates and the functions of the in-development software.

Another consideration regarding the time is that the total time needed for the development and execution of the SBP must be minimal, because SBP promises to reduce time using a compositional approach compared to other generative prototyping approaches that can be used for functional and utility testing.

Balance in creative composition

Considering the creative composition, it can be argued that the more surrogates we use, the higher the novelty and, obviously, the lower the conventionality of the software. This statement can be explained by the fact that when more surrogates are necessary, the distance to single existing software is higher and consequently the novelty is higher. This novelty can be on content level, context level, structure level, function level etc. The objective is to find the balance between the resolution and the efficiency, because more surrogates means more time and effort to realize the prototype. Hence, we strive after a minimal number of surrogates. As a matter of fact, too high number of surrogates takes use closer to the domain of generative programming, because too much should be adapted and interfaces must be programmed. We note that the number of surrogates will never be 1 in composition development since this would imply an inexistent need for a new software design. As visualized in Figure 5.3, the main goal concerning the surrogates is to find the optimum between the level of novelty and the number of surrogates.

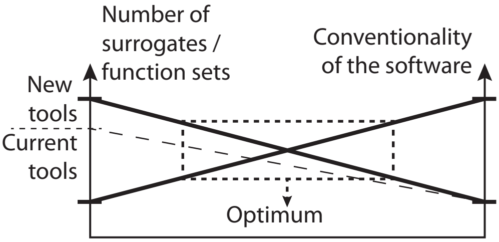

Number of surrogates / function sets
Conventionality of the software
New tools
Current tools
Optimum

Figure 5.3. Scheme of creative composition possibilities

Addressing the resolution issue

Reasoning further on the objective to find the optimal number of functions and surrogates, we can use the analogy of the extremes in combinatorial topologies to explain the resolution issue. These topologies show the number of functional sets in its extremes, varying between low number meaning a rough topology and high number of function sets resulting in a fine topology. In Table 5.3, an overview is given of the advantages and disadvantages of using a rough and fine topology.

Table 5.3. Comparing rough and fine topology

Rough topology	**Fine topology**
Fewer components (+)	More time must be invested to find surrogates for all sets (-)
Fewer interaction (+)	Larger number of interfaces (-)
Hard to find components (-)	Articulated coverage (+)
More extra code generation might be needed (-)	Lower risk of no coverage (+)

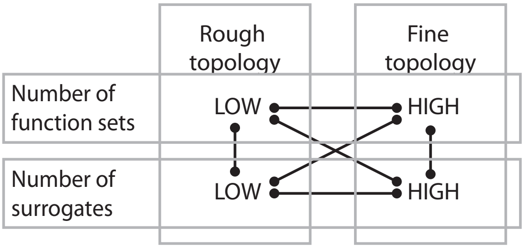


Figure 5.4. Possible combinations of surrogates and function set extremes

A rough topology represents a compositional development, while fine topology represents a generative approach. We assumed that a composite approach is less time intensive than a generative, and as the main goal is to minimize the time, an optimal topology must be found. As shown in Figure 5.4, there are different possible combinations in the number of surrogates and function sets that result in a different topology. To find the optimum, we must recompose the function set in different levels and find the highest possible level to match with the surrogates. If the functionalities are decomposed in different levels of sets and the same is conducted for the surrogates, investigation is needed to combine the functionalities and the affordances of the surrogates in order to find the highest level of compliance.

5.3.2.Underpinning theory

The underpinning theory contains those ideas on which we based to build up the theory of the SBP. To summarize, the main argumentation for using the surrogate-based prototyping methodology is that it enables to develop a high fidelity prototype for functional testing in a minimal time. To achieve this, the theory must answer following questions:

What resolution is needed?

The optimal number of surrogates will be analogous to an intermediary topology, with the aim to go for a minimal number of software surrogates. In order to be efficient, SBP capitalizes on simplification possibilities offered by functional and structural similarities, the extent of behavioral influence, and the abstraction opportunities of sub-systems and components.

What is the basis of surrogate-based prototyping?

Considering the mass of surrogates, we identified two approaches to build the prototype: pure component-based design or platform-enabled component-based design. Considering the platform-enabled design, a platform framework, must be chosen that will serve as basis for the surrogating components, plugins, extensions or modules. The novelty of using a platform-enabled approach compared to pure component-based is that it enables to reduce, or even eliminate, the most important weakness of component-based design, namely the interactions between the different components. Hence, the platform can serve as an underlying surrogate that provides the interactions among the components.

What type of surrogates should be used?

In addition to the chosen basis, two main groups of surrogate software can be identified based on their deployability and affordances: (i) mono-functional software, (ii) multi-functional software, such as Matlab and Visual Studio, which can accomplish multiple functionalities, and software packages or suites such as Microsoft office or Adobe CS, which have a collection of various software programs.

How to deal with the multilevel prototyping?

The feasibility of component-based design depends on two key conditions: composability and compositionality [26]. Composability expresses that component properties are not changing as a result of their interactions with other components within the system. It is a measure of the degree to which components can be assembled in various combinations to satisfy specific user requirements. Compositionality determines if synergic system-level properties can be established by local properties of components. A SBP is compositional if its emergent behavior may be derived from the behavior of its constituent components. Lack of compositionality causes systems that do not behave well outside a small operational envelope.

5.3.3.Procedural aspects

The developed process of how, according to our research activities, surrogate-based prototyping can be conducted, is proposed in this Section. We identified three main phases: (i) identification and selection of the surrogates, (ii) construction or design of the prototype, and finally (iii) the prototype is used for functionality and utility testing. In Figure 5.5., a schematic overview is given of the identified steps man must precede in each phase of the SBP. In this Figure also the used tools and methods were mentioned. More information on the specific tools and methods that need to be used in the process can be found in the next Section.

5.3.4.Methods and techniques

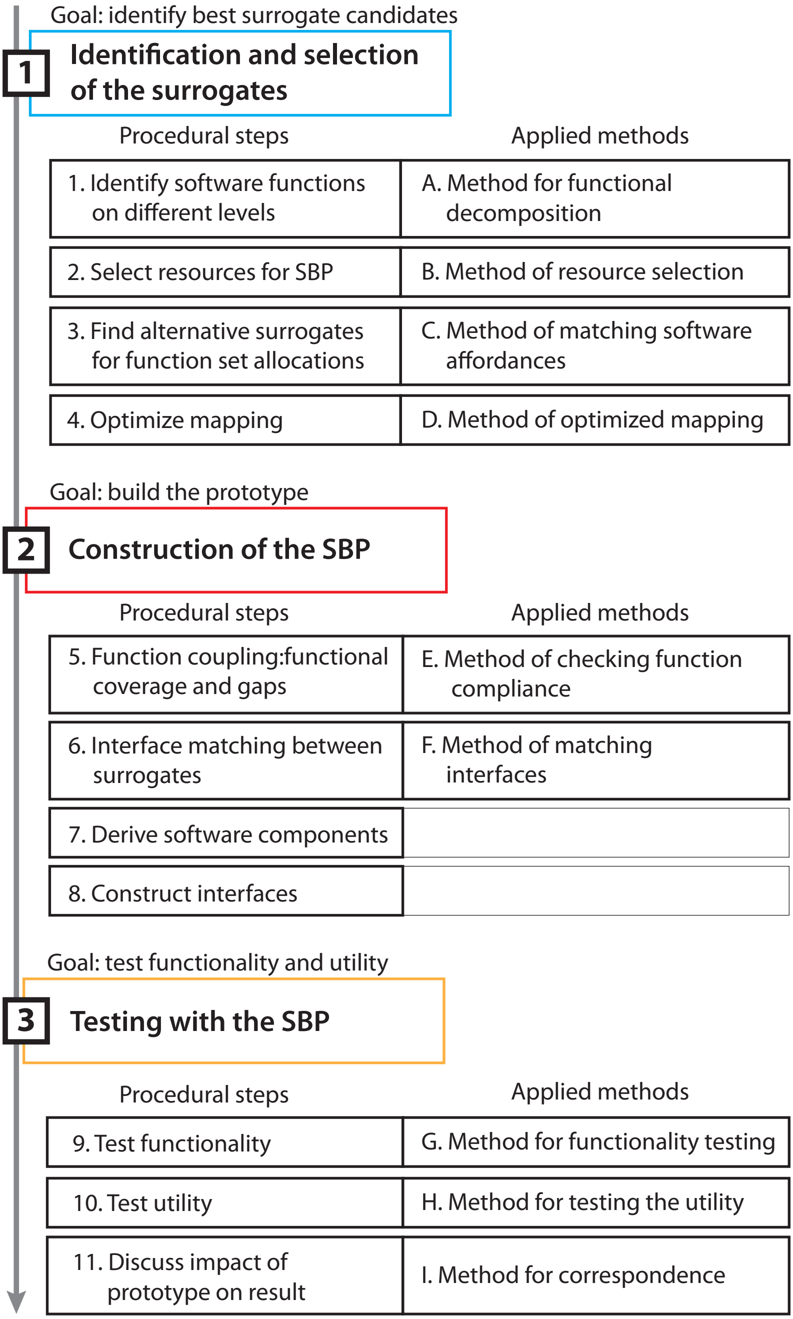
As shown in Figure 5.5., which serves as a transition figure of the procedure to the methods, it can be seen that the process of SBP involves the application of different methods and techniques to support the implementation. In chronological order, methods of: functional decomposition, resource selection, matching the software affordances, optimized mapping, checking the function compliance, matching the interfaces, functionality testing, utility testing, and correspondence validation must be applied.

Method of functional decomposition

The method of functional decomposition is presented in Figure 5.5.A. All software functions should be represented in a functional scheme, which is a hierarchical decomposition structure of all functions on different levels of detail. The identification of the clustering and relations of the functions is important in the next steps to find the relations between the possible surrogates and so their needed interaction.

Method of resource selection

The method of resource selection is presented in Figure 5.5.B. As explained in the



Goal: identify best surrogate candidates

1 Identification and selection of the surrogates

Procedural steps	Applied methods
1. Identify software functions on different levels	A. Method for functional decomposition
2. Select resources for SBP	B. Method of resource selection
3. Find alternative surrogates for function set allocations	C. Method of matching software affordances
4. Optimize mapping	D. Method of optimized mapping

Goal: build the prototype

2 Construction of the SBP

Procedural steps	Applied methods
5. Function coupling:functional coverage and gaps	E. Method of checking function compliance
6. Interface matching between surrogates	F. Method of matching interfaces
7. Derive software components	
8. Construct interfaces	

Goal: test functionality and utility

3 Testing with the SBP

Procedural steps	Applied methods
9. Test functionality	G. Method for functionality testing
10. Test utility	H. Method for testing the utility
11. Discuss impact of prototype on result	I. Method for correspondence

Figure 5.5. Overview of the process of SBP and the applied methods

underpinning theory, there are two approaches to build the SBP: pure component-based design or platform-enabled component-based design. Furthermore, at this stage a decision must be made to focus on mono-functional or on multi-functional software. As there is an incalculable large amount of software that might be used as surrogates, this well-considered limitation would make search for surrogates more efficient.

Method of matching the software affordances

The method of matching the software affordances is presented in Figure 5.5.C. Affordances of surrogate software can be defined as: the perceived and actual functional properties of the surrogate that determine when and for what purpose the surrogate can be used. Software affordances cannot be listed as the affordance proposition is defined by the interplay of the surrogate (functions & implementation) and the context (demands). To find the best surrogate for the different function set, these functions must be matched with the surrogate affordances. For every software surrogate a goodness of matching should be set. Since software programs are constructs of different concerns, multiple surrogates with different affordances are needed to prototype the full system. As the affordances are context dependent, the action of finding the affordances of a surrogate can be seen as a kind of discovery action. In this search for affordances, the functions and the objectives of the intended software should be used as the biggest mental triggers. As visualized in Figure 5.6, the objective of this search is to look for the best proportion, i.e. this surrogate with this operation can achieve this objective of these functions.

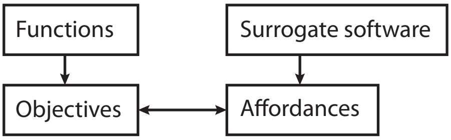


Figure 5.6. Affordance matching

The possible affordances of the surrogates can be found in two manners: (i) by observation: using literature survey and web search, different applications of surrogates can be found. Mapping the affordances of these surrogates in different contexts allows making analogies and interpreting for implementation in the current context. And (ii) by experimentation: trying to implement the function group for which you want to use the surrogate. However this is a very time-consuming activity if the surrogate software does not seem to be the best option. Probably the best technique is to first investigate by observation, and if some surrogates are selected, experimentation can start to verify if the surrogates are useful.

Method of optimized mapping

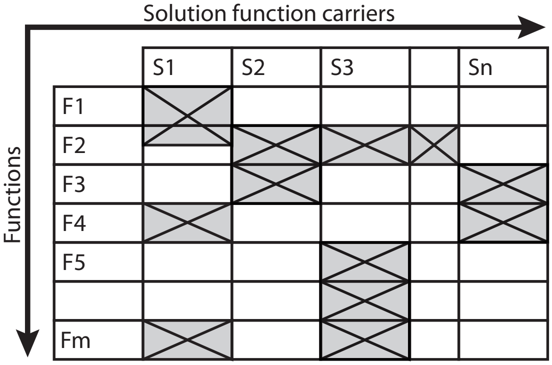
The method of optimized mapping is presented in Figure 5.5.D. Several criteria must be considered to find the optimal surrogate combination of software. As shown in Table 5.4, without order of importance, the decision criteria to identify the most appropriate surrogates for prototyping the software: (i) context of the software, (ii) functional relevance of the surrogate, (iii) composability of the surrogates, (iv) adaptability in the process, and (v) resource dependency. Extra research is needed to decide upon the order of importance.

Table 5.4. Decision criteria for surrogate selection

Decision criteria	Explanation
Context	what, why, how, for whom, where of the to-be-developed software
Functional Relevance	different functions of the software and the interrelated function sets that are made
Compos-ability	flexibility and interfacing of existing software less interaction problems if (i) multifunctional software, (ii) software packages or suites, or (iii) platform-enabled software is used
Adaptability	flow of complementary surrogates in the process mapping of the prototype iterations to determine the flow of complementary surrogates that fulfills the requirements
Resource Dependency [19]	(i) skills of the team, (ii) time and effort to build the prototype, (iii) team's preferences, (iv) longevity of the prototype, (v) fit with the prototype characteristics and foreseen method, and (vi) team's access to the surrogate software

Method of checking the function compliance

The method of checking the function compliance is presented in Figure 5.5.E. For this purpose we have adopted the technique of function compliance, using a matrix as instrument to generate an overview. As shown in Figure 5.7, a matrix scheme of the functions and the possible surrogates should be made. The idea of this representation resembles the traditional morphological matrix, but instead of mapping solutions for the functions, the identified function carriers are shown. Put emphasis on the actions needed to fill in this matrix, since it is an important element in this methodology for developing surrogate-based prototypes.



	S1	S2	S3		Sn
F1	☒				
F2		☒	☒	☒	
F3		☒			☒
F4	☒				☒
F5			☒		
			☒		
Fm	☒		☒		

S = Surrogate ; F = Function set

Figure 5.7. Function Compliance Matrix

Method of matching the interfaces

The method of matching the interfaces is presented in Figure 5.5.F. To find the best combination, the method supports each and every level of interface with a dedicated matching technique. Following levels of interface matching must be considered for each of the selected compositions of surrogates: (i) logical matching, (ii) access matching, (iii) data matching, and (iv) format matching. In Table 5.5, each matching level is explained.

Table 5.5. Levels of software matching

Levels of matching	**Explanation on the levels**
Logical matching	= logical composition of non-overlapping function carriers; three different possible relationships: overlapping, perfect fit, or there might be a gap. The connection of the functions must enhance continuity. so logically ordering of the functions is needed to make this decision.
Access matching	= based on the matching of the interfaces, looking at the data dimension. conversion components can be needed (e.g. if component A uses 3D information while component B can only handle 2D information)
Data matching	= interfacing between the surrogate combinations. data adapter needed for either data conversion or coupling: restructuring, reforming, and re-computing (data representation in complete different manner).
Format matching	= format or representation of the data: fonts, colors … format adapters needed to neutralize the file format and to convert into the requested format

Method of functionality testing

The method of functionality testing is presented in Figure 5.5.G. This method is similar to the existing component-based functionality testing methods. Assuming that during the development of the prototype, component testing and interaction tests were carried out; in this phase the black-box tests should be executed to test the functionality of the software. All system functions and combinations of functions must be tested, using correct and incorrect user input. Different testing techniques might be applied, varying from manual testing to automation. Specific criteria must be derived from the requirements to validate the software functionality.

Method of utility testing

The method of utility testing is presented in Figure 5.5.H. User-acceptance testing is an action that must be performed to further validate the software system. As utility testing should be conducted by potential users, different testing techniques that can be applied are factory acceptance testing, alpha testing and beta testing. In this process, first criteria must be defined and the testing must be planned in detail. Next, tests must be executed and analyzed to get useful results.

Method of correspondence validation

The method of correspondence validation is presented in Figure 5.5.I. The targeted system would ideally be represented through an ideal matching prototype. However, instead we have a real implemented prototype that might be different on some levels. So we have to look how close we are to the objective of the ideal prototype. The concept of the method, applied for the correspondence validation has been called conceptual distance. Having the goal to measure the conceptual distance between the prototype result and the detailed design of the system, we can conclude that if the distance is small, the prototype has a good coverage of the intended software and man can trust that the testing results will be valid for the in-development software as well. On the other hand if the conceptual distance is large, the opposite is true and the validity of the prototype for the specific software product or system must be revised.

5.3.5.Criteria for goodness

To justify the SBP methodology, which means checking its logical correctness, criteria of goodness were identified. In general, this logical correctness can be decomposed into: (i) reliability, (ii) consistency, and (iii) cohesion. Converting these aspects into criteria, we concluded that: reliability of feasibility can be measured if the methodology is executable (this can be expressed by e.g. the rational of the work, the increase of the logic consistency, the shorter prototype development time (compared to generic prototype development), the efficiency of the prototype (again with the generic prototyping as upper ceiling), the level of testability of the prototype, the level of satisfaction of the developers, and the efforts to complete the testing). Consistency can be expressed by checking if there are no conflicts between the methodology components so if the methodology is internally contradiction free (by checking if the chosen surrogates form together an appropriate replica of the functionality of the to-be-developed software). Cohesion can be measured by its friendliness to other theories, if it is facilitating or enabling the implementation of other theories.

5.4.Application of the SBP-methodology to the test case

The SBP methodology has been applied and tested using the reference case, which is aiming at developing a prototype for functional testing of a knowledge-intensive software tool to support product designers in decision making on ubiquitous augmentation of energy-intensive products [28]. SBP was used to improve (optimize) the functionality of the software before implementation. Due to its relative complexity, the proposed software tool is an adequate testing case, and SBP could be applied without constraints or limitations. We continue with the design process, using the conclusions of Section 4.7. The application of each activity step in the concrete SBP process is shortly described in the following sub-Sections.

5.4.1.Identification and selection of the surrogates

Step 1: Functional decomposition

Identifying the software functions was the first step needed for surrogates' selection. As a basis for this an interaction diagram was used, containing a detailed story of how designers' reasoning happens. The software is driven by the designers thinking, and not algorithm oriented. This means that the software is assisting the designer in his thinking process by providing the necessary information and by guiding him to identify a solution. And these operations should happen as the designers want it. The designers' thinking process in interaction with the software tool is visualized in Table 5.6.

Next, based on this detailed story, a functional decomposition hierarchy, as shown in Figure 5.8, was made. We identified a three-level hierarchical decomposition of function sets based on functionalities and on use-relations. In addition to the interaction between the designers' thinking process and the software tool support, we also need to consider the interaction and functions for the other stakeholders, as these functions also have to be inserted into the system. The second important stakeholder to consider is the knowledge engineer. By reasoning on their needed interactions, we also entered the functions of how the knowledge engineers can insert the information into the system.

Step 2: Resource selection

The identification of potential surrogates was a hard task due to the unlimited amount of software products available. Consequently, first we had to select the best resource. Therefore, we started to look for software surrogates in both the pure component-based and the platform-based approach. In the end, we decided to use the platform-based approach, as its main benefit is that it offers the taking care of the composability since the association of the modules are achieved. So no hidden interactions between components will appear because they are managed through the platform. Another decision was made to choose a multi-functional approach, i.e. to choose just one platform for the entire prototype.

Step 3: Affordances matching

In step three in the process of surrogates' selection we identified the different platform tools and their affordances. We limited our search directly to the most popular web application frameworks to make our search manageable.

Finally, Drupal was chosen as platform for this platform-based surrogate software prototyping. In short, Drupal is similar to a Lego kit, for which almost 20000 building blocks - in the form of contributed modules – are available to create an online web application, whether that is a news site, an online store, a social network, blog, wiki, or something else altogether [29]. Having this significant number of modules, they can state that only the really hard 5% needs to be coded from scratch. Drupal could be used in the context of SBP as a platform that enables the surrogating components, called modules, to work together and to facilitate the interaction among them. The use of this platform shortens the development time by enabling better interaction that should not be adjusted manually. It provides the composability of the system.

Step 4: Optimization mapping

Afterwards, all Drupal modules had to be overseen to select the appropriate module surrogates for the identified function sets. All available information on the internet, such as forums, blogs, and you tube tutorials were used to judge the affordances of different modules for the specific case. To simplify the process, possible surrogates were identified and immediately the decision criteria were used to refine the rough selection. Obviously,

Table 5.6. Interaction diagram for designer – software interactions

designers' thinking process	software support
designers starts to think about how to save energy	
before he can he must know what the energy related (direct and indirect) characteristics of his new product are	software shows fields with characteristics to describe. only those who are reflecting the designers view on the case (= semantic interpretation)
design will consider the value of each characteristic	to know the value of some characteristics, software tool offers information of existing products, so designer look into the system and find comparable products for each characteristic.
if most characteristics are known, the designer can go and look for solutions	software supports data visualization of the current case at all time in the process
designer should find possible energy saving solutions	the tool support the search for solutions from three points: search by function, search by principle, and search by technology
	possible solutions are shown in a list, mentioning their most important characteristics
	by clicking on a certain solution , more detailed information is given
designers should reason upon which solutions might be applicable	if a solution is considered to be applicable the designer should click on the add to cart button and the solution is put in a separate box for further use => first decision, which is completely based in designers reasoning
this action should be repeated until all possible solutions are found	! the link of which solution can be applicable for which function should be kept!! => is not in the current situation
next the selected solutions should be reconsidered to see how (and if) they might be inserted to the current case	more information (necessary conditions and constraints from existing cases) is shown about each solution
and the designer should also see how the different solutions for the different functions might work in combination	here the software supports the decision making by: showing the solutions, showing the functions, showing how a solution is applied in other cases
when all possible combinations are made, the designer has to get the overview and see which solution kit has the highest savings and which offer the highest sufficiency	show fields that need to be considered for trade-off calculation
	the trade-off is a pure algorithmic calculation, and gives an overview of the economic best solution without considering which solution has the highest sufficiency.
the designer chose which combination of solutions he wants to use in his product	print out of the possible combinations, and of the trade-off results can be made
designer continues his design process, realizing the ideas on energy saving	

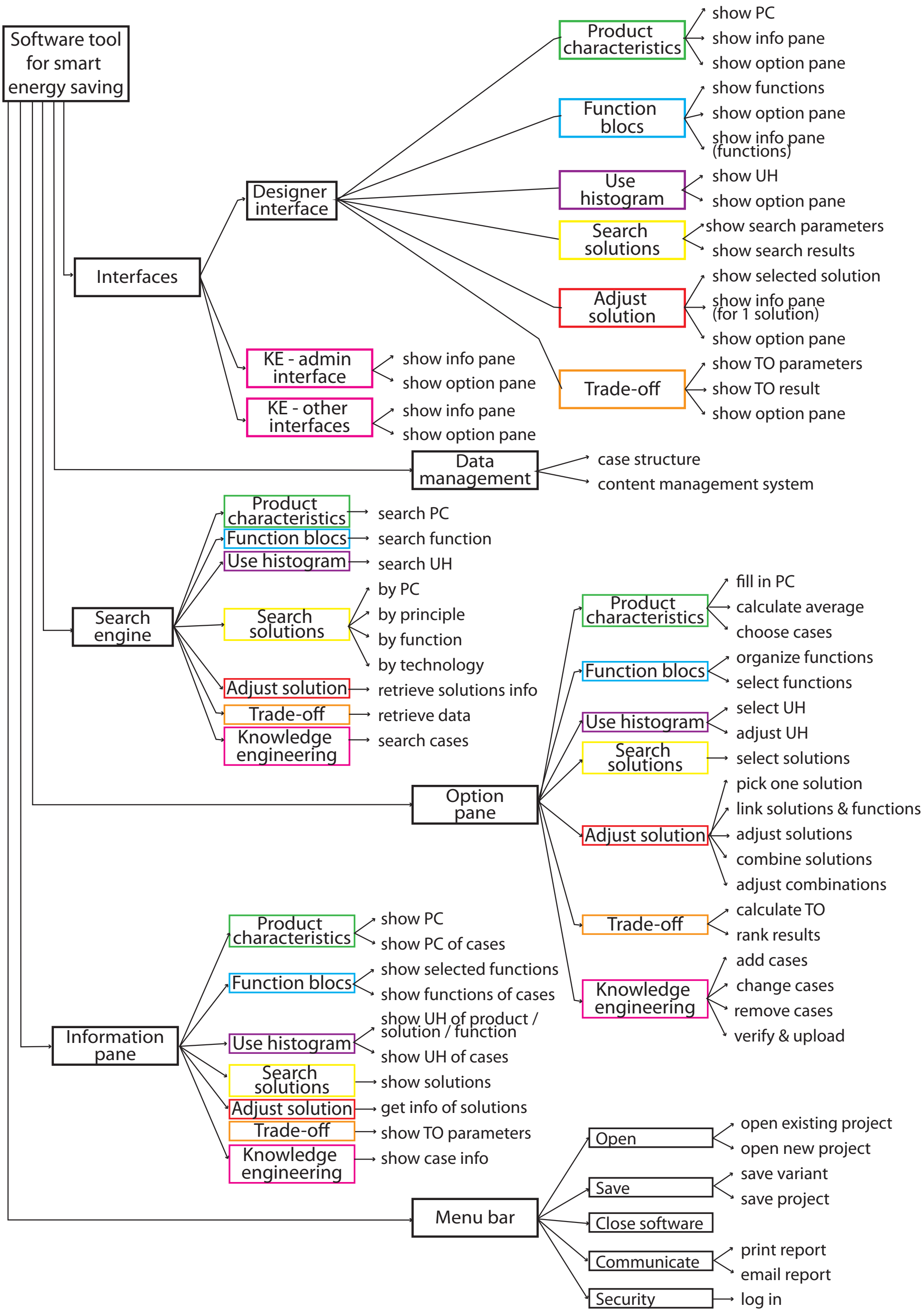


Figure 5.8. Functional decomposition scheme

the functional relevance was the main criteria for surrogate, but also the adaptability and resource dependencies were important in the selection process.

5.4.2.Construction of the SBP

Step 1: Checking the functional compliance

Next, the highest functional coverage with the least amount of gaps had to be found. In Table 5.7, the compliance matrix is shown in which the different possible surrogate modules are matched with the software functions. We must notice that the different modules in the Drupal platform are not related to each other in a hierarchical structure as the function

Table 5.7. Compliance matrix

Main functions	Product characteristics			Search solutions				Adjust solutions		Trade-off		*KE	
Function sets Modules↓	Insert new product case	Open previous case	Save alternative version	Retrieve product case and characteristics	Reuse case characteristics	Search solutions with different keywords	View selected solutions	Combine solutions for current case	Adjust combinations	Retrieve parameters and calculate trade-off	View TO results	Add, remove, edit product cases	Security and accounts
CCK	🟥							🟥				🟥	
Charting	🟥											🟥	
Computed field									🟥	🟥			
Content types	🟥							🟥	🟥			🟥	
Core modules		🟥	🟥	🟥	🟥								🟥
Data search						🟥							
Data taxonomy	🟥					🟥						🟥	
ECK	🟥							🟥	🟥				
Query										🟥			
Rules										🟥			
Ubercart						🟥	🟥	🟥					
Views		🟥	🟥	🟥	🟥	🟥	🟥				🟥	🟥	
Views Calc										🟥	🟥		
Webform								🟥	🟥				

*KE = Knowledge engineering

structure is. The Drupal modules form a holonic system in which different autonomous developed software components (modules) are related to each other as they are required by some and or require other modules. The relationship between the main modules is heterarchical while sub-modules are hierarchically connected to its main modules.

Step 2: Interface matching between surrogates

The holonic structure of the Drupal system creates the interconnected elements that take care of the composability. Consequently, the issue of interface matching will be on all levels covered by the underlying platform. This will save much time as no adapters must be found or coded.

Step 3: Deriving the software components

Following the principle of CBD, we developed the surrogate prototype in a bottom-up fashion. To do so an important step was to convert the functional structure of the software into meaningful data schemes that aim to show the software from a content view. As shown in Figure 5.9, the main item in the different level schemes is the data that is defined, processed and converted in the software product. In the Figure, also the used modules are shown for the different purposes. To give an impression of the surrogate-based prototype, a few screenshots can be seen in Figures 5.10 – 5.13. Figure 5.10 shows a screenshot of how a new product case can be generated at the start of a new project. Figure 5.11 shows a screenshot of how the prototype can be used to search energy saving solutions. Figure 5.12 shows a screenshot that illustrates how the energy saving solutions and the product case can be combined into a new energy saving product kit. Figure 5.12 and 5.13 give an preview of the administrator side of the prototype and focus on content organization (Figure 5.12) and data visualization (figure 5.13).

Step 4: Constructing the interfaces

As explained in Step two, all interfaces were supported by the Drupal platform. So no extra effort was needed to construct extra interfaces between different components.

5.4.3.Testing of the surrogates based prototyping

Step 1: Functionality testing

By choosing the approach of the platform-based SBP, we had the opportunity of using an extra module that executed a part of the functionality test. The Simpletest module of Drupal creates a virtual web browser and uses it to walk through the software in a series of tests, comparable to what we would do if we were doing it by hand [21]. However, because the software was not algorithm oriented, but designers-driven, no useful results were achieved form the automatic tests. Therefor manual tests were executed by the developers. To do so, we took the interaction diagram that was used in the beginning of this phase and went through the system to see if the designers thinking process was indeed efficiently supported by the mentioned software actions and to see if the software reacts according to the designers logic.

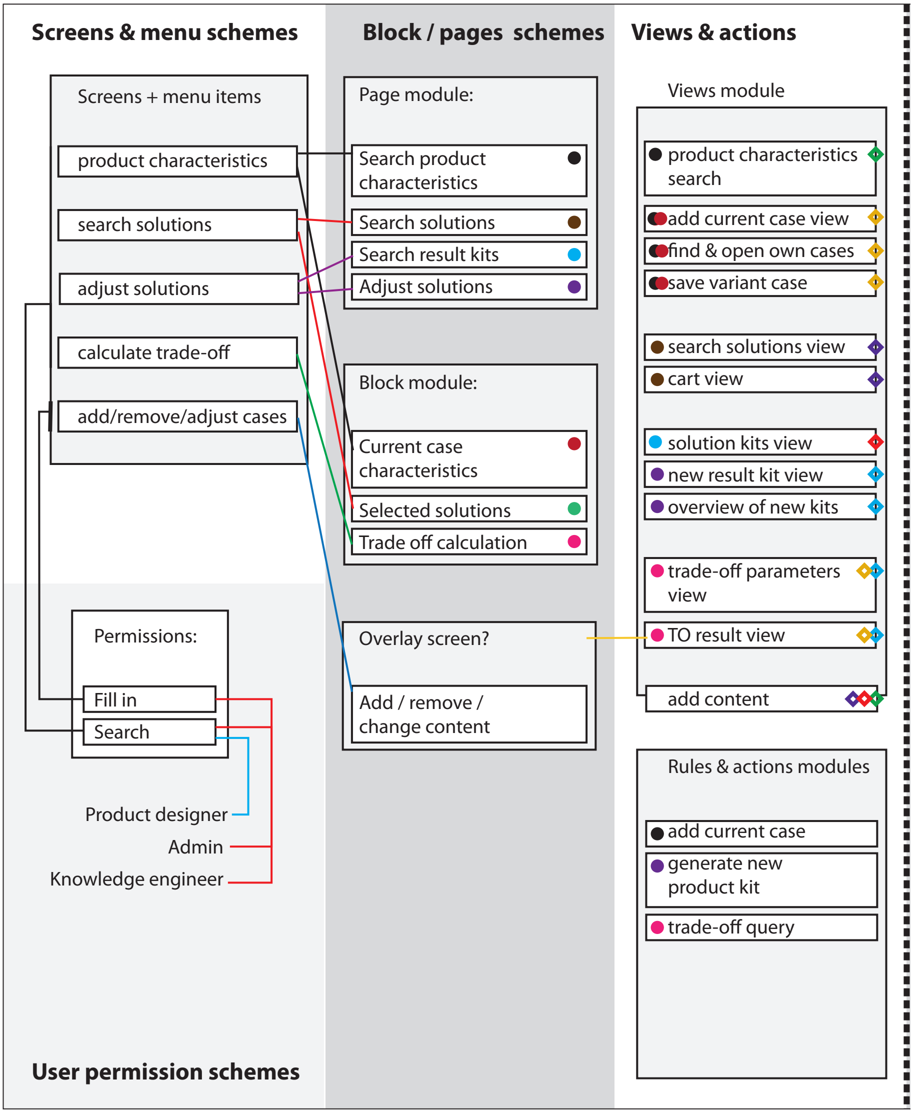


Figure 5.9.   Data level schemes (figure continues on next page)

We concluded that the Drupal prototype successfully executed the intended functionalities with an acceptable performance level. However following adjustments should be made, before finalizing the complete system:

- The available information must be represented in a more visual way (using schemes, visual representations of products in charts, use scenarios, pictures, images…) because

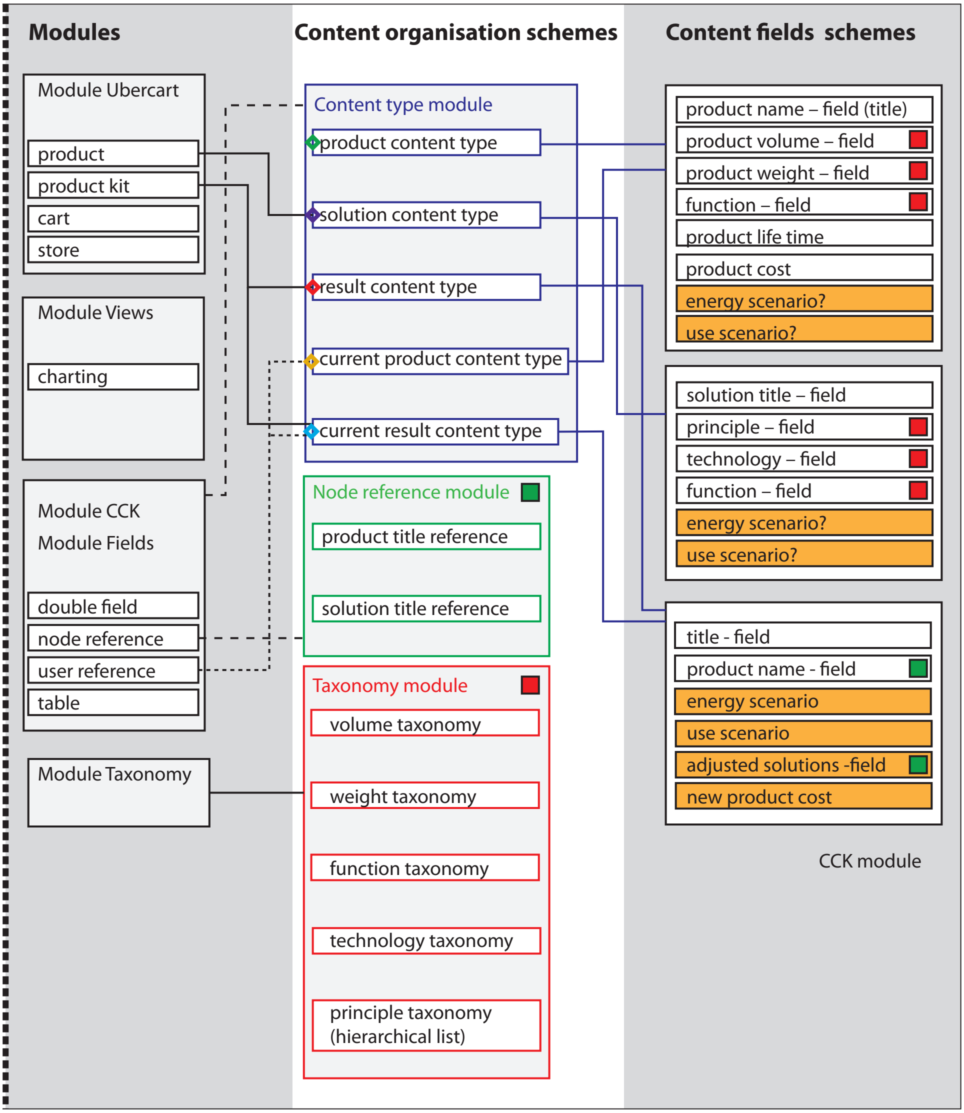
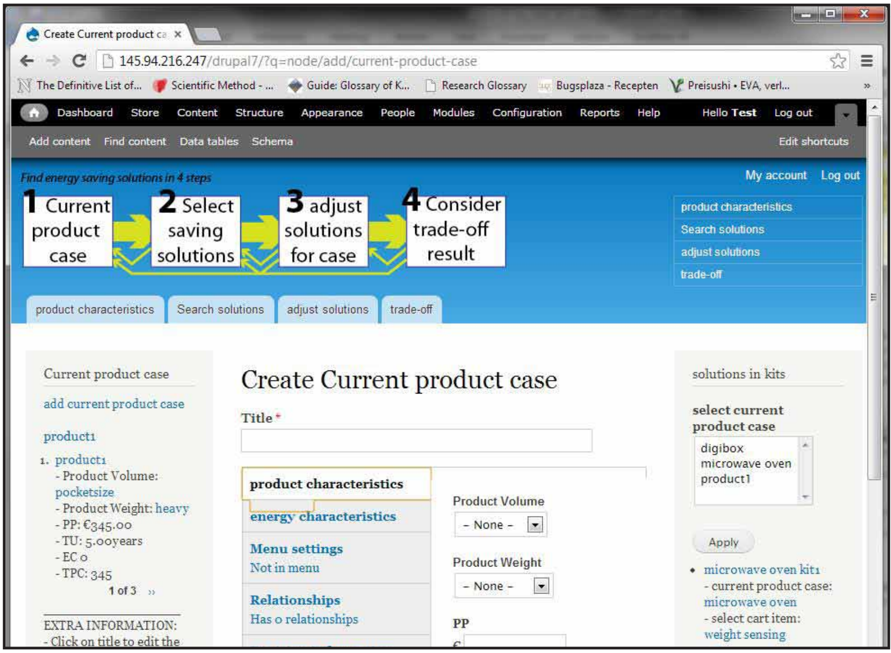


Figure 5.9. *Data level schemes (figure continued from previous page)*

designers' thinking is very visual. Applied in the software, we can improve the graphical aspect in the representation of products, by visualizing the use scenarios on timelines, by visualizing the energy consumption in charts, by showing the link between the functions and the saving solutions, and by graphically showing the effects of applied energy savings.



product1

1. product1
   - Product Volume: pocketsize
   - Product Weight: heavy
   - PP: €345.00
   - TU: 5.00years
   - EC 0
   - TPC: 345

1 of 3 ››

- microwave oven kit1
  - current product case: microwave oven
  - select cart item: weight sensing

Figure 5.10. Screenshot of the Drupal prototype: demonstrating how to create a new product case

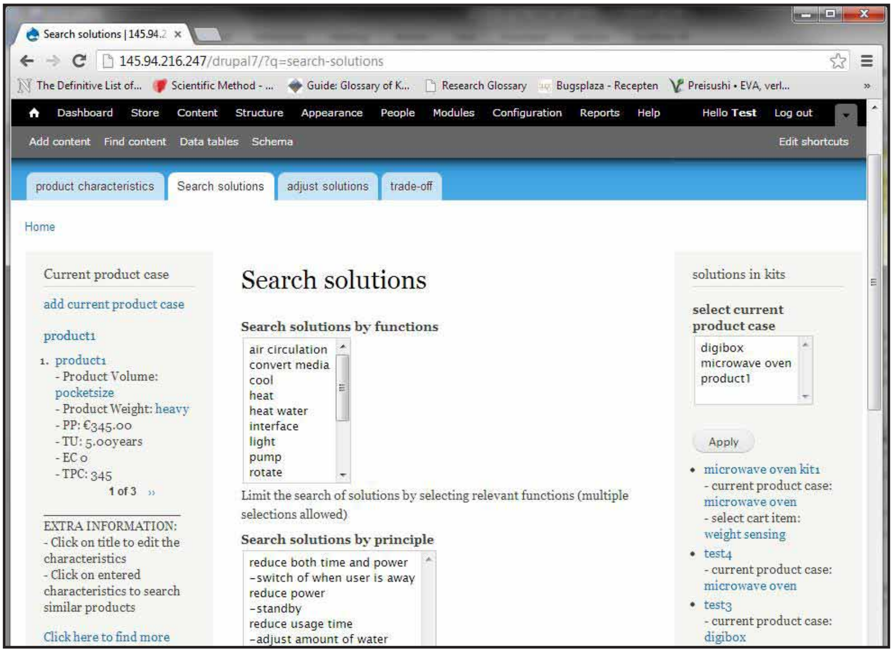
We also discovered that there must be a better differentiation between the necessary conditions and sufficiency constraints, because designers must be informed about what aspects are necessary and what aspects give additional information that might be useful to detect constraints in the new design

- In the step where designers will apply the selected solutions to their case, it is necessary to show the link between the selected solution and the function(s) for which the solution might be applicable. At the moment, this link is lost because all selected solutions will end up in one single list.

- Because products always have major and minor functions, it would be interesting to identify this difference in the software to remind designers about the impact of a saving solution on the usage aspects of the to-be-developed software.

- In addition to the pure algorithmic trade-off results, there should also a possibility to rank the solution kits according to their sufficiency level.

Presently, we only considered the functionality from the perspective of the main stakeholder, which is the product designer. However, additional testing should be carried out to consider the perspective of the other stakeholders as well, especially those of the knowledge



Current product case

add current product case

product1

1. product1
   - Product Volume: pocketsize
   - Product Weight: heavy
   - PP: €345.00
   - TU: 5.00years
   - EC 0
   - TPC: 345

EXTRA INFORMATION:
- Click on title to edit the characteristics
- Click on entered characteristics to search similar products

Click here to find more

Search solutions

Search solutions by functions

air circulation
convert media
cool
heat
heat water
interface
light
pump
rotate

Limit the search of solutions by selecting relevant functions (multiple selections allowed)

Search solutions by principle

reduce both time and power
-switch of when user is away
reduce power
-standby
reduce usage time
-adjust amount of water

solutions in kits

select current product case

digibox
microwave oven
product1

- microwave oven kit1
  - current product case: microwave oven
  - select cart item: weight sensing
- test4
  - current product case: microwave oven
- test3
  - current product case: digibox

Figure 5.11.   Screenshot of the Drupal prototype: search for energy saving solutions

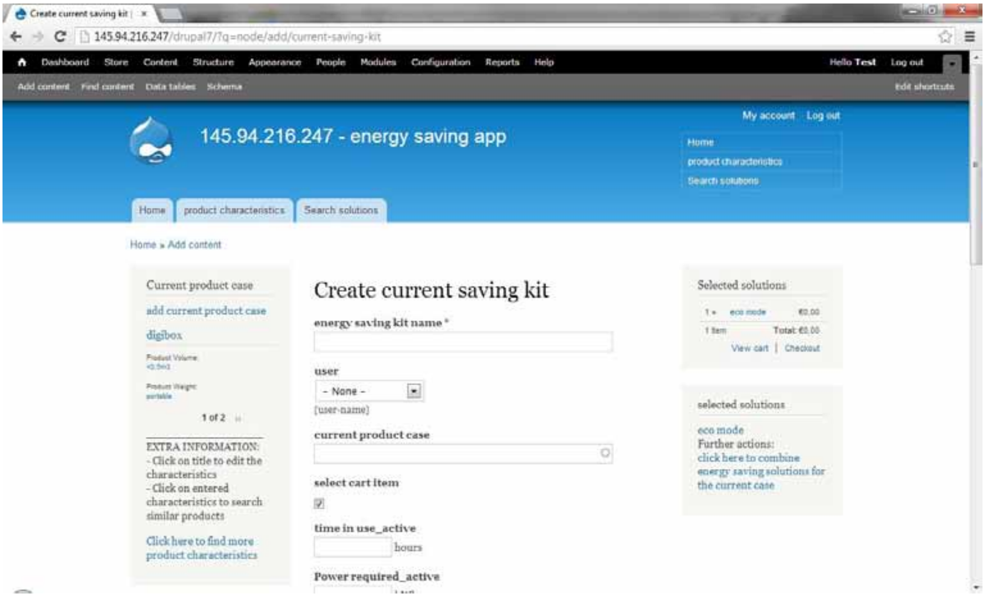
engineers who have to insert their knowledge into the system. Here probably the ontology and terminology that need to be used in order to link and retrieve the data will be issues that need to be solved before going further with the realization of the software tool.

Step 2: Usability testing

In this application case, usability testing was not executed as we did not considered it to be crucial for validating the goodness of the SBP methodology. Usability testing is rather time and resource consuming, and the results of this testing approach are always confounded by the used prototyping tools.

Step 3: Correspondence validation

We measured the conceptual distance between the intended system and the prototyped system. The distance could be considered as low since the intended functions can be executed as prescribed in the specifications. We concluded that the surrogate-based prototype is a good representation of the intended software tool and that it is a meaningful representative to test with on the functionality level. We could conclude this because all functions could be realized in the prototype and could be discussed on a detailed level with stakeholders in order to look for improvements.



Current product case

add current product case

digibox

1 of 2

EXTRA INFORMATION:
- Click on title to edit the characteristics
- Click on entered characteristics to search similar products

Click here to find more product characteristics

Create current saving kit

energy saving kit name *

user

- None -

[user-name]

current product case

select cart item

☑

time in use_active

hours

Power required_active

Selected solutions

1 ×	eco mode	€0.00
1 Item		Total: €0.00

View cart | Checkout

selected solutions

eco mode
Further actions:
click here to combine energy saving solutions for the current case

Figure 5.12. Screenshot of the Drupal prototype: create a new energy saving product kit

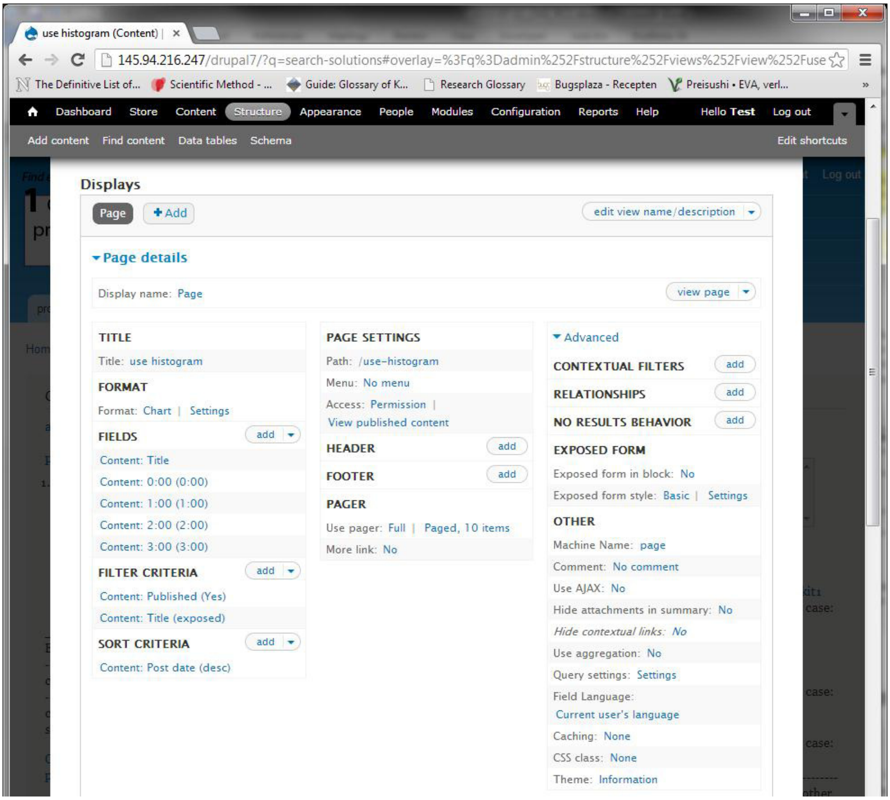
5.5.Confirmative experiments and studies

5.5.1. Explanation on the general conduct of the confirmative research

In the confirmative research experiment, an application case was used to demonstrate and discuss the goodness of the SBP methodology. Empirical testing in concrete application cases is known to be the most effective way of testing methodologies, although it is a reasoning-with-consequences strategy [30, 31]. We applied this strategy, because no other theoretical or non-experimental strategies could be considered for confirmative validation testing. The SBP methodology has been applied and tested in the research project aiming at developing a prototype for functional testing of the reference case. SBP was used to improve (optimize) the functionality of the software before implementation.

5.5.2.Organization of the experiment

Research was organized according to the procedural steps, explained in 5.3.3. In the step of the identification and selection of the surrogates, function sets were identified and resources were selected, so affordance matching could be carried out to find the allocations of function sets and surrogates. Based on this the functional testable prototype was developed which was used in the last step to execute the functional tests. During the execution of the SBP process, self-observation was conducted to be able to reason with the consequences of all actions, considerations, problems and results.



Displays

Page details

Display name: Page

TITLE

Title: use histogram

FORMAT

Format: Chart | Settings

FIELDS

Content: Title

Content: 0:00 (0:00)

Content: 1:00 (1:00)

Content: 2:00 (2:00)

Content: 3:00 (3:00)

FILTER CRITERIA

Content: Published (Yes)

Content: Title (exposed)

SORT CRITERIA

Content: Post date (desc)

PAGE SETTINGS

Path: /use-histogram

Menu: No menu

Access: Permission | View published content

HEADER

FOOTER

PAGER

Use pager: Full | Paged, 10 items

More link: No

Advanced

CONTEXTUAL FILTERS

RELATIONSHIPS

NO RESULTS BEHAVIOR

EXPOSED FORM

Exposed form in block: No

Exposed form style: Basic | Settings

OTHER

Machine Name: page

Comment: No comment

Use AJAX: No

Hide attachments in summary: No

Hide contextual links: No

Use aggregation: No

Query settings: Settings

Field Language: Current user's language

Caching: None

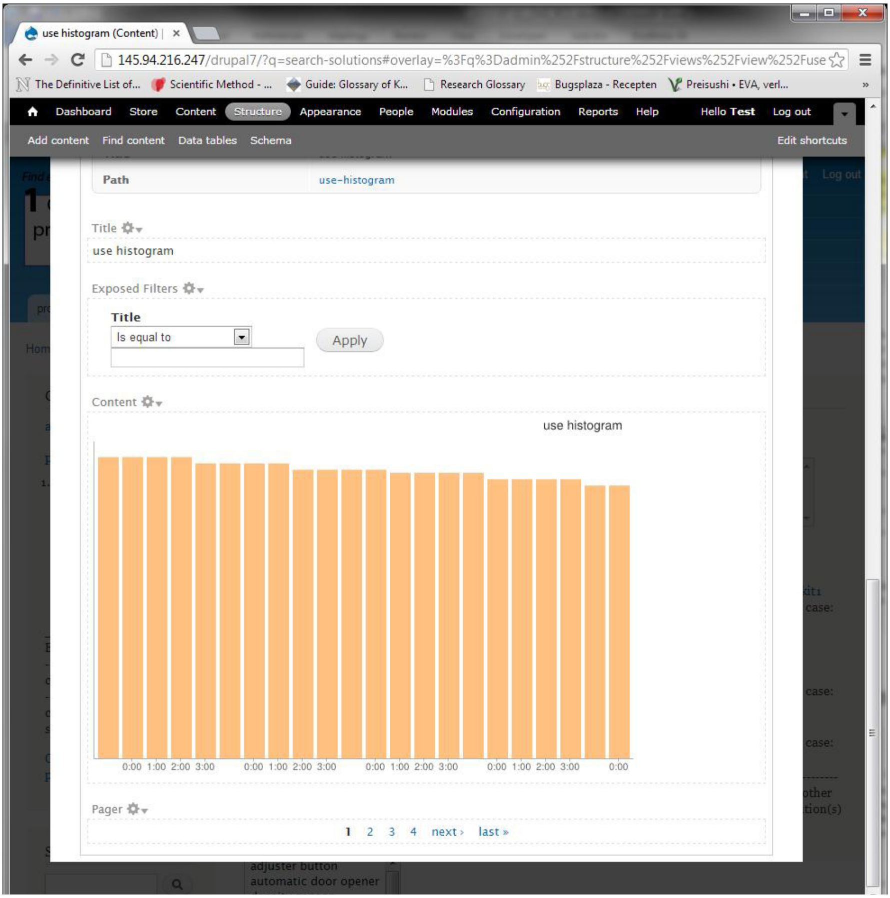
CSS class: None

Theme: Information

Figure 5.13. Screenshot of the Drupal prototype: administrator view on content organization and demonstration

5.5.3.Coding, processing and interpreting data

In this Section, we will further focus on the interpretation of the outcome, process and methods for the SBP methodology. The previously defined criteria for goodness are useful to support this reasoning with consequences process. In general, we were pleased with the efficiency and effect of the software prototype development and testing. The improvements received for the software tool could not be gathered without prototyping these functionalities and the SBP seemed to be a good method to develop a testable tangible prototype in a short time-range. Regarding the validation of the SBP for the application case, we can conclude the following: (i) the case is part of the intended application domain of the SBP methodology, as the case concerned a complex software tool for smart energy saving. (ii) The use of the SBP methodology helped to increase the logic consistency in the process. (iii) It also shortened the prototype development time and costs as the prototype development could be achieved by the existing development team. (iv) The level of testability of the SBP was very realistic and consequently the developers were satisfied. In addition, (v) we could also



Path: use-histogram

Title: use histogram

Exposed Filters

Title: Is equal to

Content

Pager

Figure 5.14. Screenshot of the Drupal prototype: administrator view on data visualization

conclude that the smart energy saving software was a representative full-covering case that could be used to explore the opportunities of SBP.

5.6.Confirmative research concerning the SBP methodology

5.6.1.Justification of the SBP methodology

Indirect justification was chosen for asserting a logical reflection on the developed methodology, using reasoning with consequences strategy. Based on the empirical tests using the reference case, a convincingly experience was built around the applicability of the SBP methodology. Reasoning with the consequences, we were able to scope its properness and to identify the limits of applying the SBP methodology. The result could convincingly be expressed by the execution in the reference case. Reasoning with the consequences of the test result, we could conclude that the SBP was logically error free by discussing its reliability, consistency and cohesion.

Reliability aspect

The methodology was implemented in the development of the reference case, to examine if it was executable. The application of the methodology was completed successfully; here we discuss its most important aspects: However, both negative and positive aspects should be mentioned. Having a very positive validation of the SBP, there were nevertheless also some complications experienced. Herewith an overview of the considerations:

- Need for finding the optimum: how will the Opt (F, S) be known, unless you have started the project, and spent significant time to review what functions (F) and surrogates (S) are available? Further, once the F and S with the required affordances are found, does it make sense to spend time doing the optimization? Will that in fact reduce time?

- Importance/utility of the SBP: Importance of the SBP is in the possibility that all functions can be realized and tested before the final implementation. Because by developing a first testable prototype, the exact tool functions should be detailed and many practical issues appear that must be solved before realization. These functionalities should be considered from the perspective of all stakeholders

- Time issue: we can conclude that a solution must be found to reduce the time needed to discover the most appropriate modules. Limiting our focus to a specific platform was very important. The chosen Drupal system was very interesting for our purpose; however it also had a very high learning curve, which took a lot of time to increase the efficiency. Nevertheless, we also have to mention that due to the use of the platform-based design approach, much time was saved since interactions had not been developed.

Consistency aspect

The SBP methodology was also found internally consistent. Two aspects should be discussed regarding the consistency:

- Difficulty of getting an overview on all possible surrogates: as there are an infinite number of potential surrogates, consequently it is impossible to get an overview of the possible affordances of all surrogates. In contrast to the theory of what characteristics to match it is very hard to select appropriate software, because no-one knows all software that can be used and there is no support engine existing to support in this search. In addition, there is no certainty on how many surrogates we do need to prototype the software. [1]

[1] Some search engines however try to support in this actions: for example the database websites such as www.download.com, www.shareware.com, www.softsearch.com, www.tucows.com try to give an overview of available software, but none of them is (and can be) complete. Moreover, if this is the aim, an important aspect is to standardize the terminology and to find the best keywords. Similar problems emerged in CB SD, as shown by [32], when selecting a component, many solutions have to be considered. This is easier to achieve if all needed information is available

- Content view instead of functional or procedural: One of the important considerations from the case development is that during the development, we shifted from a procedural, functional view to a content view of the software implementation. In the previous phases of software development, a physical structure of the software is used, focusing on functionalities, structures, and information flows, considering how the software should fulfill its objective. During the development of the SBP, we needed to consider the same software from the content view, considering meaning and data schemes as structures of data and their relationships. Here the question is more related to what data is transferred, communicated and processed through the system.

Cohesion aspect

Cohesion can be measured by checking the facilitation of one theory to other theories. Here, we have the opportunity to check the cohesion between the methodology and the SBP methodology. We concluded that using the MAP in the concept integration phase, all required information was available to start go further with the SBP in the system development phase. This necessary information comprehends a validated concept of the entire software with a detailed overview of its different functionalities.

5.6.2.Validation of the SBP methodology

In this Sub-Section, the internal validity of the experiment was discussed. Validation may focus on multiple aspects; however we decided that construct validation and content validation were the most appropriate ones here. The method used for validating the methodology was logical reasoning on the aspects that delivered the solution.

Construct validation

The first aspect is the construct validation: As it is important to validate if what had to be measured was truly measured. Therefore, we investigated the different constructs or elements that were used during the operationalization of the SBP. In the SBP methodology, we could identify following main constructs: (i) the functional sets, (ii) the surrogates selection, (iii) the surrogate-based prototype, (iv) the test executions, (v) the data evaluation, and (vi) the adjusted system. The methodology of SBP was developed to increase stakeholder involvement in the system development phase to test the functionality and usability. In the process, a surrogate-based prototype was built to simulate all function sets using a selection of surrogates. This prototype was use to execute functionality tests. We can conclude that the application experiment was a valid approach as we could observe how the desired

in one place. Web-based component portals such as www.eCots.org, www.SourceForge.net, www.ComponentSource.com, and www.Flashline.com attempt to provide this functionality. Component catalogues, supplied by portals, contain information about a range of vendor solutions described in a relatively uniform way (in some circumstances open source options as well). These catalogues normally rely on ontologies and domain hierarchies to function. In comparison with CB SD, where standardization of information that must be communicated helps components specifications, it would be needed in the case of SBP to know the software specifications.

effect was achieved and how the different constructs of the methodology were needed to converge into a validated system design.

Content validation

In the content validation, we measured the extent to which a measure represents all facets of the SBP methodology. The measure used in this research was the applicability of the SBP methodology in the reference case. The methodology of SBP was developed to increase stakeholder involvement in the system development phase, using a testable prototype of the software, functionality and usability testing can be achieved to improve the system design. Using the reference case, an experiment was set up to test if the SBP indeed supports achieving the functionality test. We can conclude that the experiment was a valid approach as we could observe how the methodology was used to achieve the testable prototype and that the prototype could be used for testing. The selected reference case was valid to measure the applicability of the methodology, as it belongs to the operation domain.

5.6.3.Consolidation of the SBP methodology

Consolidation has two aspects to discuss, the de-contextualization and the re-contextualization. De-contextualization or generalization is not considered to be relevant for the SBP methodology. We could argue that the SBP can also be used for other software development approaches, where complexity and time issues are opposing the need for fast functionality testing. The principle of SBP is already used in the development of physical products to test the functionality and working principles and mechanisms, so it should also be considered in the development of product-service combination and in system design. However, more research is essential on the coupling of physical and cyber prototyping. As we do not want to use the SBP out of the context of the complete DSDM, we do not consider it to be important here.

The re-contextualization or specialization is more important here, regarding the information that is transferred to the next step. We have to consider how the information that comes out of this phase will be used in the further realization of the software, namely if the current available design knowledge is useful for the realization of the software. We consider if this re-contextualization for software production is supported by the SBP methodology, and if after this methodology, the design of the software is conducted on all levels (abstract to practical).

5.7.Concluding remarks

Proposition 1:

> Surrogate-based prototyping (SBP) allows exploring the functional discrepancies exploration of the software since it supports the testing of the operations (functional realization, robustness and computational performance) of the software in a midterm phase of software development.

Proposition 2:

An SBP is successful if a high level of composability and compositionality is achieved. To achieve the matter, affordance matching of the surrogates with the functional sets must be conducted, and the latter can be achieved through the establishment of the interfaces between the tools.

Proposition 3:

The SBP goes beyond the conventional concept of pure component-based design and avoids the problem of interfacing of heterogeneous components, using a platform-enabled approach

Proposition 4:

The surrogates combinations should be customized to replicate the functionality required by the final software, consequently hi-fidelity predictions can be made.

Proposition 5:

The success of SBP was amplified by the necessary conversion of the software view. A logically organization of the component-based flow of the system is needed to detail the functions sets and to find the best surrogates. However, for the construction of the SBP a content view was needed to show the data management using different schemes of levels.

Proposition 6:

SBP allows reasoning about the stakeholder-computer interaction on such a detailed level that all functions and usability aspects can be reconsidered for improvement.

Proposition 7:

The importance of the SBP is in the possibility that all functions can be realized and tested before the final implementation. Because by developing a first testable prototype, the exact tool functions should be detailed and many practical issues emerge that must be solved before realization.

Future work should be performed on two aspects. Firstly, an investigation must be carried out on how to efficiently optimize the design of the component-based surrogate prototype. Examination on how optimization can be achieved in Drupal to reduce the complexity, by decreasing the number of functions, the number of modules, and or the number of interfaces must be conducted. In addition, the relationship between these aspects must be defined, as it is not just linear. Secondly, future work is also needed to achieve the development of a continuous evolving overview of the available surrogate software possibilities.

5.8.References

[1] Parametric Technology Corporation, P., (2006), “Detailed design - developing a completely defined product design that is fully documented for manufacturing”, 2071-DetailDesign-

TS-1206, p. 4.

[2] Majid, R.A., Noor, N.L.M., Adnan, W.A.W., and Mansor, S., (2010), “A survey on user involvement in software development life cycle from practitioner’s perspectives”, Proceedings of the Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, pp. 240-243.

[3] Crnkovic, I., (2013), “Introduction to component-based software engineering”, Mälardalen University, 2013.

[4] Sharp, j.H., and Ryan, S.D., (2010), “A theoretical framework of component-based software development phases”, the DATA BASE for Advances in Information Systems, Vol. 41 (1), p. 20.

[5] Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A., (2004), “Mining version histories to guide software changes.”, Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, Washington, DC,, p. 9.

[6] Elfatatry, A., (2007), “Dealing with change: Components versus services”, Communications of the ACM, Vol. 50 (8), p. 5.

[7] Sommerville, I., (2011), “Software engineering”, 9 ed., Pearson Education Inc.

[8] Tarr, P., Ossher, H., Harrison, W., and Stanley M. Sutton, J., (1999), “Degrees of separation: Multi-dimensional separation of concerns”, Proceedings of the Proceedings of the 21st international conference on Software engineering, ACM, Los Angeles, California, United States, pp. 107-119.

[9] Ossher, H., and Tarr, P., (2001), “Using multidimensional separation of concerns to (re)shape evolving software”, Commun. ACM, Vol. 44 (10), pp. 43-50.

[10] Papazoglou, M.P., Traverso, P., Dustdar, S., and Leyman, F., (2007), “Service-oriented computing: State of the art and research challenges”, IEEE Computer Society, Vol. 40 (11), pp. 38-45.

[11] Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., and Munro, M., (2000), “Service-based software: The future for flexible software”, Proceedings of the Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, IEEE, pp. 214-221.

[12] Fleischmann, A., and Stary, C., (2012), “Whom to talk to? A stakeholder perspective on business process development”, Universal Access in the Information Society, Vol. 11 (2), pp. 125-150.

[13] Heineman, G.T., and Councill, W.T., (2001), “Component-based software engineering: Putting the pieces together”, Addison-Wesley Reading.

[14] Frakes, W.B., and Kang, K., (2005), “Software reuse research: Status and future”, IEEE Trans. Softw. Eng., Vol. 31 (7), pp. 529-536.

[15] Vyatkin, V., (2013), “Software engineering in industrial automation: State of the art review”.

[16] Khan, A.I., Alam, M.M., and Khan, U.A., “Validation of component based software development model using formal b-method”.

[17] Singh, C.P., (20013), “Presentation on component-based software engineering”, South Asian University, New Delhi, India, 20013.

[18] Vitharana, P., (2003), “Risks and challenges of component-based software development”, Communications of the ACM, Vol. 46 (8), p. 6.

[19] Brown, A.W., (1998), “From component infrastructure to component-based development”, Proceedings of the international workshop on component-based software engineering, Sterling Software, p. 4.

[20] Keil, R., (2012), “Component-based software development for cortex-m microcontrollers”, ARM, the architecture for the digital World, 2012.

[21] Cechich, A., Piattini, M., and Vallecillo, A., (2003), “Assessing component-based systems component-based software quality”, Cechich, A., Piattini, M., Vallecillo, A. (Eds.), Vol. 2693, Springer Berlin / Heidelberg, pp. 1-20.

[22] Gomez, J.M., Alor-Hernandez, G., Posada-Gomez, R., Rivera, I., Mencke, M., Chamizo, J., Sanchez, F.G., and Toma, I., (2008), “An approach for component-based software composition”, Proceedings of the Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA ‘08, pp. 195-200.

[23] Gao, J., Tsao, H.S.J., and Wu, Y., (2003), “Testing and quality assurance for component-based software”, Artech House, p. 468.

[24] Nirpal, P.B., and Kale, K.V., (2011), “An overview of software testing methodology”, International journal of knowledge engineering, Vol. 2 (1), p. 6.

[25] Wu, Y., Pan, D., and Chen, M.-H., (2001), “Techniques for testing component-based software”, Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems, IEEE, Skovde , Sweden, pp. 222-232.

[26] horváth, I., (2012), “Beyond advanced mechatronics: New design challenges of social-cyber-physical systems”, Proceedings of the ACCM-Workshop on „Mechatronic Design“, Linz, Austria, p. 20.

[27] Arnowitz, J., and Berger, N., (2010), “Effective prototyping for software makers”, Morgan Kaufmann Publishers, p. 584.

[28] Du Bois, E., and Horvath, I., (2011), “An easy-to-use methodological approach for considering ubiquitous controllers in energy use optimization”, Proceedings of the EcoDesign, Kyoto, Japan.

[29] The Drupal Association, (2012), “Drupal modules”, http://drupal.org/, 2012.

[30] Jensen, S.Ø., (1995), “Validation of building energy simulation programs: A methodology”, Energy and Buildings, Vol. 22 (2), pp. 133-144.

[31] Kleijnen, J.P.C., (1995), “Verification and validation of simulation models”, European Journal of Operational Research, Vol. 82 (1), pp. 145-162.

[32] Sjachyn, M., and Beus-Dukic, L., (2006), “Semantic component selection – semacs”, Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS 2006), p. 7.

Chapter 6

Research cycle 5

Assessment of the designerly software development methodology

6.1. Introduction

6.1.1. Objective of research cycle 5

The validation of design methods is important for the continuing advancement of both design theory and the professional practice of engineering. Researchers in design theory proposed going through validation processes to guide the development and evaluation of new methods. Professional practitioners need validation processes to determine which methods to employ, and when and how to employ them. Validation of methodologies can be not based on mathematical modeling but on somewhat subjective evaluations [1]. A methodology typically operationalizes human knowledge that also contributes to the subjective nature [2]. In the last phase of the promotion research, we had to validate the proposed designerly software development methodology in a qualitative as well as in a quantitative manner. Actually, our *objective* was to check the external validity of the software design methodology. For the sake of completeness we note that internal validation of the work and findings has also been made, but it was done in the confirmative parts of each research cycle.

Because different definitions are available, we rely on the definition that external validation is the extent to which the results of a study are generalizable or transferable [3-5]. To explain what this definition mean in our particular context, we revisit the current state of the research, until now we only discussed the internal validation of the single phase methodologies. The two differences compared to previous chapters is that we consider the validation not on the single phase methodology level, but on the level of the DSDM, and we shift from internal validation to external validation to complement the complete validation. We assume that the internal validation carried out for the single phase methodologies stays valid in the total methodology. The external validation could only be performed at the end of the process when the findings are known. Considering the validation in the context of the DSDM, we found that the external validation was most efficient using a reflective validation approach. Comparison was not possible as in the study only a single case was developed. Moreover, executing an additional comparative validation would have raised the need for an extra research cycle in which a comparison could have been made by a simultaneous development of a specific software product using on the DSDM and a traditional software development method.

6.1.2. Framing of the research approach

Research cycle 5, which is discussed in this chapter, is an operational research cycle that is focusing on the assessment of the external validation of the DSDM. The approach of the research cycle is shown in Figure 6.1. To understand how to do this validation, we dived deeper into the literature to find suitable validation methods for validation our methodology (Section 6.2). In short, we could not find a specific generic external validation method for this context in the literature. However, we could derive one method called the validation square that seemed to be a generic external validation method, which we could use directly or after adaptation. In Section 6.3, the theoretical and methodological fundamentals of this validation method were discussed, plus the adjustments and extensions, which were required for our context. In addition, the operationalization of this quadrant-based method is detailed for the specific purpose (Section 6.4). In Section 6.5, the execution of the assessment is discussed. Then, Section 6.6 reports on the findings of the execution of the validation assessment. Finally, in Section 6.7 some concluding propositions are formulated.

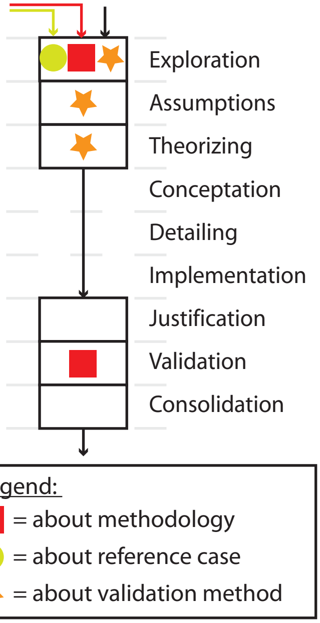


Figure 6.1. Approach RC5

6.1.3. On validation approaches

In Figure 6.2, which shows a general overview, we identified two validation means: (i) direct validation, and (ii) indirect validation, and two approaches for the validation of a methodology: (i) reflective approach and (ii) a comparative approach. Direct validation is performed using the methodology while indirect validation is achieved by reasoning with the consequences. Here, the methodology is evaluated based on its impact on process change, people's satisfaction, process characteristics, behavior aspects, resources, etc. The

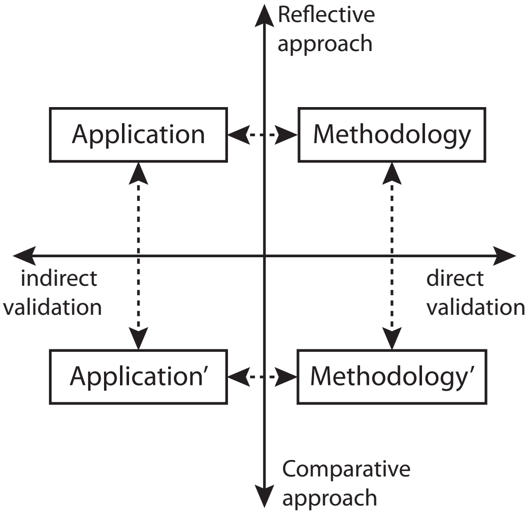


Figure 6.2. Methodology validation possibilities

other difference was made between a reflective approach and a comparative approach. Reflective approach is focusing on the theoretical opportunities and limitations of the methodology while in a comparative validation approach; two or more methodologies are compared.

6.2. Overview of methods for validating design engineering methodologies

To achieve an overview of the available methods for validating design engineering methodologies, a literature study was executed, focusing on two aspects: (i) existing validation methods that might be applied for external validation in our context, and (ii) possible validation criteria.

6.2.1. Findings about external validation methods for software development methodologies

Validation depends on the purpose of the methodology and its intended use [6], so considering the context of the validation while selecting the appropriate validation method is most important. In this part of the review, we analyzed existing generic external validation methods in the context of validating software development methodologies. One of the challenging research problems in validating a software engineering methodology (SEM) is dealing with the complexity that emerged because the SEM involves the use of human knowledge in its phases. To measure such knowledge, Lee and Rine [7] use case study research design, which is an empirical research alternative in designing a research plan that establishes a logical link from the data to be collected to the initial questions of study. For an effective research case study, they say that it is necessary for the validation exercise to first have designed a case study specific to the characteristics of this invented SEM. On the other hand [8] advises to use surveys to gather empirical data for the validation of methodologies. According to Kitchenham et al. [9], the most important methods for software methodology validation are: formal experiments, quantitative case studies and feature analysis validation. Briand et al. [10] do not only consider the empirical validation but also the theoretical validation of methodologies. Similarly, Schön and Argyris [11] proposed a framework for evaluating methodologies that included checks on: (i) internal consistency, (ii) congruence with the espoused theory, (iii) testability of the theory, and, ultimately, (iv) effectiveness of the theory.

6.2.2. Generic methods for external validation of methodologies

Since there was no worthily specific methodology that is developed for validation in our context, related contexts were identified and applied validation methods in these contexts were discussed. A relation was found with software validation, design knowledge validation and model validation domains, and we also considered the domain of research methodology validation. Software validation is important in the development process to consider the user's point of view, so different methods are used in each stage of the software development process [12]. Most validation methods are using different kind of prototyping [13] that serve as the source of requirements and enhance the developers' understanding of the system objectives and the users' expectations, as well as the system functionalities. Nevertheless, validation of the prototype is also crucial [14].

In the domain of design knowledge validation, different approaches could be found, we base on the literature review carried out by [15]. One framework suggested by [16] emphasizes the fit between problem-solving behaviors and the problem environment, rather than the internal consistency of the behaviors. The framework of Schön and Argyris [11] for validating theories can also be used related to professional practice. A similar framework was proposed by Pedersen et al. [17] in which they suggest a balanced approach that includes the evaluation of internal consistency and effectiveness. In the framework, design theories are validated according to the principles of a validation square consisting of four quadrants: (i) theoretical structural validity, (ii) empirical structural validity, (iii) empirical performance validity, and (iv) theoretical performance validity. Frey and Dym [15] conclude from a comparison of design and medicine methodologies to use simulation models in validation where possible, since this technique has proven its quality in the medicine domain. To increase the confidence in a simulation model several well documented and comprehensive validation methods should be used combining several validation techniques. Landry et al. [18] and Sargent [19] defined five types of validity related to the modeling process: (i) conceptual, (ii) logical, (iii) experimental, (iv) operational, and (v) data validation. As discussed by [20, 21], many people consider that empirical validation is a more powerful approach to validation. Empirical validation should in principle compare a 'true' model, based on measurements obtained from physical experiments, with simulated results from a mathematical model implemented in a program. Nevertheless, in case of designing a new system, comparison with a true model is not possible, so [22] compares the implemented model behavior with its assumptions and specifications.

Lastly to discuss research methodology validation, we can base on Dellinger's [23] overview of the quantitative and qualitative research validation approaches. Since qualitative research validation seemed to be most related, we summarized his review. Over the past few decades, many researchers have participated in these discussions. Lincoln and Guba [24] suggested the need to develop an entirely different approach to assess validity than what are traditionally used by quantitative researchers. These theorists developed the concepts of trustworthiness, which corresponds with Campbell and Stanley's [25] concepts of internal and external validity. Eisner [26] took this one step further by not using the word validity but instead used the word credibility. Maxwell [27] identified five types of validity: descriptive validity, interpretive validity, theoretical validity, generalizability, and evaluative validity. Eisenhart and Howe [28] advocated for a collective validation construct in which general standards for conducting qualitative research should be used as guidelines.

6.2.3. Findings about validation criteria

In addition to a technique, method or framework to execute the validation, there should also be some criteria by which the proposed design methodologies are judged to ensure that their use will consistently yield the correct design, i.e., that these methods are valid. We concluded from the literature study that also the validation criteria are context dependent. Here an overview is given of criteria from the context of software applications, (software) design methodology, model development and qualitative research validation. According to [14, 29], software applications are validated using the following criteria: correctness, consistency, sufficiency, performance, necessity, level of expertise, builders/users' risk, maintaining objectivity, and reliability. [9, 30] described the validation of design methodologies by the following criteria: (i) basic: it must be logical, complete, understandable, usable, internally consistent etc. (ii) use: it must be helpful, produce the specified, usable and relevant results, use meaningful reliable information, not bias the designer; (iii) gain: it must provide added value.

According to [1, 21, 31], the two most important criteria for model validation are model accreditation (model satisfies criteria) and model credibility (confidence to use model and information derived, level acceptable to the user). Additionally, [22] measures performance for industrial models by primary measures such as throughput, system cycle or response time, and work in process. In addition, a number of secondary or explanatory measures may be of interest, such as resource utilization, size of local buffers, and throughputs for subsystems or particular types. Many definitions of the various aspects of validity in qualitative research specifically refer to the how-to-dos of establishing credibility, authenticity, trustworthiness, criticality, and integrity, to name a few. Dellinger and Leech [23] identify them as primary aspects. Secondary criteria refer to important and flexible aspects of quality criteria that are in addition to the primary criteria, including explicitness, vividness, creativity, thoroughness, congruence, and sensitivity.

6.2.4. Some concluding remarks

We concluded from this literature review that many methods exist to validate methodologies, models and products. On the highest level, we could identify two approaches: empirical and theoretical validation. However, to validate the DSDM, our search concluded with a negative result. We found that none of the found validation methods is directly applicable. Nevertheless a general validation approach can be adapted to our context using some specific changes and additions. For the specific purpose of validating the DSDM, we decided to use the method of the validation square [17], since it was developed to be engineering-oriented. In this framework both empirical and theoretical validations are considered.

The validation square method is comparable to the framework proposed by Briand et al. [10] and those of Schön and Agryris [11], but its major advantage is that it handles and combines different levels of complexity (functional, structural elements, interfaces/communication, technical solutions). Considering the needed validation criteria, we concluded that the proposed criteria are useful, but still need to be combined for our specific purpose. In the following Section, the purpose of the authors of the validation square is shown and in addition, we also discussed what adjustments are needed and what criteria would be best for our specific context of the designerly software development methodology.

6.3. Theoretical and methodological fundamentals

6.3.1. Initial interpretation of the validation square

In this section we build further on the concept of the validation square, presented by Pedersen and Seepersad [17]. We used it as a generic framework which could be operationalized for our specific case. According to the authors, the purpose of the 'validation square' (VS) method is to introduce a rigorous framework for validating engineering design methods. Considering the theoretical and methodological fundamentals, the framework is based on two primary tasks: establishing: (i) the structural validity of the design methodology, and (ii) the performance validity of the design methodology. These two primary aspects are incorporated in the validation square. As illustrated in Figure 6.3, the validation square is divided into four quadrants. Considering the theoretical and methodological fundamentals, the framework is based on two primary tasks: establishing (i) the structural validity of the design methodology (left half), and (ii) the performance validity of the design methodology (right half). In addition, there is also a division into a domain-independent and a domain-specific upper and lower half, the latter is associated with the validity of the method for the domain-specific examples investigated in the research, the matter for broader domains of application. The theoretical parts have a predictive nature while the practical part has a reflective approach for the validation.

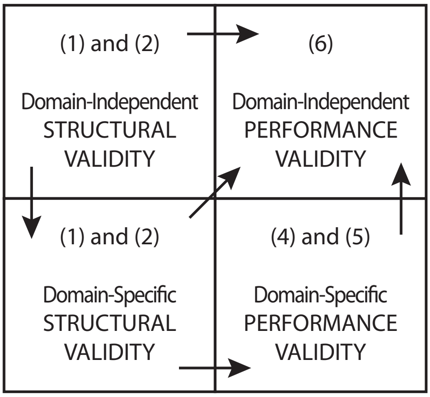


Figure 6.3. The validation square

6.3.2. Re-interpretation of the method in application context

To operationalize it, the principle of the validation square has to be transferred to a validation method that is specialized for our context. Since the VS method is specified in a general sense, a first action will be needed to adapt it to our application of validating a designerly software development methodology. Therefore, following changes were essential:

1. Although several authors, [7, 32] recommend to use multiple cases in order to adopting several different viewpoints, we based in this research on only one application case. The principle of extrapolation was used to reason about other possible applications. The only condition is that this single case is representative which means that the case covers the

complete methodology with all constructs. If this is true, other examples are redundant since they cannot fulfill a role in the validation process that is not fulfilled by the case.

2. Other cases were used for deductive reasoning to extend the domain-specific reasoning. By extrapolating from the reference case, we could not prove that all cases are inductively good but deductive. As a result, using the validation square method we are not able to tell more about cases that are not similar to the reference case.

3. Initially, the validation square intended for both qualitative and quantitative means. We targeted that both performance and structural validation should be conducted in a qualitative manner, based on the principle of reasoning with consequences.

4. The (internal) validation of the individual constructs was already discussed in their respective chapters. Nevertheless, to externally validate the entire DSDM, external validation of each construct as part of the overall methodology is still needed.

5. To apply the validation square method, the most important performance indices should be identified, since the initial interpretation did not touch upon the necessity of criteria. They did not suggest possible criteria that could be used. We can base on the criteria that were identified in the literature study to identify the most appropriate ones.

6.4. Operationalization of the methodology for our particular case

To operationalize the quadrant-based external validation method (QEVM) we considered both the aspects and steps that should be detailed for each quadrant. Figure 6.4 provides an overview of the complete validation process including the execution steps for each quadrant. In the following subsections, each quadrant is explained in more detail, especially focusing on providing evidence on why the step is needed.

6.4.1. Clarification on the assessment of theoretical structural validity

The objective of this test is to do a domain-independent structural validation of both the overall method and the individual parent constructs. The validity can be measured using the information flow in the entire process of the methodology, as it is valid if the generation of all required pieces of information is supported through the methodology (= necessary condition) and when the producing of the information happens when it is needed (= sufficiency condition). To achieve this, the requirements of the outcomes of the method and the process by which the method generated the outcomes should be known (STEP 1). High level requirements should be broken down into a hierarchical set of more specific requirements. In addition, the characteristics of the intended context for application of the method should be included and may include details of the intended physical domains, types of performance parameters, classes of variables, and product architectural characteristics. As the meta-methodology is based on different constructs it is also important to identify the parental relationships between the meta-methodology and its constructs and complete

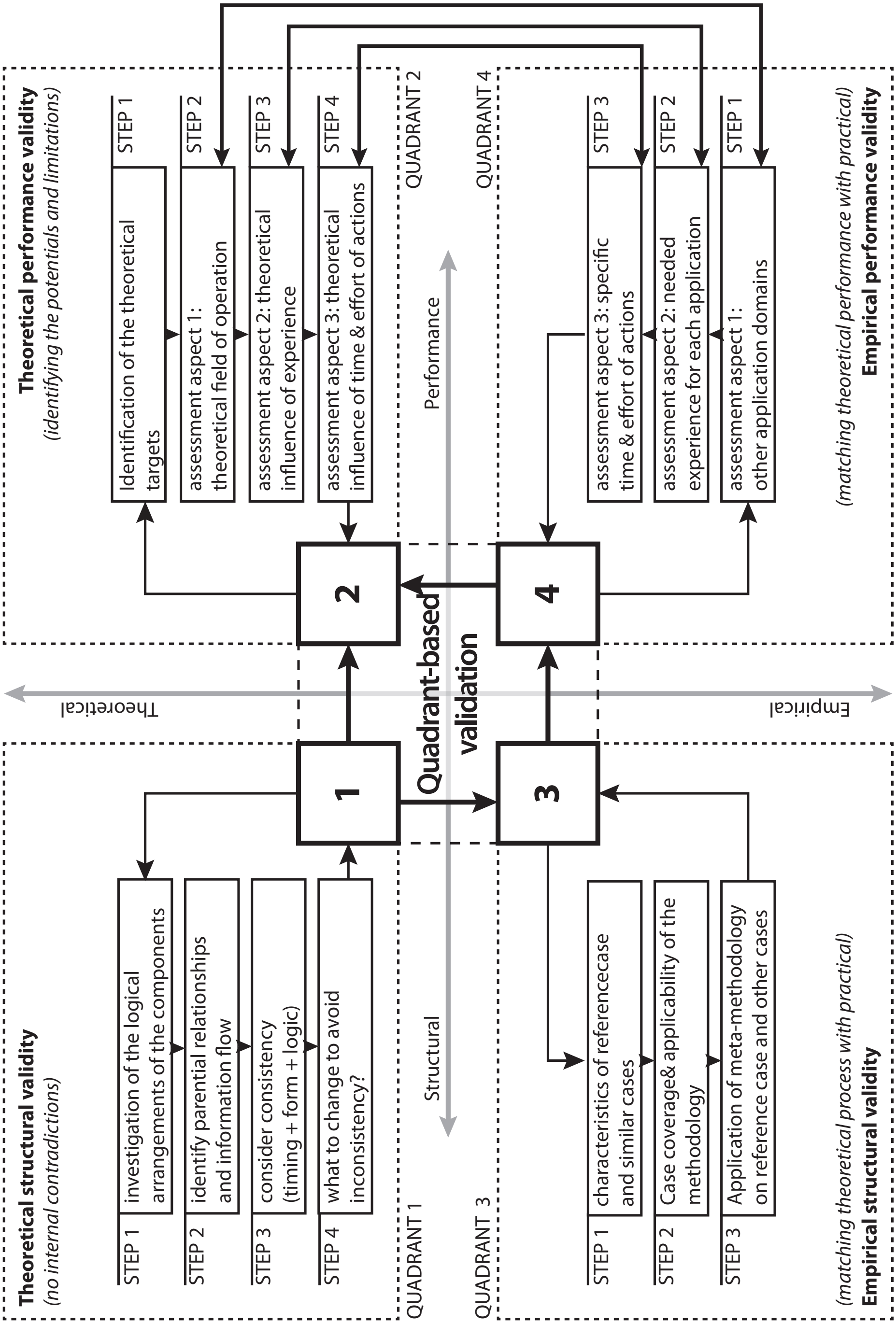

Theoretical performance validity
(identifying the potentials and limitations)
STEP 1
Identification of the theoretical targets
STEP 2
assessment aspect 1: theoretical field of operation
STEP 3
assessment aspect 2: theoretical influence of experience
STEP 4
assessment aspect 3: theoretical influence of time & effort of actions
QUADRANT 2
QUADRANT 4
STEP 3
assessment aspect 3: specific time & effort of actions
STEP 2
assessment aspect 2: needed experience for each application
STEP 1
assessment aspect 1: other application domains
(matching theoretical performance with practical)
Empirical performance validity
Performance
Structural
Theoretical
Empirical
1
2
3
4
Quadrant-based validation
Theoretical structural validity
(no internal contradictions)
STEP 1
investigation of the logical arrangements of the components
STEP 2
identify parential relationships and information flow
STEP 3
consider consistency (timing + form + logic)
STEP 4
what to change to avoid inconsistency?
QUADRANT 1
QUADRANT 3
STEP 1
characteristics of referencecase and similar cases
STEP 2
Case coverage& applicability of the methodology
STEP 3
Application of meta-methodology on reference case and other cases
(matching theoretical process with practical)
Empirical structural validity

Figure 6.4. Introducing dedicated executions steps in the validation square

the information flow (STEP 2). This information should be collected in order to establish the internal consistency of the proposed design methodology by considering the timing, formulations and logic between the different constructs (STEP 3). Lastly, suggestions should be made on how inconsistency can be avoided (STEP 4).

6.4.2. Clarification on the assessment of theoretical performance validity

The objective of this quadrant is to validate the domain-independent performance of the methodology. The theoretical performance is separated from the practical implementation, which can be used as testing means. To validate the theoretical performance, all targets that should be achieved by the methodology, must be identified and performance aspects must be defined. To execute the validation in this context, three criteria of performance validation should be identified. The first criterion is the identification of the theoretical field of operation (STEP 1). Questions as "Can we find applications domains for methodological efficiency?" and "What are the potential domains of application?" should be answered, by reasoning, to identify the characteristics where it will and where it will not work properly. Finally, the boundaries should become clear by rational analysis and interpretative reasoning with consequences. The next criterion to discuss is the influence of experience on the performance (STEP 2). User's experience might have a big influence on the performance of the methodology, so it is important to know what experiences (skills, competences, knowledge …) are essential in general to use the methodology on an appropriate level. The third criterion focusses on the theoretical influence of time and effort of all actions (STEP 3), because the amount of time and effort should be known in advance to be able to balance it with the added value of the methodology.

6.4.3. Clarification on the assessment of empirical structural validity

The objective of this quadrant is to do a domain-specific structural validation. Practically, it involves building confidence in the appropriateness of the example problem. Consequently this means that the characteristics of the example problem must be mapped (STEP 1) to see how the methodology and the example case are covering each other (STEP 2). Consequently, on the one hand, it is important to show that the meta-methodology can be applied for the case and what aspects it covers (and which not). On the other hand, the characteristics of both the design problems for which the methodology is intended and those that are not covered must be identified. By (i) documenting that the data from the example can be used to support conclusions with respect to the performance of the design methods, (ii) documenting the example's simplified assumptions and (iii) mentioning that its data can be compared, contrasted, and processed to evaluate the performance of the proposed design method, the appropriateness of the example case should be shown (STEP 3).

6.4.4. Clarification on the assessment of empirical performance validity

In this quadrant the aim is to validate the domain-specific performance of the software development methodology. This should be achieved by checking how the targets are achieved and what the performance is of the methodology in reaching them. The same performance validation criteria, as in Section 4.2, will be considered but from the perspective of application examples. In order to enlarge our reasoning other application examples should be identified for which the methodology can be as efficient as in the source application or even more efficient (STEP 1). All application examples should be evaluated by following aspects: (i) resembling functionality, (ii) user requirements, (iii) necessary resources, (iv) level of sophistication: modeling, data, environment, (v) communication intensity, and (vi) level of standardization (reusability). Furthermore, the specific experiences needed (STEP 2) and the specific time and effort of all actions (STEP 3) of the identified possible application examples should be discussed. As in the other quadrant, the sample applications should be discussed by rational analysis and interpretative reasoning with consequences. In order to be able to obtain conclusion from the validation square, the theoretical and empirical performance validity should be compared for each of the above mentioned aspects.

6.5. Execution of the validation

In this Section, we want to revisit the validation assessment of the designerly software development methodology by summarizing the conclusions, derived in the particular quadrants, after applying QEVM. Regarding the empirical validation, we used a single reference application case in a deductive reasoning. Because we can conclude, based upon the results of the experiments, that the reference case was effectively developed using the DSDM, we accept its theory to be true and consequently we could claim comparable things for those products that are part of the same family, and have comparable characteristics.

6.5.1. Execution of the assessment specified in the first quadrant

The structural validation could be achieved by discussing the structure and information flow on two levels: (i) universal structure of DSDM and (ii) the level of the single phase methodology constructs. The DSDM is focusing on three phases in the development process of software products, and for each phase a specific construct methodology was developed. As shown in Figure 6.5, the information of the developed software follows a logic path through the process of DSDM and its constructs.

On an abstract level, we can say that the DSDM methodology has a linear structured process in which a phase must be finished before going to the next step. However, the processes of the different constructs are both iterative and linear: depending on the complexity it is in some parts necessary to do more iteration before having a satisfied result. The consecutive logic of the different constructs in the different phases is supporting the constructive character of the methodology. In the information flow, four moments of data transformation can be identified: (1) data transformation that is needed as a preparation for the construct methodology, (2) data transformation that is performed during the methodology execution, (3) Data transformation during the concluding phase of the methodology, and (4) Data transformation in between the different phases of the software development that must be

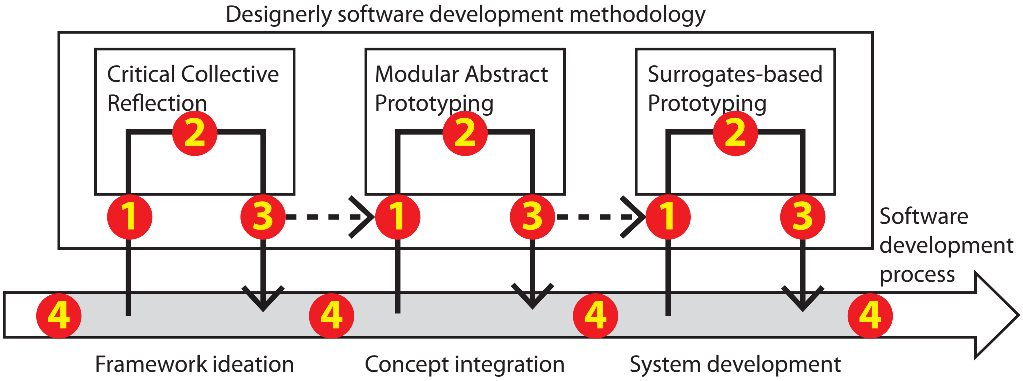

Designerly software development methodology
Critical Collective Reflection
Modular Abstract Prototyping
Surrogates-based Prototyping
2
1
3
4
Software development process
Framework ideation
Concept integration
System development

Figure 6.5. Logical flow of data through the DSDM

carried out by the developers in order to be able to go to the next phase (The numbers are referred to in Figure 6.5).

6.5.2. Execution of the assessment specified in the second quadrant

During the performance validation, the potentials and limitations of the DSDM regarding the performance were identified. The targets that should be reached are shown in Figure 6.6. To know the level of validity, we discussed the theoretical performance of reaching all targets, according to: (i) the field of operation, (ii) the influence of experience on the performance, and (iii) the influence of time and effort of actions. Typical application cases have a complex functionality, user requirements that are rather uncertain and unclear in the beginning of the process, and high level of sophistication due to environmental aspects, modeling need and data processing. More detailed characteristics and boundaries are shown in Table 6.1. The theoretical field of operation can be based on the characteristics mentioned in the first column. In the second column, their boundaries are given.

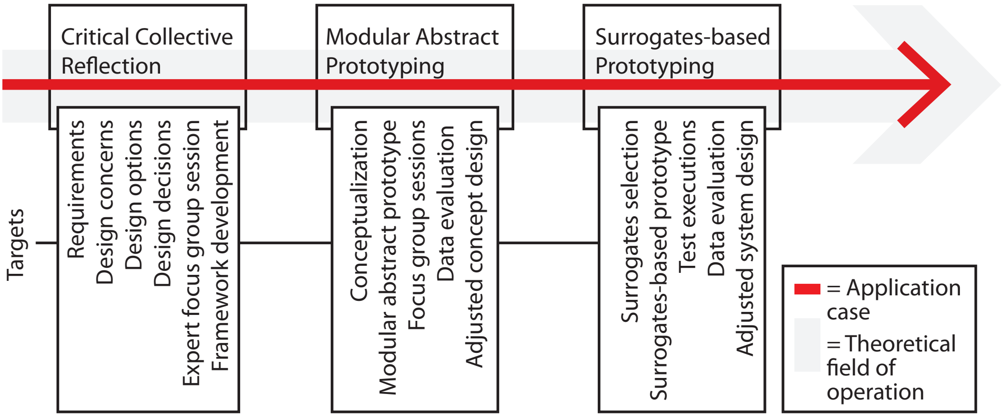


Figure 6.6. Performances of the DSDM

Table 6.1. Characteristics and boundaries of the operation field of the DSDM

Characteristics	**Boundaries**
Resembling functionality	complex functionalities evolving
User requirements	abstract, vague requirements evolving
Communication intensity (stakeholder involvement)	multiple user involvement high involvement (from stakeholder and or developers side)
Level of sophistication: environment (time, budget, people, skills, organization structure)	to support multidisciplinary development teams (different people with different skills) small budget, limitations in time and organizational aspects
Level of sophistication: modeling	High level of sophistication (advanced modeling required to deal with complexity, and to be able to deal with the different stakeholders throughout the different phases)
Level of sophistication: data	High level of sophistication (knowledge base)

Table 6.2. Identification of experiences/ action

Phase	**Actions**	**Needed experience/skills**
CCR:	1. Requirements engineering	Research skills
	2. Deriving design concerns	Reasoning skills
	3. Deriving design options	Analyzing and reasoning skills
	4. making design decisions	Analyzing and reasoning skills
	5. Organizing and executing expert session	Presenting, communication skills
	6. Framework development	Synthesizing and reasoning skills
MAP:	7. First conceptualization	Reasoning skills
	8. MAP design	Design skills, graphic demonstration skills
	9. Execution of focus group sessions	Communication, presenting skills
	10. Data evaluation	Analyzing, reasoning skills
	11. Adjusted concept design	Synthesizing and reasoning skills
SBP:	12. Surrogates selection	Reasoning skills
	13. SBP design	Design skills, computer skills
	14. Execution of tests	Computer skills, reasoning skills
	15. Data evaluation	Synthesizing and reasoning skills
	16. Adjusting the system design	Synthesizing and reasoning skills

To identify what experience matters in the use of the DSDM, in Table 6.2, an overview is given on the competences, experiences and skills that have an influence on each action that is essential to reach a specific target. The amount of each skill needed will be very much depending on the context of application, as well as the other specific required experiences and knowledge in the domain of application. However, in general, we found that instead of just programming skills, the development team also needs research, reasoning, presentation, design computer, and graphic skills to use the DSDM. In addition, we also had to discuss the time and effort necessary to execute each action. Different effort and time is needed depending on the specific action. However, the total time required to execute the development using the DSDM is considered to be lower than other approaches, because fewer iterations are needed, and because a higher SH-adjustment is achieved. Figure 6.7, an overview is given of the estimated effort and time for each theoretical action. We reused the actions as described in Table 6.2, who all need a certain time and effort to be completed.

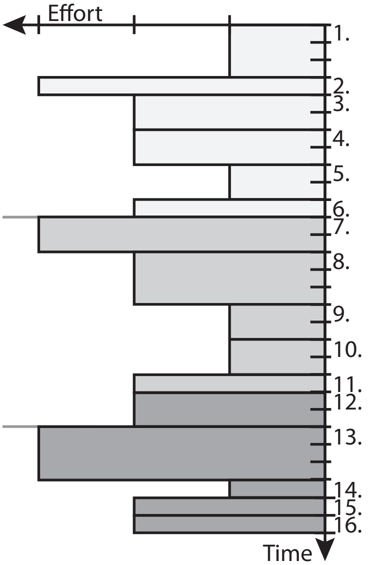


Figure 6.7. Estimated effort and time (on a scale from 1 - 3) (numbers are referring to the actions described in Table 6.2)

6.5.3. Execution of the assessment specified in the third quadrant

The empirical structural validation was conducted by matching the theoretical process with the practical application reference case developed during the entire research. Regarding the empirical validation, we used a single reference application case in a deductive reasoning. Because we can conclude, based upon the results of the experiments, that the reference case was effectively developed using the DSDM, we accept its theory to be true and consequently we could claim comparable things for those products that are part of the same family, and have comparable characteristics. The empirical performance validity was achieved by identifying the performance by reasoning on the reference case plus possible application cases: (i) software for an alarm system, (ii) a company information system to manage production, (iii) a product-service system for furniture reuse, and (iv) an interactive video-wall to communicate about cultural events. In addition, comparison was accomplished to compare the conclusion of the theoretical validation with the empirical validation.

The empirical structural validity was carried out by matching the theoretical process with the practical application case. Before we could do this, we had to map the characteristics of the reference case: the major objective of the tool is to support designers in their decision making process on smart energy saving possibilities. Since process automation is not desired, continuous user interaction will be used to support the designer in his thinking process. Throughout this guidance, the awareness must be enhanced on how to use ubiquitous controllers to save energy, and furthermore a community can grow on the possibilities of energy efficiency using ubiquitous controllers. The design support software should be an online application and is a typical example of a complex software product since it is based on engineering principles and has a research oriented design. For such software it is difficult to formulate the complete requirements in advance, due to many reasons. Most importantly is that different stakeholders are involved in the process and in the use of the product, i.e. product designers, software developers and knowledge engineers. In addition, these stakeholders request high level of involvement. On the other hand, low amount of time, small budget, single person-team, few programming skills, high development skills are also some important characteristics. This is in contrast with the fact that the stakeholders request high level of modeling to be able to discuss the design. The complex data structure of how the energy saving opportunities can be combined with the multiple functions of the electronic household appliances, requests a knowledge type of data base and flexibility of data types, together in a complex integrated model.

Table 6.3. Methodology and case coverage

DSDM – relevance indicator	**Software case – fulfillment indicator**
to support the software development process	a software tool for smart energy saving
deal with uncertain and unclear user requirements in the beginning of the process	for the software case, it is difficult to formulate the complete requirements in advance, due to many reasons.
was developed to deal with complex functionalities	it is a complex software product (based on engineering principles) the complexity will grow if a community can grow next to it
the DSDM aims to co-design with stakeholders	the stakeholder request high level of involvement.
support as a communicating means project documentation is achieved through the prototypes	most importantly is that different stakeholders are involved in the process and in the use of the product. moreover, high level of modeling is needed able to discuss the design.
focusing on relative complex projects multi-abstraction levels	combining energy saving in household appliances knowledge base with multiple data types, in a complex integrated model
DSDM is especially for multi-disciplinary teams	low amount of time, small budget, single person-team, few programming skills, high development skills are also some important characteristics of the software case

Matching the theoretical process with the practical application case, we can conclude that the application fulfills the specific requirements of the DSDM and that DSDM was a relevant methodology for the development of the software case. In Table 6.3, both the fulfillment indicators, who link the case to the application specific requirements and the relevance indicator, who show how much the methodology is relevant for the application, are shown. It is important to show that the meta-methodology can be applied for the case and what aspects it covers (and which not). On the other hand the characteristics of the design problems for which the methodology is intended must be identified plus those that are not covered. A comparison of the two 2 logical processes of the theory of meta-methodology

Table 6.4. Comparison of the application case and the methodology

Phase	Meta-methodology actions	Concrete case development steps
CCR:	1. Requirements engineering	Investigation of the five different knowledge domains related to the software case and the context.
	2. Deriving design concerns	Identifying the most critical design problems
	3. Deriving design options	Searching solutions for each of the problems
	4. Making design decisions	Development of the first theory of how the software can work
	5. Organizing and executing expert session	Execution of an expert session with experts in the five different knowledge domains.
	6. Framework development	Comparing conclusions of the session with the literature study and developing a framework of the software
MAP:	7. First conceptualization	Developing a real life story in which all aspects of the software are addressed.
	8. MAP design	Converting the story into narration and enactment of different modules.
	9. Execution of focus group sessions	Inviting participants of the different stakeholders and organize stakeholder discussion focus group sessions
	10. Data evaluation	Write down, process and analyze semantically the data of the focus group sessions and compare them with each other
	11. Adjusted concept design	Conclude how the concept must be improved
SBP:	12. Surrogates selection	Identify the function sets and decomposition and select relevant software that can be used to mimic these function sets
	13. SBP design	Develop the prototype
	14. Execution of tests	Test the functionality of the software
	15. Data evaluation	Analyzing test results to identify change proposals
	16. Adjusting the system design	Last improvements of the software before production.

on the one hand and the process of the concrete case development on the other hand was conducted, as visualized in Table 6.4. By comparing the different methodology actions and the concrete steps in the case development, we can make a link of what parts of the software case development are referring to a certain action of the meta-methodology.

6.5.4. Execution of the assessment specified in the fourth quadrant

As shown in Figure 6.6, not only the performance of reaching each target in the reference case is considered, but also the performances of the family cases that were retrieved in the third quadrant (6.5.3). In Table 6.5, an overview is given of the characteristics of the potential applications together with the characteristics of the original example case. Next, considering the theoretical influence of experience, we can conclude that although the amount of each skill needed is also depending on the context of the application, also the methodology requires specific skills and experiences. In Table 6.6, an overview is given of the amount of experiences needed to develop the applications.

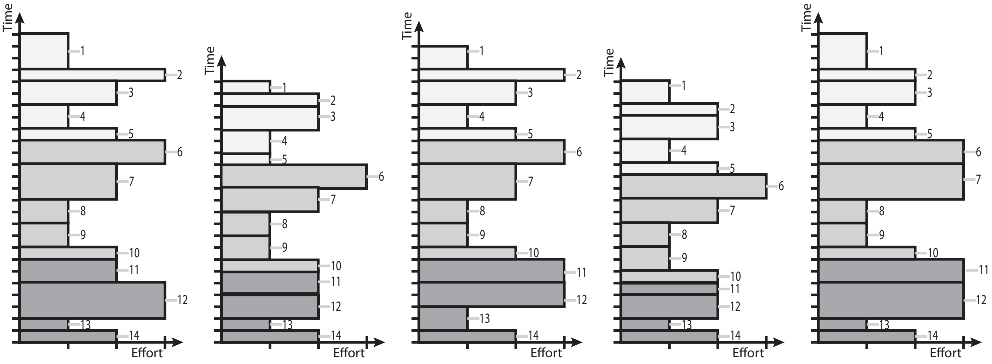
Table 6.5. Discussion of potential applications

	Smart energy tool	**Alarm system**	**Production management**	**Furniture system**	**Interactive video-wall**
resembling functionality	supporting designers in decision making	communicate with users, police, neighbors…	structure and manage the production process	being a channel for reuse of furniture	communicate about cultural events
main require-ments	show cases; support selection process; calculate trade-off	inform; detect intrusion; frighten and warn	show production process info; support & control the process	inform on available furniture; manage sale, transport…	react upon certain gestures; inform people
level of so-phistication: environment	high	high	high	high	high
level of so-phistication: modeling	high	high	normal	normal	high
level of so-phistication: data	high: knowledge base needed	normal	high	high: flexible 'passage'	high
communica-tion intensity	high	normal	high	high	high

Table 6.6. Reasoning about the amount of experiences needed to develop the applications

	Smart energy tool	Alarm system	Production management	Furniture system	Interactive video-wall
Research skills	▨▨▨□	▨▨□□	▨▨□□	▨□□□	▨▨▨□
Reasoning skills	▨▨▨▨	▨▨▨□	▨▨▨▨	▨▨▨□	▨▨▨▨
Presentation skills	▨▨□□	▨▨□□	▨▨□□	▨▨▨□	▨▨□□
Design skills	▨▨▨□	▨□□□	▨□□□	▨▨▨□	▨▨▨□
Computer skills	▨▨□□	▨▨□□	▨▨□□	▨▨□□	▨▨□□
Computer graphic skills	▨▨□□	▨▨□□	▨▨□□	▨▨□□	▨▨▨▨
Programming skills	▨□□□	▨□□□	▨▨□□	▨□□□	▨▨▨□

Lastly, the time and effort required for each action in the development of the identified application cases matches in general the theoretical one. Differences are mainly due to different levels of complexity caused by the number of stakeholders and functionalities. In Figure 6.8, a schematic overview of these is given.



Smart energy tool Alarm system Production system Furniture system Interactive video-wall

Figure 6.8. Needed time and effort to develop the application examples (numbers are referring to the actions described in table 6.2)

6.6. Findings of the validation assessment

In this Section we want to revisit the validation assessment of the designerly software development methodology by summarizing the findings of each quadrant after applying the quadrant-based external validation method (QEVM).

6.6.1. Findings on theoretical structural validation

We found that the DSDM is theoretical structural valid. We argue this statement with the following arguments on the logic of the structure and information flow: Based on the logic of DSDM and constructs, we can conclude that the DSDM is composed of three single phase methodologies that logically support the designers in their development process. Moreover, the logic of supporting the software development process is high: the three single-phase methodologies each influence a different phase of the methodology and support the stakeholder-involvement on a structured and specific manner. We found that the methodology supports the development of the right information at the right time, which provide a high logic of information flow. Due to the evolution in the design process, the DSDM also supports the necessary data transformation to consider all aspects of the design process. In short, no critical internal contradictions were found in the DSDM and considering the use of the critical collective reflection methodology, the modular abstract prototyping methodology, and the surrogates-based prototyping methodology. To avoid inconsistency in the use of the DSDM, it is important: (i) to follow the logical order of the prescribed phased on the development process; (ii) to apply all sub-methodologies: in some cases it might be better to not execute one of the sub methodologies. However, if one construct is not used, essential information might be missing to go further with next construct. And lastly, (iii) to transform data properly or information might be lost or twisted.

6.6.2. Findings on the theoretical performance validation

To identify the level of theoretical performance validity we identified the characteristics where the methodology will and where it will not work properly, the different skills and experiences that were crucial to perform each action of the DSDM, and the theoretical needed time and efforts. The evaluation of these aspects gave further information on what is required to reach all targets and the potentials and limitations regarding the performances of the DSDM. It was also conducted to determine when the DSDM has a valid theoretical performance. We conclude that the DSDM is valid if the application case has a complex functionality, evolving, uncertain and unclear requirements in the beginning of the process, and a high level of sophistication due to environmental aspects, modeling need and data processing needs. To reach all targets, we found that the DSDM is valid if the designers have following competences, experiences and skills: research, reasoning, presentation, design computer, and graphic skills plus those related to the application context. Lastly we can argue that the validity is also set by the time and effort required to reach all targets. We found that the DSDM has a high level of validity because the necessary time and effort results in a better adjusted solution in a relative short time frame, using an efficient and effective process.

6.6.3. Findings on the empirical structural validation

The DSDM was found valid from empirical structural point of view. We could argue this because the application of the DSDM resulted in a logical developed product, during an organized information flow. By comparing the fulfillment indicators, who link the case to the application specific requirements and the relevance indicator, who show how much the methodology is relevant for the application, we concluded that the reference case was useful to test the empirical validity of the DSDM. We found that in both the reference case and in the generated family cases, no inconsistencies or contradictions could be found in the executed data transformation actions and in the retrieved information, not in the extrapolations to other cases. The DSDM was found to be a robust, sensitive and transparent methodology that is focusing on involving the stakeholders and reducing the complicatedness by bringing structure.

6.6.4. Findings on the empirical performance validation

The DSDM's empirical performance validity was found to be high. The performance of the reference case and the other possible applications were discussed. To consider the theoretical field of operation, all application examples were evaluated by following aspects: (i) resembling functionality, (ii) user requirements, (iii) necessary resources, (iv) level of sophistication: modeling, data, environment, (v) Communication intensity, and (vi) Level of standardization (reusability). The identified application cases all match the theoretical application field and would profit from the application of the DSDM, i.e. the needed experiences, time and effort would result in a better stakeholder-oriented product that is developed in an efficient and effective manner. We found that the DSDM methodology achieves the targets set in Chapter 2: the methodology supports the development of understandable, reliable, efficient, and modifiable products by managing the complexity (using separation of concerns and different level of abstraction), dealing with evolving requirements, and relying on early confirmation by constant stakeholder involvement.

6.7. Concluding remarks

We derived many useful conclusions based on the validation effort presented in this chapter regarding (i) the validation method, and (ii) the validation outcome of the DSDM. The most important ones are described in the following propositions:

Proposition 1:

> The DSDM has been proved to be a valid methodology for the development of software products that have complex functionality, user requirements that are rather uncertain and unclear in the beginning of the process, and high level of sophistication due to environmental aspects, modeling needed and data processing.

Proposition 2:

Application of the DSDM methodology supports the development of understandable, reliable, efficient, and modifiable products by managing the complexity, dealing with evolving requirements, and relying on stakeholders.

Proposition 3:

The quadrant-based external validation method combines structural and performance assessment actions in both the theoretical domain and the application domain. In each quadrant, the different steps allow both qualitative and quantitative assessment according to various criteria in a reflexive manner, starting out from the main structural and performance characteristics.

Proposition 4:

The QEVM is a valuable method for the validation of this designerly software development methodology.

Proposition 5:

The proposed approach has a large application potential and is flexible enough in single-case, reflexive, context dependent assessments. Further research is needed to explore whether this quadrant-based external validation method can be applied in a context independent manner.

6.8. References

[1] Sargent, R.G., (2005), "Verification and validation of simulation models", Proceedings of the 37th Winter Conference on Simulation, Orlando, Florida, pp. 130-143.

[2] Seepersad, C.C., Pedersen, K., Emblemsvåg, J., Bailey, R., Allen, J.K., and Mistree, F., (2006), "The validation square: How does one verify and validate a design method? Decision making in engineering design", in: Decision making in engineering design, Lewis, K.E., Chen, W., Schmidt, L.C. (Eds.), ASME Press, p. 303.

[3] Palmquist, M., (2013), "Glossary of key terms", in: Writing@CSU (Ed.), Colorado State University, 2013, p. http://writing.colostate.edu/guides/guide.cfm?

[4] Regents of the University of Michigan, (2013), "Research glossary", Chile Care & Early Education Research Connections Office of Planning, Research and Evaluation, Administration for Children and Families, U.S. Department of Health and Human Services 2013, pp. http://www.researchconnections.org/childcare/research-glossary.

[5] Shuttleworth, M., (2009), "Types of validity", Vol. 2013, Explorable.com, 2009, pp. http://explorable.com/types-of-validity.

[6] Macal, C.M., (2005), "Model verification and validation", in: models", w.i.t.a.s.s.m.a. (Ed.), center for complex adaptive agent system simulation, Chicago, IL, 2005.

[7] Lee, S.W., and Rine, D.C., (2004), "Case study methodology designed research in software engineering methodology validation", Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'04),, Banff Centre, Banff, Alberta, Canada,, pp. 117-122.

[8] Gottschalk, P., (2002), "Empirical validation procedure for the knowledge management technology stage model", Informing Science The International Journal of an Emerging

Transdiscipline, Vol. 5 (4), p. 30.

[9] Kitchenham, B., Linkman, S., and Law, D., (1997), “Desmet: A methodology for evaluating software engineering methods and tools”, Computing & Control Engineering Journal, Vol. 8 (3), pp. 120-126.

[10] Briand, L., El Emam, K., and Morasca, S., (1995), “Theoretical and empirical validation of software product measures”, Proceedings of the ISERN-95-03, international software engineering research network p. 23.

[11] Schön, D., and Argyris, C., (1975), “Theory in practice: Increasing professional effectiveness”, Jossey-Bass, San Fransisco, p. 224.

[12] Wallace, D.R., and Fujii, R.U., (1989), “Software verification and validation: An overview”, Software, IEEE, Vol. 6 (3), pp. 10-17.

[13] Hickey, A., and Dean, D., (1998), “Prototyping for requirements elicitation and validation: A participative prototype evaluation methodology”, Proceedings of the Americas Conference on Information Systems (AMCIS 1998), pp. 797-800.

[14] O’Keefe, R., and O’Leary, D., (1993), “Expert system verification and validation: A survey and tutorial”, Artificial Intelligence Review, Vol. 7 (1), pp. 3-42.

[15] Frey, D., and Dym, C., (2006), “Validation of design methods: Lessons from medicine”, Research in Engineering Design, Vol. 17 (1), pp. 45-57.

[16] Simon, H.A., (1990), “Invariants of human behavior”, Annual Review of Psychology, Vol. 41, pp. 1-19.

[17] Pedersen, K., Emblemsvåg, J., Bailey, R., Allen, J.K., and Mistree, F., (2000), “Validating design methods & research: The validation square”, Proceedings of the ASME Design Engineering Technical Conferences, ASME, Baltimore, Maryland, p. 12.

[18] Landry, M., Malouin, J.-L., and Oral, M., (1983), “Model validation in operations research”, European Journal of Operational Research, Vol. 14.

[19] Sargent, R.G., (1984), “A tutorial on verification and validation of simulation models”, Proceedings of the winter simulation conference, Sheppard, S., Pooch, U., Pegden, D. (Eds.), IEEE Press, pp. 114-121.

[20] Jensen, S.Ø., (1995), “Validation of building energy simulation programs: A methodology”, Energy and Buildings, Vol. 22 (2), pp. 133-144.

[21] Kleijnen, J.P.C., (1995), “Verification and validation of simulation models”, European Journal of Operational Research, Vol. 82 (1), pp. 145-162.

[22] Carson, J.S., (2002), “Model verification and validation”, Proceedings of the 34th conference on Winter simulation: exploring new frontiers, Winter Simulation Conference, San Diego, California, pp. 52-58.

[23] Dellinger, A.B., and Leech, N.L., (2007), “Toward a unified validation framework in mixed methods research”, Journal of Mixed Methods Research, Vol. 1 (4), pp. 309-332.

[24] Lincoln, Y.S., and Guba, E.G., (1985), “Naturalistic inquiry”, Sage., Beverly Hills, CA, p. 75.

[25] Campbell, D.T., and Stanley, J.C., (1963), “Experimental and quasi-experimental designs for research.”, Rand McNally, Chicago, p. pp.

[26] Eisner, E.W., (1991), “The enlightened eye: Qualitative inquiry and the enhancement of educational practice”, Prentice Hall, Upper Saddle River, NJ, p. 264.

[27] Maxwell, J.A., (1992), “Understanding and validity in qualitative research”, Harvard Educational Review, Vol. 62, pp. 279-300.

[28] Eisenhart, M.A., and Howe, K.R., (1992), “Validity in educational research”, in: The handbook of qualitative research in education LeCompte, M.D., Millroy, W.L., Preissle, J. (Eds.), Academic Press, San Diego, CA, pp. pp. 643-680.

[29] Bednar, J.A., and Robertson, D., (2007), “Verification and validation”, SAPM Spring, 2007.

[30] Olewnik, A.T., and Lewis, K.E., (2003), “On validating design decision methodologies”,

Proceedings of the ASME 2003 Design Engineering Technical Conferences - Design Theory and Methodology Conference, ASME, Chicago, IL., p. 10.

[31] Jagdev, H.S., Browne, J., and Jordan, P., (1995), "Verification and validation issues in manufacturing models", Computers in industry, Vol. 25 (3), pp. 331-353.

[32] Eisenhardt, K.M., (1989), "Building theories from case study research", The Academy of Management Review, Vol. 14 (4), pp. 532-550.

Chapter 7

Conclusions, reflections and future research

7.1. Conclusions

In this Section, the conclusions of this promotion research are described. We concluded on various aspects and for each aspect we described: (i) the conditions that are assumed, (ii) findings from the research, and (iii) the propositions.

7.1.1. Concerning the Designerly Software Development Methodology

Proposition 1: *In order to achieve an optimum support and efficiency, the proposed DSDM has been developed as a multi-phase stakeholder-oriented designerly methodology, which offers proper procedures, instruments and methods for three single-phase methodologies: (i) critical collective reflection, (ii) modular abstract prototyping, and (iii) surrogate-based prototyping.*

As a requirement, it can be stated that a designerly software development methodology was needed which could enhance the stakeholder communication, involvement and co-development in the software development process. It seemed to be necessary to extend the influence of the DSDM to the following three phases of the software development process: (i) framework ideation, (ii) concept integration, and (iii) system development.

Based on the exploration, it was found that the underpinning theory of the DSDM was based on these three ideas: (i) stakeholder involvement enables qualitative change proposals, (ii) managing complexity and evolvability is a critical issue, and (iii) changing fidelity during the process is a manner to handle the total complexity. Based on empirical studies and literature review, it has been found that using the component methodologies in combination lend itself to a more effective and stakeholder-centered process.

7.1.2. Concerning the research approach

Proposition 2: *Concurrent development of the parts and the whole of the methodology and applying it to an evolving reference case, lends itself to a short cycle learning process.*

We stated that the DSDM should be stakeholder centered and industry-oriented. We found that applying a reference case allowed scenario-based and process-oriented thinking. The development of the DSDM cast light on the need for co-development. Therefore we studied how to apply the DSDM in context.

Proposition 3: Braking down the research project into a sequence of interconnected research cycles not only helps structuring the work, but also finding the necessary and sufficient scope and balance of the research topics and activities.

We found that a methodological framing could be applied in this research. Thinking as evolving research means is in line with the Design Inclusive Framing methodology.

7.1.3. Concerning the reference case

Proposition 4: DSDM can be applied to all cases of interactive software development which show a similar structure and (performance) targets as the reference case.

We hypothesized that a support tool was needed for smart energy saving in consumer durables using ubiquitous augmentation. Based on the research, we found that smart energy saving using ubiquitous controllers is a complex task for product designers. Supporting designers in smart energy saving could be achieved by a software tool that support the designers-thinking process with the adequate structure, information, and simulations. The software tool for smart energy saving had to be interaction-based rather than algorithm-based or computational.

7.1.4. Concerning the phase of framework ideation using CCR

Proposition 5: The methodology of CCR enables better collective requirements engineering and framework conceptualization through the direct reflection of expert-stakeholders on the proposal demonstrated by the software developers

We considered the following conditions: the CCR should deal with the complexity of under-defined and conflicting problems. It should be able to handle a broad solution space. It should consider the multiple aspects of conceptualization (e.g., functional or structural). The solutions need multidisciplinary knowledge. It should consider the emergent needs of the stakeholders, which may drastically change in the entire set up.

We found that stakeholder involvement is crucial in the first decision making process with a view to identify a relevant solution because the most important decisions are made here. To handle the complexity, the design was split into manageable parts or concerns. During the guided expert discussion, we experienced that a collective assessment was gathered on the design decisions, a shared understanding was created, and the acceptance was enlarged through interiorization. Based on the theory of triangulation, the design decision of the development team could be compared with those of the expert stakeholders and the functional and structural framework could be generated and enhanced.

7.1.5. Concerning the concept integration phase using MAP

Proposition 6: *MAP offers the possibility for a rapid development of modularly configurable and presentable content. It also supports focused demonstration to stakeholder groups and their decision making process, by using abstract prototypes.*

It seemed to be necessary to pair the advantages of high fidelity prototyping with the modest cost of low-fidelity prototyping at early stages in the software development. It has been assumed that the concept of generic abstract prototyping could facilitate demonstration and early validation of software concepts with stakeholders. We assumed that modularization could increase the efficiency of demonstration to stakeholder groups with different demonstration modalities of their preferences and interests.

The modularization was found to be an effective means to achieve the articulated demonstration of software concepts and contexts. The adaptable prototype structure enhances content development flexibility, criticality needed to serve multi-focused stakeholders and break down complexity. MAP was found useful for software developers and other stakeholders to show how it will influence the real-life environment and processes. The best is if the MAP works together in the perceptive and the cognitive channels of human communication. As we experienced, modular abstract prototyping could lead to a significantly deeper and more rigorous assessment of the proposed concepts by stakeholders, and to a more consolidated feedback and enhancement proposals to designers.

7.1.6. Concerning the phase of system development using SBP

Proposition 7: *The proposed SBP methodology allows fast and cost effective tangible prototyping for functionality and usability testing, based on a composition of surrogate-software means.*

Proposition 8: *By using platform-based SBP, it is possible to go beyond the conventional concept of pure component-based design. A platform-based SBP reduces the problem of interfacing of heterogeneous components.*

In order to explore the functional discrepancies of software, it should support fast testing of the operations (functional realization, robustness and computational performance) of the software in a midterm phase of software development. The surrogates' combinations should be customized to replicate the functionality required by the final software with a high-fidelity.

As the technological trends suggest, the pure component-based software development instantiation is evolving into a platform-based instantiation, which facilitated the interfaces and data transfer between the modules (interoperability). A high level of composability and compositionality are to be achieved. Towards this end, the affordance matching of the surrogates with the functional sets must be conducted. The success of SBP also emerged through the conversion of the software view. Logical organization of the component-based flow of the system was needed to detail the functions sets and to find the best surrogates. But for the construction of the SBP, a content view was needed to represent data management concepts.

7.1.7. Concerning the validation method and the validation outcome of the Designerly Software Development Methodology

Proposition 9: *DSDM proved to be a valid methodology for the development of interactive software products that have complex functionality, user requirements that are rather uncertain and unclear in the beginning of the process, and that are complex due to environmental aspects, needed modeling and data processing.*

Proposition 10: *The adapted quadrant-based validation is a valuable approach for validation of software development methodologies. In addition, it has a large application potential and is flexible enough in single-case, reflexive, context-dependent assessments.*

There is a prevailing need to support the development of interactive software products of complex functionality, user requirements, uncertain operation processes and high level sophistication by an effective validation.

The validation square was found to be a valuable theoretical concept for the DSDM, but it had to be highly adapted to the specific validation context. The quadrant-based external validation method combines structural and performance assessment actions in both the theoretical domain and the application domain. In each quadrant, the different steps allow both qualitative and quantitative assessment according to various criteria in a reflexive manner, starting from the main structural and performance characteristics.

7.2. Personal reflections on the research done and the achieved results

The goal of this Section is to reflect upon and share my practical experiences concerning the entire research. This reflection I mainly focus on those intangibles that do not necessarily belong to the scope of scholarly conclusions, but are nevertheless important to discuss. While, the conclusions Section concentrated on the scientific reasoning and novelties, and the future research Section highlights the open issues and recommends directions for further studies in an objective manner, the personal experiences and opinions are formulated below in a subjective way.

7.2.1. Reflection 1: Return on investments

As a first reflection, I want to cast light on the trade-off issue between the efforts and benefits that was experienced during the prototyping actions. My impression has been that the efficiency of prototyping is partly determined by the achieved quality and partly by the time invested. The extra time spent on optimizing the prototype might not significantly increase the quality of the prototype, and therefore prototype optimization may in fact not be necessary. It is better to achieve a trade-off must be made between the perfection of the prototype and the time and effort spent to develop it. This is a return on investment issue. My experience has been that too large efforts and too much time are needed to fully develop a demonstrable prototype of a largely complex product. It is a hell of a job that requires multi-level abstractions and thinking.

My personal experience, which is not scientifically proven, is that the quality and comprehensiveness of the prototype not linearly correlates with the informedness of the stakeholders. Finding a tentative optimum in the trade-off is difficult to quantify. Although for me it has been proven that I achieved the balance in the reference case. We cannot identify what the enabler of achieving the balance was. If we reflect on the return on investment of the three methodologies: in the MAP, this means the time needed for making the demonstration and the quality of the demonstration. In the SBP, time needed is determined by the time needed to choose the surrogates, to optimize the surrogates, to optimize the interactions, and to code missing pieces.

7.2.2. Reflection 2: Appropriateness of prototyping from a designers perspective

As I observed, the appropriateness of prototype is determined by the representation of intended software. One of the most important considerations of using prototypes as demonstration means of to-be-developed software products is that the prototype must be an appropriate demonstration of the software concept, content, and context. I experienced that the understanding of stakeholders is crucial to receive valuable comments. It has been proven that the presentation quality has a significant impact on this. From a designers perspective of in-process prototyping, we conclude that the convincing nature of a carefully constructed technical information content can easily be destroyed by a poor structuring and implementation (presentation quality) of the prototypes, and vice versa. An exaggerating presentation may over- or under-emphasize the real technical issues and the functional quality of the presented software tool.

Therefore I experienced that the appropriateness of the various prototypes needs to be carefully pre-tested in an independent and critical manner. Arguments can be made based on our experience that testing criteria should concern the foreseeable functional and utility qualities of the demonstrated software concept, rather than the attractiveness or appeal of media used. Moreover, the criteria such as exactness, transparency, understandability, completeness, and fidelity, should be adjusted to the needs for the specific design phase. In addition, in the last phase of the development, the conceptual distance between the intended system and the prototyped system can be measured. The distance is considered low if the intended functions can be executed as prescribed in the specifications.

7.2.3. Reflection 3: Stakeholder sampling

Next to the prototyping issues, also the issue of stakeholder sampling and participation have to be discussed. Reflecting on my personal perspective, so we are not discussing the validity, but considering if I felt I got the answers I was looking for from the invited stakeholders, and if I think the people involved were the right people. This is because I experienced a difference between the ideal and the real sampling. Especially in the early development phase, when the problem was still fuzzy and ill-defined, the invited stakeholders were often talking about what they believed, not about what was asked. They were influenced by the broader context and by their daily practice. We practiced that this problem was minimized when we further detailed the design. Consequently, we cast light on the importance to reconsider stakeholders' involvement and needed sampling in each phase to select the proper people, and to clearly define the expectations and guide the stakeholders in the discussions.

Another issue that enlarged the difference between the ideal and real sampling is the stakeholders' participation. My observation is that the interaction and learning of the people is a good manner for creating a shared understanding of the software concept. And as I experienced, it avoids the need for iterations. However, conditions for participation, e.g. timing and location also caused problems to bring the right people together at the same time and place. Therefore, especially in the early phases of the development process, it is important to show the stakeholders' advantages of participating.

7.2.4. Reflection 4: Conceptualization of the multiphase methodology

I also want to reflect upon the multiphase methodology. We based on the fact that software development happens through the different defined stages. And these stages were the basis of the DSDM. I understand that due to the different characteristics of each phase, the methodology had to be projected as three single phase methodologies that each could fulfill one phase. However, we experienced no contradictions to the discrete nature of the phases and we observed that they are congruent in the entire DSDM.

The major consequence of the different natures of the phases is that continuous evolutionary prototyping was not appropriate. But the sequence of discrete stages still produce a continuous stream of software development, which forced us to realize that the transition over the phases had to be there and that data transformation from one phase to the next had to be foreseen.

7.2.5. Reflection 5: Multi-framing research approach

We decided to split the research into five research cycles, to manage the complexity of the research problem. I could reflect very positive upon this decision, as my impression has been that it allowed us to deal with each phase in detail and to be able to develop the single phase methodology components separate. Besides, we could also discuss the entire DSDM on a different level. I also recommend the applied DIR framing methodology. For me it has been proven that the use of design skills and methods allowed us to generate knowledge that we could not generate otherwise, and that could be used to justify and improve the methodology.

7.2.6. Reflection 6: Use of a reference case

In this research we used a practice-oriented approach. The overall research approach has a dual flavor. It comes from the fact that we wanted not only to synthesize a DSDM but also to apply it to a practical case in order to support immediate learning and optimization for consequences. Consequently this looping of simultaneous development of the methodology and a reference case, is observable is almost every research cycle. The reference case was developed to be useful and reliable research means. The following advantages were observed: (i) the reference case was used to monitor the methodological elements in practice, (ii) from the process of applying the methodology to the reference case, we learned from the interaction between the methodology process, methods and techniques and the practical situation, and this helped a lot to improve the methodology, (iii) using a reference case that was simultaneous developed gave a direct testing opportunity to improve and detail the methodology, (iii) the simultaneous development allowed an evolutionary development over the different phases, and (iv) the combination of theoretical reasoning and practical implementation resulted into a better description of the methodologies. The reference case was not intended to be a final product, and consequently it was not developed so far. It was only developed into the manifestations needed for the particular stages. Regarding the reference case, my impression has been that it was a correctly selected interactive and knowledge intensive sample. By applying the proposed phase methodologies, the case could be developed to the needed level. For me it has been proven that the case was a good reference means.

7.3. Recommendations for future research

Finally, in this Section we end the thesis with some ideas and possible directions for further research. These recommendations are based on following objectives: (i) identification and elimination of knowledge gaps, (i) making it clear what uncertainties are more robust, and (iii) extending the research beyond its current limits. This knowledge identification is carried out for the different aspects of the research: (i) on the level of the DSDM methodology, (ii) on the level of the single phase methodologies, and (ii) on the level of the research approach. A visualization of our reasoning system is shown in Figure 7.1. Because it was not

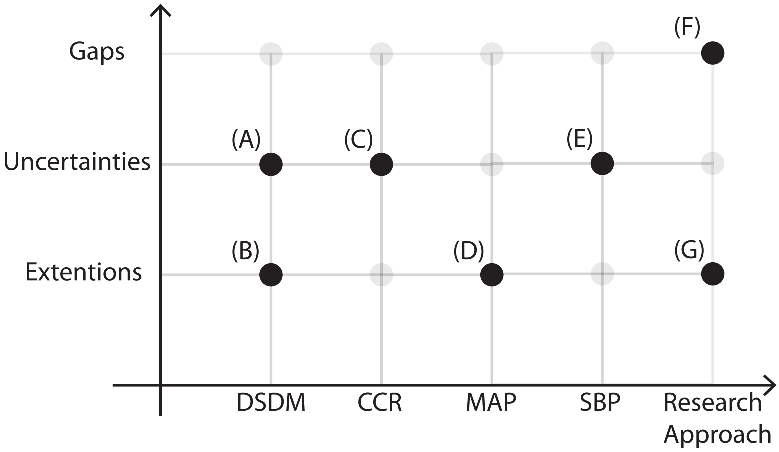


Figure 7.1. Reasoning model for future research recommendations

possible to identify recommendations for all aspects, we indicated in the Figure those that could be discussed.

7.3.1. Regarding uncertainties in DSDM (A)

The DSDM is only focusing on the three phases defined before by the three single-phase methodologies. But there might be a need for a fourth single-phase methodology. Further investigation of the other development phases is needed to identify if there is a need.

7.3.2. Regarding extensions to DSDM (B)

On the general level of the DSDM methodology, we would recommend: (i) to investigate in which contexts, apart from those investigated, the DSDM might be useful, and (ii) to explore the possibilities to extend the methodology to intangible product development in general: development of product-service combinations, services, systems and experiences. On the level of developing complex systems, we would recommend to further investigate if the same or a similar approach can also be used for the development of the non-software parts of the system, and how the full development must be managed by designers.

7.3.3. Regarding uncertainties in CCR (C)

Related to the development of the CCR methodology, future research is needed to investigate how the interiorization of stakeholders can be increased. This issue was not investigated since it emerged as a finding in the experiment of CCR application.

7.3.4. Regarding extensions to MAP (D)

Related to the MAP methodology, future research is needed with a view to identify fully-interactive prototyping. This fully-interactive prototyping is hard to achieve but would open a new world of possibilities in early prototyping. To achieve it, we recommend further research on technology realizations, and on practical implementation and behavior level.

7.3.5. Regarding gaps in SBP (E)

To ease the execution of SBP, the identification of surrogates must be explored and supported. During the experiment with the reference case, we experienced that due to the large, almost unlimited amount of possible surrogates, it is impossible to have an overview to simplify the decision making process.

7.3.6. Regarding uncertainties in SBP (F)

In addition, during the experiment we also experienced that it is hard to find the best surrogate-combination. Future research should be performed in this direction to support developers and SBP builders as well.

7.3.7. Regarding gaps in the research approach (G)

In the research we choose to have a single reference case that represents a family of cases. We were pleased with the results, but we would recommend a comparative study to closely compare the advantages and disadvantages of DSDM compared to traditional or agile development methods.

7.3.8. Regarding extensions to the research approach (H)

The complete development of the reference case was not the focus of the research. If it should be completely developed, following actions should be taken: (i) realization of the software tool, i.e. programming and coding of software, (ii) sales and distribution, and (iii) maintenance. As it was not in the scope, we did not develop the complete software tool, because additional research actions were needed to explore knowledge, e.g. on what should be the contents of the knowledge base. These actions could not be carried out without distraction from the main focus of the research, which was the development of the DSDM. Other issues in the software development that need to be resolved to make the tool fully operational include: incorporation of semantic search capabilities, enabling automatic content generation, and dealing with the ontology problem comprehensively.

List of abbreviations

ABR	Analogy-based Reasoning
AP	Abstract Prototyping
APM	Abstract Prototyping Module
ASD	Agile Software Development
ATC	Auxiliary Technical Concept
CB	Component-based
CBD	Component-based Design
CBR	Case-based Reasoning
CCR	Critical Collective Reflection
CO SD	Component-Oriented Software Development
CPS	Cyber-Physical Systems
DIR	Design Inclusive Research
DN	Demonstration content
DSDM	Designerly software development methodology
DX	Demonstration Context
FGS	Focus Group Session
GAP	Generic Abstract Prototyping
HCD	Human-Centered Design
HCI	Human-Computer interaction
IAP	Interactive Abstract Prototyping
KE	Knowledge Engineering
MAP	Modular Abstract Prototyping
PBR	Probability-Based Reasoning
PD	Product Development
PX	Presentation Context
QEBV	Quadrant-based External Validation
RC	Research Cycle
RE	Requirements Engineering
RIDC	Research In Design Context
RTC	Representative technical concept
SA	Software Architecture
SBP	Surrogate-based prototyping
SD	Software Development
SEM	Software engineering methodology
SH	Stakeholders
TC	Technical Concept
TO	Trade-off
UCD	User-Centered Design
UT	Ubiquitous Technologies
UX	User Experience
VS	Validation Square

List of Figures

Chapter 1.

Chapter 2.

Chapter 3.

Chapter 4.

Chapter 5.

Chapter 6.

Chapter 7.

List of tables

Chapter 1.

Chapter 2.

Chapter 3.

Chapter 4.

Chapter 5.

Chapter 6.

Summary

A designerly methodology for software development

Background to research: Trends influencing industrial product development

The knowledge platform of this promotion research is industrial product development that has been strongly influenced by four major trends in recent decades. These trends are: (i) diversification of human and social needs, (ii) rapid development and uncontrolled proliferation of advanced technologies, (iii) evolution of product manifestations and implementations, and (iv) increasingly sophisticated design approaches and methodologies. Together, these trends have led to a characteristic shift in approaches to product realization.

The first trend is striving towards an increased sense of well-being. As a result of this trend, society has moved closer to a consumer-oriented economy with continuously changing human needs. The different kinds of human needs and the relationships between them have been described by Maslow's model of the hierarchy of needs. Represented as a pyramid, this model includes five hierarchically arranged levels that build on each other. At the bottom of the hierarchy are the physiological needs, which are complemented by the safety, social, and esteem needs. The top-level expresses the self-actualization needs. Lower level needs must be fulfilled first, in order to meet those at the higher level.. When addressing the higher level needs, designers usually face complex design tasks, particularly when this challenge must be addressed in ever-changing situations. For these reasons, designing products for the satisfaction of the higher-level needs entails new ways of thinking, novel strategies, and innovative concepts.

The second trend is an accelerated technological evolution and diversification. Evidence of this can be seen in many forms, for instance, it is reflected in the rapid developments in the electronics industry. Electrification started some 150 years ago. Some 90 years ago, complex electronic controller solutions had already been developed in the military industry. Seventy years ago, the concept of digital computing was introduced and rapidly proliferated in the industry. Since the 1960s the emerging computer technologies have been complemented by advanced software and information processing technologies, advanced material technologies, and energy provisioning technologies. As a result, technological evolution has gained an even larger momentum. Technological evolution has been highly influential on both the manifestation and the realization of the product. The nature of modern products is no longer determined by their hardware parts, but often by their embedded software components and knowledge bases. The range of physical hardware components has been extended with processors, sensors, actuators and transformer components. Not only have the variety and functionality of hardware components have changed drastically, but also their scale and integration level. Miniaturization has been the dominant force of change, leading to micro- and nano-scale components. The software technologies have also been rapidly developing, both in terms of their enabling algorithms and their information/ knowledge contents.

The third trend is the observable evolution of product manifestations and implementations. This trend is fuelled by the above developments of technologies that have contributed to the formation of new product paradigms, which in turn lend themselves to completely different material, energy, and information flows in modern products, such as cyber-physical consumer durables and services. The emerging new product paradigms not only imply a sophistication of products, but also change their meaning. According to the current knowledge, three generations of products can be identified. They are visualized in Figure 1. The first-generation products are assembled hardware products, software implementations, and pure services. The second-generation products show a growing level of integration and move towards integrated systems. They are substantiated by three types of systems: (i) product-service systems, (ii) embedded systems, and (iii) information systems. The third-generation products are complex systems and environments, which are conceptualized and implemented according to the principles of cyber-physical systems.

Therefore, they are often referred to as cyber-physical consumer durables and services. Either they involve a high level of interaction with people and the environment in which they are embedded and other products, or they may be more autonomous and adaptable. Other infrastructural systems with resembling functionality and implementation technologies are called ‘the Internet of things’ or self-contained systems as ‘complex adaptive systems’, but these names also indicate some notional and conceptual differences. The third generation high-end products are highly complex, decentralized, open, adaptable, intelligent, or even

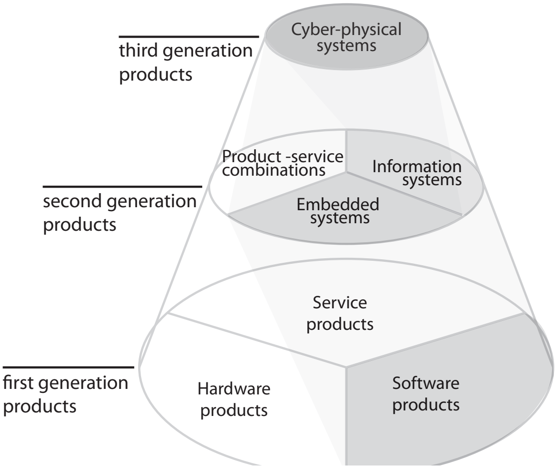


Figure 1. Evolution of products and the growing importance of the domain of software development

evolving implementations of customer durables. Typical features of these cyber-physical systems are the strong multidisciplinary functions and the ability to penetrate human social and cognitive domains. These systems brought different forms of human-system interactions to the foreground. For completeness, we must acknowledge that a new generation of products does not fully replace the older generations, but coexist with them. It means that, at a given point in time, different generations of products can be seen on the market.

Altogether, the three above-mentioned trends lend themselves to a fourth trend. This concerns the changes in design approaches, methodologies and technologies. These are becoming more sophisticated due to the expansion of opportunities (the affordances of technologies) and to the demands of fulfilling new customer needs and increased user-expectations. This fourth trend is also reflected in the shift of designers' focus. Namely, their attention is moving away from pure form, function, materials and manner of production concerns and towards utility, usability, human experience and desirability concerns. This means that, in addition to form, function and materialization, the meaning of the products is also becoming an important phenomenon for designers. The shift in design approaches from function-focused ones through consumer-oriented ones to human-centered ones is an essential development. This provided the starting knowledge platform and motivation for this promotion research, and played an important role in conceptualization of the research problem.

We observed that the progression indicated by the above trends may be blended with Maslow's model of human needs. Therefore, we have extended the coverage of this model with the changes discussed above in the focus of designing and with the varying design approaches. Shown in Figure 2, this compound conceptual model expresses all of the concerns that industrial design engineers should address when designing competitive products. The objective of this promotion research is to address the issues of the top layer of the extended Maslow model, which incorporates the needs for self-actualization, desirability and human-centered approach.

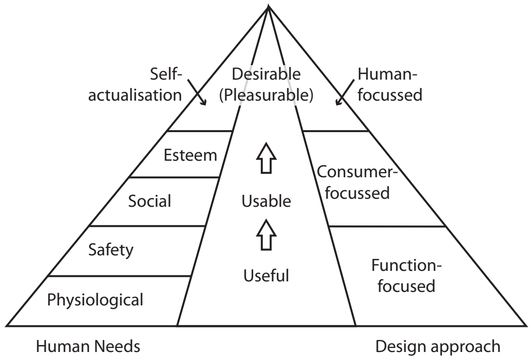


Figure 2. A conceptual model on the current concerns of industrial design engineering

Research domain and problem

In this research project, we focus on knowledge exploration and synthesis for methodology development. The main research problem addressed here is that of conceptualization and implementation of a designerly methodology to support realization of design software tools by which designers and engineers can develop complex second and third generation products. As the interacting trends imply, some specific requirements should be considered. For instance, the methodology has to be: (i) procedurally structured, (ii) human-centered, (iii) adapted to the designers' mindsets, and (iv) supporting the designers with relevant specific methods, instruments and techniques. The close relationship between products and their users creates the need for a more intense stakeholder involvement during the design process. This enables designers and engineers to understand the needs for change and to cope with the challenges of complex functionalities and fast realization processes in a context-dependent manner. The novelty of the reported research approach is in the epistemological, methodological and procedural symbiosis of the methodology development and the application case development. It was assumed and has been confirmed that the dialectic interaction of the support tool and the application case provides benefits for both. As the literature shows, several generic methodologies have in the past been developed without considering the specifics of concrete applications. However, we presumed that, in case of a concurrent elaboration, we have the opportunity to implement a kind of 'reflexive practice', or, in other words, to follow an approach that allows for the fine tuning of the methodology to representative applications and the achievement of efficiency through practical experiences. However, as explained below, only one complex reference case could be developed in this promotion research project due to capacity limitations.

In this research we have concentrated on the development of software as products or components of complex systems. This problem domain was chosen because software (i) yields the largest opportunities to meet the requirements rooted in complexity and evolution, and (ii) has a large influence on the sophistication of products, but (iii) is also the most difficult part to develop in complex systems (see Figure 1). In the last decades, there was an intense diversification of software products and continuation of this process can still be observed. Software products manifest as self-contained application packages, but also as embedded software for controlling systems, or agents of complex information systems, or synergetic constituents of cyber-physical systems.

For this research, the type of software we specifically focused on was defined by acknowledging a real-life need, namely, that of an interactive and knowledge-intensive design support tool able to facilitate concept generation and trade-off forecasting in case of ubiquitous augmentation of domestic appliances. A software tool providing the necessary functionality for this reason was selected as a test case for our human-centered software development methodology. This test case also took the role of an archetype of a family of similar design support tools. By using it as a reference case, we could consider a family of design support tools in our work and grasp a range of technical and human issues associated with the dedicated software development methodology.

Interactive software applications (in particular, application-focused design software tools), are used by designers who expect the software tool (i) to support their thinking and creation processes, (ii) to allow for greater freedom in the conceptualization and investigation of solution concepts, and (iii) to process dynamically changing real-time data, while (iv) also allowing easy and effective interaction and data/knowledge retrieval and management. As a consequence of these expectations, the development of this family of software tools needs an intense stakeholder involvement.

However, in the current software engineering practice, the importance of focusing on all stakeholders did not receive sufficient attention and emphasis. We discovered that the two most wide-spread approaches currently being practiced, namely the traditional development methodology and the agile development methodology, are not sufficiently human-centered. The traditional human-centered design methods do not go beyond typical customer research. They consider the overall utility, and typically the design and evaluation of the user interfaces, rather than the specific details of harmonizing operation processes of the software tool with the thinking processes of household appliance designers. Even the currently known agile methodologies overlook the stakeholders/end-users needs and the user experience and satisfaction aspects. Equally important is the fact that these approaches are not suitable for the development of second and third generation complex software products or systems.

Our forerunning investigations explored that the need for human-centeredness in software design processes emerged from different perspectives: (i) team-perspective (dealing with (large) multi-disciplinary teams), (ii) process-perspective (intense stakeholder involvement), and (iii) product-perspective (design of software is inseparable from the design of human activities). The handling of these perspectives individually and in combination may be challenging and complicated as they usually involve many different types of stakeholders, such as end-users, suppliers, clients, marketeers, management, knowledge experts, IT maintenance, and so on, who are involved in different phases (specification, algorithm development, coding and production, distribution, usage, maintenance, etc.) of the product life-cycle, and context of the system. On the other hand there are no general rules, since the need for stakeholder involvement always depends on the concrete cases.

Research vision and objectives

This PhD research strongly envisioned that software tools belonging to the category of interactive application software (e.g. design support tools) should be developed according to a participatory design strategy. Our observation was that the human-centered design of interactive software has not reached the desired and potential level, compared with many hardware products. The research vision was that a designerly methodology was needed in order to solve the above-mentioned problem of interactive software development. As a research problem this poses two challenges: (i) reconceptualization of the development process of interactive software towards a designerly (stakeholder-oriented) approach, and (ii) establishing a robust basis for a new methodology that covers the early phases of software development where critical decisions are made. Our primary objective was not increasing the efficiency of the product development, but increasing the utility and quality of interactive software products. By involving the stakeholders in the early phases, software products can be more custom-made and more needs' appropriate. Despite the extra time and efforts, it is worth involving stakeholders in the utility and quality enhancement of software. Obviously, they must be involved in the most critical points of the process, and in order to achieve a significant impact, some reconceptualization of the process is deemed to be necessary. As we know, the most critical decision points are in the fuzzy front end and in the conceptualization phases of software design, though those decisions that are typically made during the implementation phase cannot be neglected. Considering these facts, we hypothesized that a combination of different single-phase methodologies are needed to provide effective support to every particular phase of software development.

Consequently, the objective of the PhD research was set to conceptualize, elaborate and test a designerly software development methodology (DSDM) that supports stakeholder involvement in the most critical phases of software design. We decided to apply a structured view on the software development process and introduced a methodological framing by which we could focus on the subsequent phases. Stakeholder involvement has to start during the identification process of the design requirements and when an overall conceptual framework of the software tool is constructed. Stakeholders should also be involved when the concept of the software tool has been developed (it should be demonstrated to stakeholders and justified and validated through their involvement). Finally, stakeholders should be involved when a pre-implementation version is completed and take part in the testing and critique of this. To complete these activities efficiently, the above phases need dedicated methodologies that we called single-phase (component) methodologies. They were coherently and transitively integrated into the targeted multi-phase support methodology, called DSDM.

Research hypothesis and assumptions

Focusing on humans and their experiences is a key-issue in current product development. Our generic research hypothesis suggests that software development could benefit from following the principles of human-centeredness that are applied in traditional product development. Based on our previous literature study and practical experiences, we investigated the differences between the development of hardware and software products. Furthermore, we investigated why we cannot directly use the human-centered design principles of consumer durables to software development. Our generic research hypothesis claims that specific methodological principles gathered from the domain of modern consumer durable development could be used as a basis of the targeted designerly software development methodology. A graphical illustration of our hypothesis is shown in Figure 3. It has a broader relevance than the area of interactive design support tools alone – its claims can in principle be extended to the domain of cyber-physical systems.

The above hypothesis rests on the assumption that traditional methods of consumer durables design offers useful design principles and that they can be taken over to the development of interactive software products. It is possible to defend these assumptions, because there is extensive literature on the principles and approaches of human centeredness in consumer durables design, where optimal physical and cognitive interaction with humans is an important factor of product success. In this domain, designers have a firm intention to customize the product to end users and, towards this end; they closely involve and interact with various stakeholders in the development process. The stakeholder involvement is supported by the use of various demonstration means, visuals, and virtual and physical prototypes, such as sketches, mock ups, CAD models, and tangible prototypes. In the design process, prototypes are used to discuss and evaluate the design with stakeholders towards improvements. Verifications and validations occur at different phases of the development process and consequently different means are used. Incorporating the relevant principles of consumer durables development into the domain of interactive design software development is, however, not straightforward. There are important differences between the two domains. The most significant ones are: (i) the difference in the tangibility or material manifestations of products, which entail different prototyping means, and (ii) the difference in the interaction with the physical product and software products. It seems that it is more difficult to actualize early demonstrations of intangible products and consequently, they require a higher level of empathy from stakeholders with the design, as well as an ability to provide suggestions for improvements.

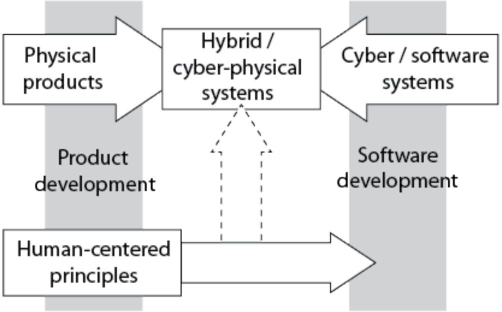


Figure 3. ***Visualization of the main hypothesis***

Overall research approach

Due to the variety of objectives and contexts, a multi-methodological framing was applied to set up the research design. The whole of the PhD research was broken down into five interrelated research cycles (RC x), as shown in Figure 4. Each cycle had its own objectives, context, and framing methodology. For this purpose, the methodological framing theory, proposed by Horváth, has been applied. The objective of the research cycles was to investigate the needs of the specific phases of the idealized multi-phase process (framework and requirement aggregation, concept development, and system elaboration). The investigation took place from the perspective of designers, with the aim of converting/applying the principles to software development, and to develop and test a practical single-phase stakeholder-sensitive methodology for each phase. In the first research cycle we investigated the need for stakeholder involvement in the current software development approaches, and described the context of the research process. We analyzed the phenomena of stakeholder-oriented design, and considered the gaps and important issues to deal with in our methodology. During the execution of RC 2, 3 and 4, we investigated the three most critical phases discussed above. In research cycle 2 we examined the methodology development issue in the context of requirements engineering and framework ideation.

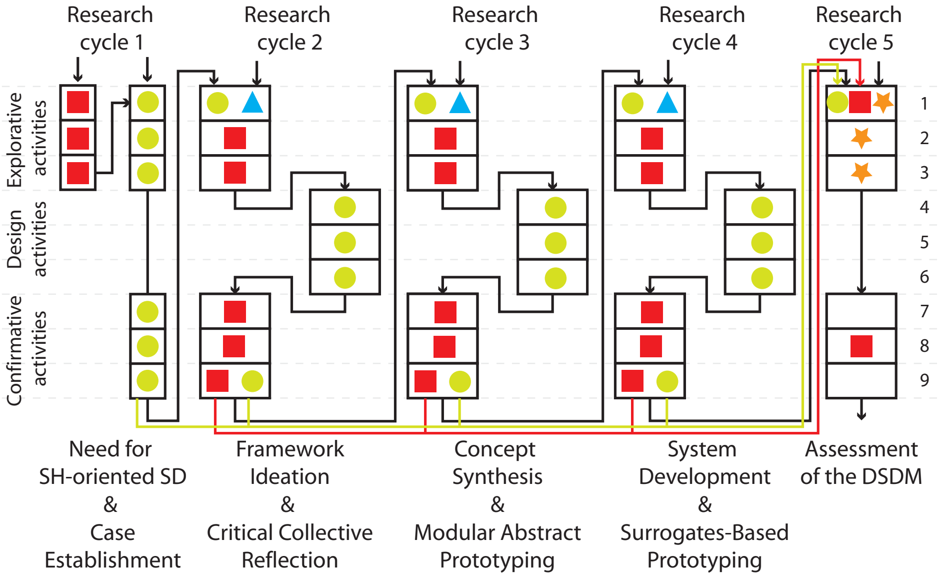

Research cycle 1
Research cycle 2
Research cycle 3
Research cycle 4
Research cycle 5
Explorative activities
Design activities
Confirmative activities
1
2
3
4
5
6
7
8
9
Need for SH-oriented SD & Case Establishment
Framework Ideation & Critical Collective Reflection
Concept Synthesis & Modular Abstract Prototyping
System Development & Surrogates-Based Prototyping
Assessment of the DSDM

Figure 4. Methodological framing of the research

In research cycle 3, the context and the influencing factors of enabling concept synthesis and demonstration were investigated. In research cycle 4, the research work focused on surrogate-based prototyping in the context of detailing functionality and usability testing. In research cycle 5, we drew conclusions about the entire research through a multi-aspect external validation of the proposed multi-phase methodology.

In order to support the execution of research cycles 2, 3 and 4, the framing methodology of design inclusive research was applied. In these research cycles, various implementations of the reference tool were used as dedicated research means. The framing applied provided sufficient methodological support for each of the phases and facilitated the testing and validation of the conducted research actions and the findings, respectively. In research cycles 1 and 5, a higher-level abstraction was applied because the focus of these cycles was on the multi-phase methodology, rather than on the single-phase methodologies. In the case of these two cycles, research in design context was used as methodological framing. The reason behind this decision was that we investigated phenomena closely related to design in specific contexts. In the schematic overview of the complete research approach, shown in Figure 4, the symbols refer to the knowledge generated during the research activities. Knowledge was generated concerning the whole of the targeted DSDM (and its component methodologies), related to the issues of the specific development phases, related to the reference case, and related to the required validation method.

Constraints on the research conduct

At the discussion of the research domain and problem, we argued about the necessity of developing a reference application case parallel with the multi-phase software development methodology. The primary reason was that, at the time of developing an execution plan for the research project, we realized that a methodology development cannot be separated from the definition of the family of application cases that it is intended to support. The practical advantages of considering some concrete reference case(s) from the very beginning of the designerly software development methodology were also considered. Consequently, our decision has been to elaborate and learn from a reference case parallel with the conceptualization and implementation of the multi-phase methodology. The co-development of the methodology and the reference case resulted in a co-evolution during the research process. Ideally multiple cases should have been developed and investigated, but due to time and capacity limitations, we had to compromise on conducting a single-case study. On the other hand, in defining this particular reference case, we had in mind that it should be a representative of a family of relevant application cases. We believe that this traversal (intertwined) development of the DSDM with the reference case did not impose strict limitations on the results obtained. At the same time, it introduced a conceptual novelty in the conduct of the research. This novelty came from the fact that the DSDM, coupled with the reference case, could be used as an evolving research means in the research cycles, which were framed as design inclusive research.

Typical examples of highly interactive software applications are the various implementations of design software tools, such as CAD, CAE, DFX, etc. These software products are strongly contextualized and process-related, in order to support designers efficiently. The success of these products is hidden in the fact that they are adapted or adjusted to designers' mindsets, and that they are supposed to fit their natural working process. The selected reference case is a software tool for smart energy saving using ubiquitous controllers. This case was selected because in order to be able to support the software development process, stakeholder involvement was crucial. The aim of the tool is to support the designers not only in their modeling and analysis process, but also in their decision making process, by offering them structural information and trade-off calculations. Thus the conceptual basis of the software tool is not just a composition of algorithms, but the decision-making process and mental reasoning of designers. The highly interactive nature of the considered design tool required a high amount of action-related and decision-making knowledge. An optimal development of this kind of software tool projects ahead the need for participatory conceptualization and design, in which the end-user (designer) is not the only stakeholder. Software developers and administrators of the software, as well as the various knowledge engineers involved (such as energy saving experts and controller device suppliers) should be involved in the software development process.

The essence of the designerly software development methodology

Implementation of the working hypothesis led us to a theory that explained what kind of designerly software development methodology could support stakeholder involvement in the most critical phases of software design, and how this methodology could be applied with success. The *objective of the DSDM* is to systematize and facilitate the involvement of the stakeholders relevant to each phase of the development. DSDM was developed to support the implementation of second and third generation software products, in particular design support tools. These interactive software tools feature functional and structural complexity and the need for an ability to evolve in order to continue fulfilling the needs of their stakeholders. The practical aim of the proposed methodology is to achieve an optimal stakeholder involvement with the hope of increasing the efficiency and effectiveness of conceptualization and detail design.

Based on literature, we learned that a methodology should be specified by an underpinning theory and should offer procedural scenarios, problem solving instruments, a set of methods, and should define the criteria of goodness. The underpinning theory of DSDM formulates three conceptual pillars: (i) context-sensitive stakeholder involvement, (ii) managing complexity and evolvability, and (iii) achieving an increasing level of fidelity. In the application of the methodology, a key issue is to obtain constructive feedback from the stakeholders, including qualitative change and improvement proposals and/or quantitative measures. To handle the complexities accompanying comprehensive systems, we build on the principle of separation of concerns. The complexity of the targeted products also implied that it was not possible to consider all possible requirements, and the characteristics of the different stakeholders could not be known at the start of the development process. The outcome of the application of the component methodologies included in the DSDM operates with an increasing level of fidelity. This level depends on the amount of information available in the above three phases. The methodology we have developed suggests, starting with a high-level abstraction (that can be embedded in a low-fidelity prototype) and ending with a high-fidelity testable prototype of the detailed software system. These subsequent forms of prototypes can be adjusted to the contents, stakeholders and contexts. In the process of exploring the stakeholders' opinion, ideas, and recommendations, the prototypes of a growing fidelity support both the interrogation and the constructive activities.

As mentioned above, the DSDM focuses on three critical phases of the software development process: (i) framework ideation, (ii) concept integration, and (iii) system development. Because of the diversity of the stages of the development process, different phase-methodologies have been proposed for each individual phase. These single-phase methodologies are as follows: (i) critical collective reflection (CCR) that supports stakeholders' reflections on the requirements and the conceptual framework. By our reasoning, contrasting the proposals of the development team with that of the expert stakeholders contributes to achieving better framework solutions. (ii) Modular abstract prototyping (MAP), which supports concept integration and consolidation. The proposed modular approach allows for the consideration of the differences in interests and viewpoints of the stakeholders and to demonstrate the software concept according to their needs. The discussion enabled by MAP will result in necessary change proposals and more consistent software concepts. (iii) Surrogate-based prototyping (SBP) that supports fast realization and investigation of testable, tangible prototypes, which are developed by using existing software components or platforms. Based on SBP, not only the functionality, but also the usability of the software products can be tested rapidly and at a low cost. As demonstration and testing means, SBPs facilitate the provision of concrete low-level feedback on the attributes and behavior of the software before its final realization. The growth of fidelity of the prototyping increases over the three phases of software development as shown in Figure 5.

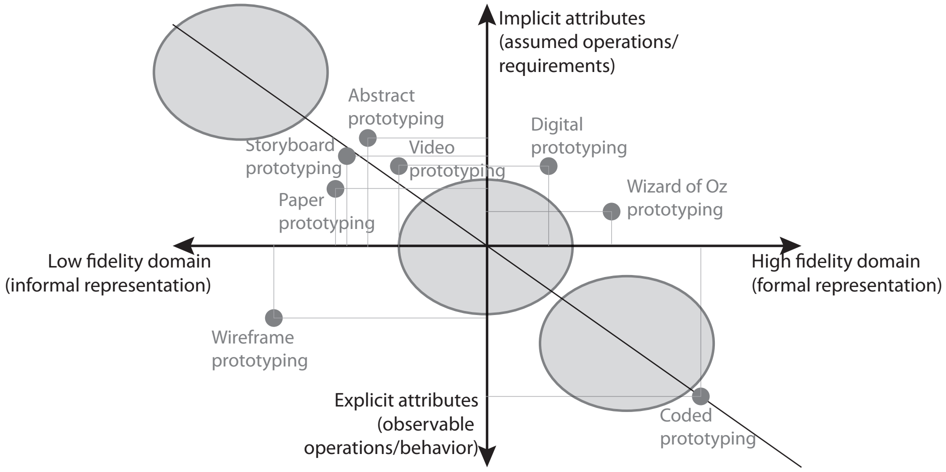


Figure 5. ***Evolution of the fidelity of prototypes in the three component methodologies***

Phase 1: Framework Ideation

In the framework ideation phase, a methodology was needed that supports: (i) blending the knowledge of multiple domains into a consistent body of knowledge, and (ii) developing a system-level understanding and a conceptual framework of an abstract solution. The framework ideation phase is characterized by its transition from a problem description (manifested as requirements), into an abstract solution that appears as a functional and structural framework. Typical in this phase is the problem of incomplete context knowledge, and ill-defined and conflicting ideas. Stakeholders should be involved in the discussions concerning the expectations, needs and goals of the software and the critical design decisions. We developed a novel methodology which is called Critical Collective Reflection (CCR). The CCR methodology is based on the principle of triangulation. It aims to compare the results of the development team with those offered by expert stakeholders. By obtaining feedback from industrial experts on the underpinning design decisions and the proposed manifestation of the conceptual framework, a shared understanding could be created, and the functional framework can be enhanced.

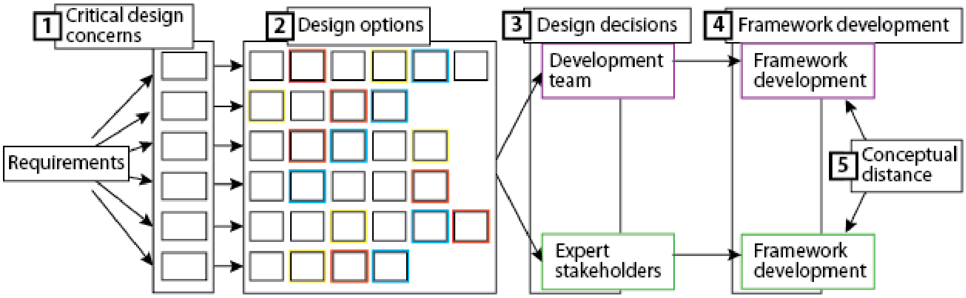


Figure 6. Process of framework ideation using CCR

With regards to the process of CCR, since it is essential, we have to highlight the fact that before modeling the requirements during the problem definition, a sufficiently deep investigation was conducted regarding the related, contrasting knowledge domains and the context of the problem. The process of CCR is divided into five steps, as shown in Figure 6. To handle complexity of software development, different design concerns were identified and the targeted software was decomposed into manageable parts. A design concern is the decomposition into parts for which design options can be found. By linking the relevant requirements, and grouping them into specific design concerns, it proved to be easier to find a solution for them. As a next step, design options should be found. The method of morphological analysis was used to find and map solution options in a visual graph that facilitates an overview. This overview helps making design decisions on the level of the individual design concerns and combining the solutions of the different design concerns in an overall concept. Design decisions are to be made by the development team as well as by the expert stakeholders. The development team should convert its ideas into a first conceptual framework of the intended software. Through the guided expert discussion, a collective assessment can be carried out, taking into consideration the different design decisions required. Based on the information received from the expert assessment session, an enhanced framework is built. As an indication and measure of concept validity of the framework, the conceptual distance among the features of the two versions (original and improved) of the framework is examined formally. The concept of semantic distance is used to express similarity or difference.

For the CCR of the reference case, we identified ten important design concerns for which design options should be generated and decisions should be made. As the development team, we considered these concerns and developed a framework containing all design decisions. In addition, questions were constructed to discuss the design concerns with stakeholders. Expert stakeholders were selected using stratified sampling, taking into account the real-life stakeholders and their interaction with the software. This stakeholder identification was a first impression of the possible relevant experts because it is not yet possible to identify them precisely. The stakeholders participated in the focus group sessions of CCR. Though we came up with a description of the ideal type of stakeholders involved, it proved to be difficult to find them in real-life, therefore it was necessary to compromise. The results of the focus group session were analyzed and converted into an enhanced functional framework. In order to compare the new framework with that of the development team, we measured the conceptual distance. Regarding the application and effects of CCR, the overall experiences were very positive. However, we recognized the following issues: (i) the high level abstraction enabled effective complexity handling, but it was important to draw very clear boundaries, (ii) the visualization and evaluation of the effect of the decisions was almost impossible during the session, and it had to be done in a post-event form, and (iii) the aim of getting a collective opinion and assessment was difficult because the stakeholders conceptualized the problems differently and did not give sufficient attention to the context of the other members' reasoning.

Phase 2: Concept integration

In the concept integration (or concept synthesis) phase, the in-development-software exists as a functional and/or procedural concept. To overcome the lack of proper modeling, simulation and demonstration means, an early software prototyping methodology was required to support testing with multiple stakeholders. This prototyping approach was intended to deal with the incomplete and vague aspects of the available information. For this purpose, we developed a novel methodology called Modular Abstract Prototyping (MAP). MAP supports the development of rich and complete medium-fidelity prototypes of the software concept in order to aggregate stakeholders' feedback-through-demonstration as of the earliest phase of software development. MAP includes a real life demonstration of all characteristic operations and interaction/use processes, including the operation of the software concept, the actions of the human actors, and what happens in the surrounding environment. From an information technological point of view, a MAP prototype is a hierarchically organized information structure, entailing modular content dissection. This way, MAP has the flexibility to collect feedback from various stakeholders and demonstration sessions. The information structure of the software concept to be demonstrated can semantically be broken down into complementing information sub-structures that are encapsulated in the difference modules of the MAP.

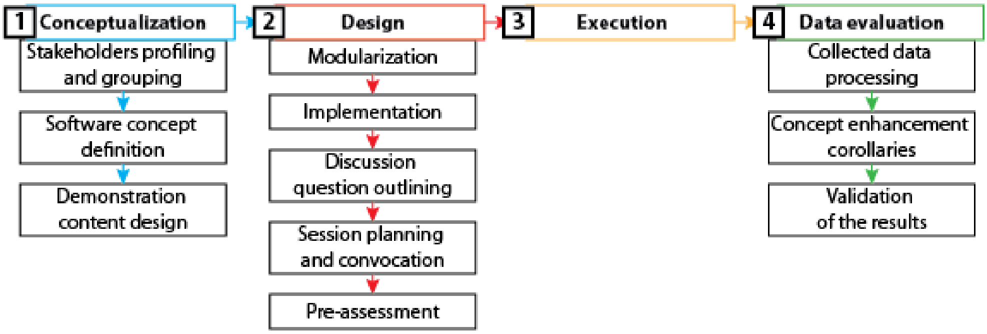

1 Conceptualization
Stakeholders profiling and grouping
Software concept definition
Demonstration content design
2 Design
Modularization
Implementation
Discussion question outlining
Session planning and convocation
Pre-assessment
3 Execution
4 Data evaluation
Collected data processing
Concept enhancement corollaries
Validation of the results

Figure 7. Process of MAP towards testing concept integration

The whole execution process of MAP can be split into four steps, as visualized in Figure 7: (i) conceptualization: considering the technical information required to build the software‘s prototype, (ii) detail design of the prototype: an important issue regarding this is to find the optimal number of modules (i.e. the AP resolution), which is related to the number of different stakeholder, their modalities and interests. A well-chosen combination of modules provides adequate information for stakeholders in a cognitively controlled way. (iii) Having the modular APs, the entire demonstration session can be broken-up into a series of sub-sessions, in which dedicated MAP contents are presented to different stakeholders in focus group sessions per stakeholder. This way, information overload of the stakeholders is reduced, and the development of demonstration material is more flexible and efficient. Lastly, (iv) the data evaluation is completed by transcribing and organizing stakeholders’ feedback into a concrete conclusion document for concept improvement.

We applied the MAP methodology in the reference case. To start the MAP process, we identified three types of stakeholders, each having their own characteristics and interests: (i) product designers, (future end-users of the design support tool); (ii) software developers (programmers of the application); and (iii) knowledge engineers, for knowledge processing about previous cases (i.e. the database with knowledge on past cases). We built six prototype modules, using visuals and animated movies to represent the narration, using specific enactments, depending on the content of the modules. In Figure 8, screenshots of these modules are shown so as to give an impression of the implementation of MAP. To gauge the reactions of stakeholders, we used passive (observant) stakeholder participation in focus group sessions. In the execution of the confirmative assessment of the concept, per profile, focus group sessions were organized, hosting relatively small groups of people (6-12 participants). In total, we organized four sessions with product designers, two sessions with software developers, and two sessions with knowledge engineers. Data evaluation of the outcome of the focus group was conducted using semantic coding.

After analyzing the impact and frequency of the various aspects, we could identify the operation field and evaluate the most important change proposals. Based on the appropriate proposals, an improved concept could be built. It took a relatively large amount of time and effort to build the entire prototype. Nevertheless, we found that it was an effective means to: (i) detail the software concept for the development team, and (ii) to support the harvesting of feedback from the stakeholders. In general, a significant amount of data could be generated during focus group sessions for the case study that was demonstrated. The MAP clearly displayed its effective role as a stakeholder-tailored, information-rich demonstration means.

Phase 3: System development

The third critical phase in the software development is the system development, which is also called detailed design. This raises the need for an intermediate (high-fidelity) prototyping approach, which offers a testable implementation that lies between abstract prototyping

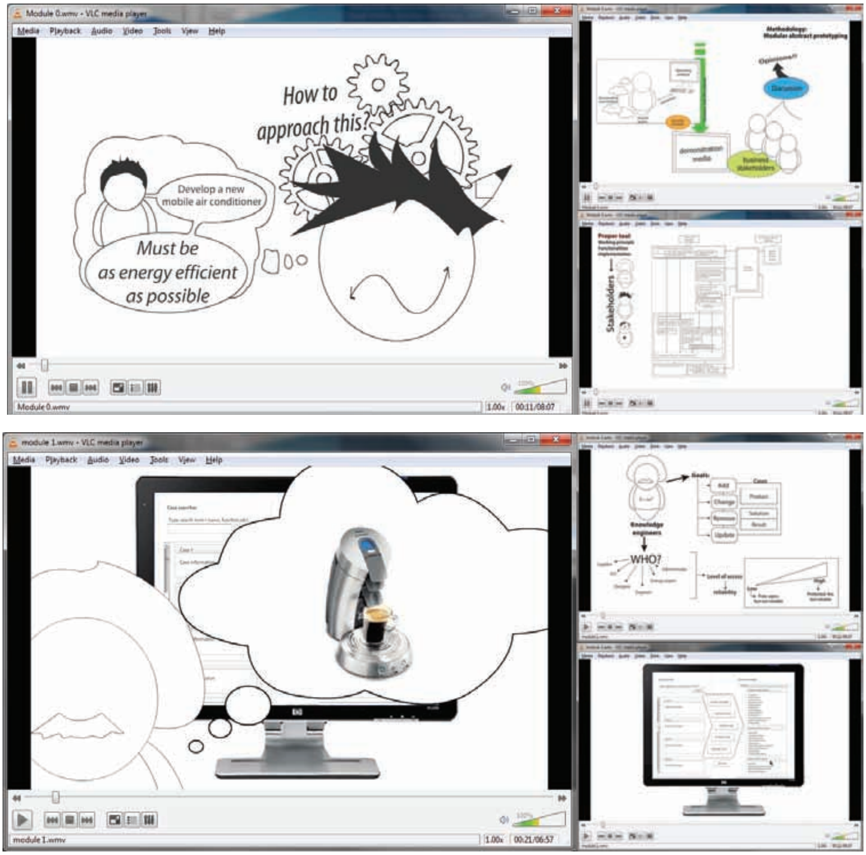


Figure 8. Screenshots from the MAP prototype modules in case

and fully developed software. This intermediate implementation is intended to be realized relatively quickly and with low costs and to be usable for functionality and usability system testing. We assumed that the growing number of tools, modules and components available could be used to enable a rapid testable prototype development and to reduce the need for functional or structural modifications at the end of the development process. Surrogate-based prototyping (SBP), as its name indicates, is a novel methodology which facilitates compositional software prototyping. SBP is based on the use of surrogate software as a means of simulating or prototyping different application parts or concerns. These surrogates are commercial, in-house, or open source software with certain functionalities.

Our working *hypothesis* has been that functionally testable software prototypes could be created with the purposeful combination of surrogate software. The *major objective* of SBP is to provide a relatively high-fidelity realization of the intended software functionality by exploiting the functional affordances of a set of possible surrogates. This reduces the efforts and time necessary for original code development in the prototyping phase, while it offers the opportunity for a faster functionality and utility testing.

The combination of the appropriate surrogated components can be used for concept forming and testing. In Figure 9, the process of moving towards the prototypes is shown together with the functionality and usability testing. To handle complexity in software development, separation of concerns is needed and is achieved by functional decomposition. The resultant sets of functions should form the basis of the identification of useful surrogates for prototyping the software product. Obviously, due to the complexity of realizing multiple functions, multiple surrogates should be used to construct a prototype of the entire system.

Considering the current availability and diversity of software products in the market, we can assume that there are enough software surrogates available, based on which SBP can be built. In general terms, surrogate software can be used directly (compositional) or by using the functionalities of the surrogate as a programming language (generative). Surrogates enable rapid high fidelity prototyping, thanks to the fact that composition work generally takes less time than generative software building. On the other hand, there is a need to define and develop interfaces between the often non-homogeneous components. Besides high flexibility, the lack of strict rules on how to apply the principles, the biggest challenge of using surrogates is the interface between the different surrogate components. To tackle the interface definition / implementation problem, SBP can be supported by the *novel approach* of platform-based component design. A platform-oriented approach and development resource can handle the interfacing and communication between other, different surrogate components.

We began the application of SBP to the reference case by generating interaction diagrams, which detailed the needed interactions between the software and the stakeholders, especially the product designers. Based on these diagrams, a functional decomposition and hierarchical ordering was made. Resource selection was a difficult task, due to the large number of available software surrogates. In order to eliminate the interfacing problem, we decided to follow a platform-based approach. Therefore, affordance matching was carried out with the objective of finding an appropriate platform-surrogate. The Drupal was chosen

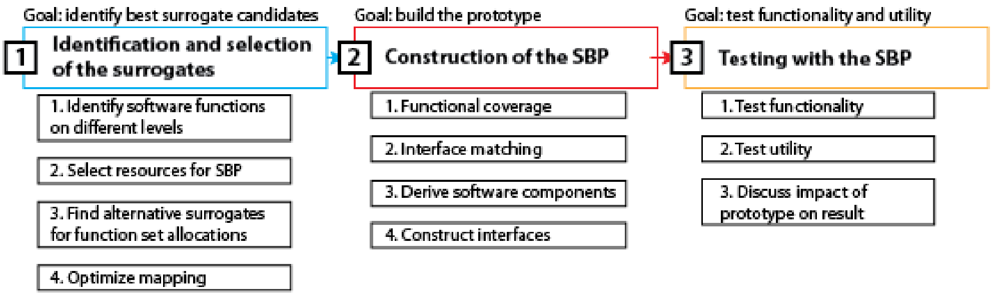
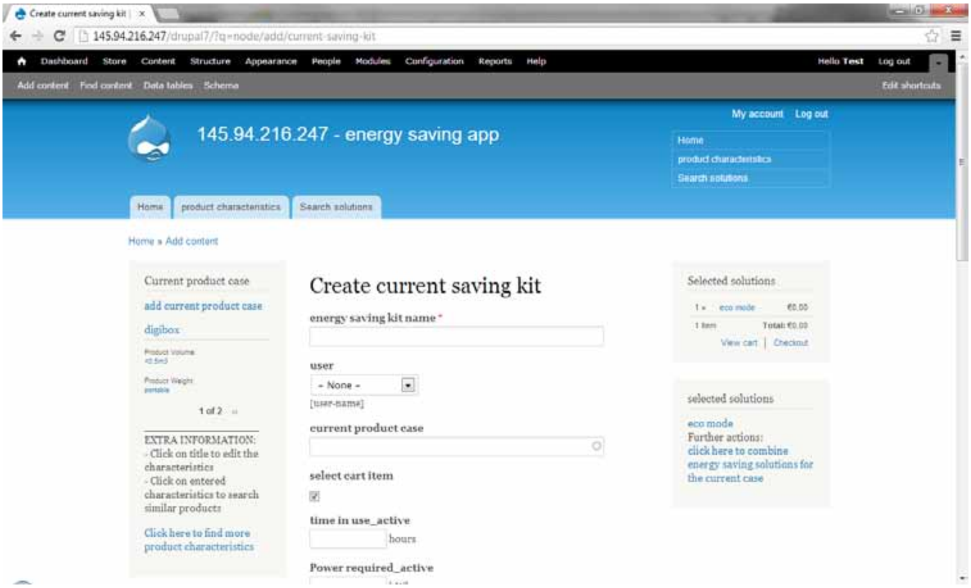

Goal: identify best surrogate candidates
1
Identification and selection of the surrogates
1. Identify software functions on different levels
2. Select resources for SBP
3. Find alternative surrogates for function set allocations
4. Optimize mapping
Goal: build the prototype
2
Construction of the SBP
1. Functional coverage
2. Interface matching
3. Derive software components
4. Construct interfaces
Goal: test functionality and utility
3
Testing with the SBP
1. Test functionality
2. Test utility
3. Discuss impact of prototype on result

***Figure 9.** Process of SBP towards software development testing*

because it had the largest match with the high level functions. Afterwards affordance matching was conducted, concerning the lower level function sets and the 20000 or more available Drupal modules. We identified the highest functional coverage and built the software components using the selected surrogate modules. The advantage of this was that the interface construction was solved throughout the Drupal platform.

The screenshot shown in Figure 10 gives an impression of the Drupal prototype. Functionality tests were carried out by the supporting development team. The intended interactions were checked to ensure that they were effectively and efficiently realized. In general, we were pleased with the efficiency and the ease with which the SBP methodology was applicable to the reference case. The implemented SBP was also used in user/application contexts. The improvement opportunities of the software tool could not be gathered without prototyping the functionalities. SBP proved to be a useful and effective method of developing a testable, tangible prototype in a short time. However, the development of the SBP was not a commodity and we faced several new challenges. As there is an infinite amount of potential surrogates, it is impossible to obtain an overview of the possible affordances of all surrogates, and it was a challenge to identify the best platform and modules. The Drupal system, as a platform, was advantageous for our purpose. However, we underestimated the high learning curve, so it took a long time to digest and implement the potentials of the platform. Regarding the improvement of the software product from the designers' viewpoint, it was important that the focus shifted from a procedural, functional view to a content view during the software implementation. The strength of SBP lies in the fact that all functions can be realized and tested before the final implementation.



Current product case

add current product case

digibox

Product Volume
<0.5m3

Product Weight
portable

1 of 2

EXTRA INFORMATION:
- Click on title to edit the characteristics
- Click on entered characteristics to search similar products

Click here to find more product characteristics

Create current saving kit

energy saving kit name *

user
- None -
[user-name]

current product case

select cart item
☑

time in use_active
hours

Power required_active

Selected solutions

1 ×	eco mode	€0.00
1 item		Total: €0.00

View cart | Checkout

selected solutions

eco mode
Further actions:
click here to combine energy saving solutions for the current case

Figure 10.   Screenshot from the Drupal prototype

Validation of the multi-phase methodology

The validation of design methods is important for the continuing advancement of both design theory and the professional practice of engineering. Researchers in design theory proposed going over validation processes to guide the development and evaluation of new methods. Professional practitioners need validation processes to determine which methods to employ, and when and how to employ them. Validation of methodologies cannot be based on mathematical modeling, but on somewhat subjective evaluations. A methodology typically operationalizes human knowledge that also contributes to the subjective nature. In the last phase of the promotion research, we had to validate the proposed designerly software development methodology in a qualitative as well as in a quantitative manner. Actually, our *objective* was to check the external validity of the software design methodology. For the sake of completeness, we note that internal validation of the work and findings has also been made, but it was done in the confirmative parts of each research cycle.

Considering the validation in the context of the DSDM, we found that the external validation was most efficient using a reflective validation approach. Comparison was not possible, as only a single case was developed in the study. Moreover, executing an additional comparative validation would have raised the need for an extra research cycle, in which a comparison could have been made by a simultaneous development of a specific software product using on the DSDM and a traditional software development method. To validate the DSDM, we studied the literature to find a suitable validation method. Our search ended with a negative result. Nevertheless, after adaptation, one specific method called the validation square seemed applicable as a generic external validation method. We studied both the theoretical and methodological fundamentals of the validation method, as well as the necessary adjustments and extensions in our context. In addition, we defined the application of the quadrant-based method for our specific purpose. The quadrant-based external validation method combines structural and performance assessment actions both in the theoretical domain and the application domain. The different steps, specific for each quadrant, allow both qualitative and quantitative assessment according to various criteria in a reflexive manner, beginning with the main structural and performance characteristics. A visualization of the quadrant-based validation is shown in Figure 11.

The quadrant-based validation proved to be a valuable method for the validation of the proposed designerly software development methodology. Based on the findings, we could conclude that DSDM is a suitable and adequate methodology for the development of software products that (i) have complex functionality, (ii) should cope with user requirements that are rather uncertain and unclear at the beginning of the process, and (iii) are complex due to environmental aspects, modeling need and data processing. The application of the DSDM methodology supports the development of understandable, reliable, efficient and modifiable interactive software products by managing the complexity of the software concept, dealing with its evolving requirements, and relying on stakeholders' feedback.

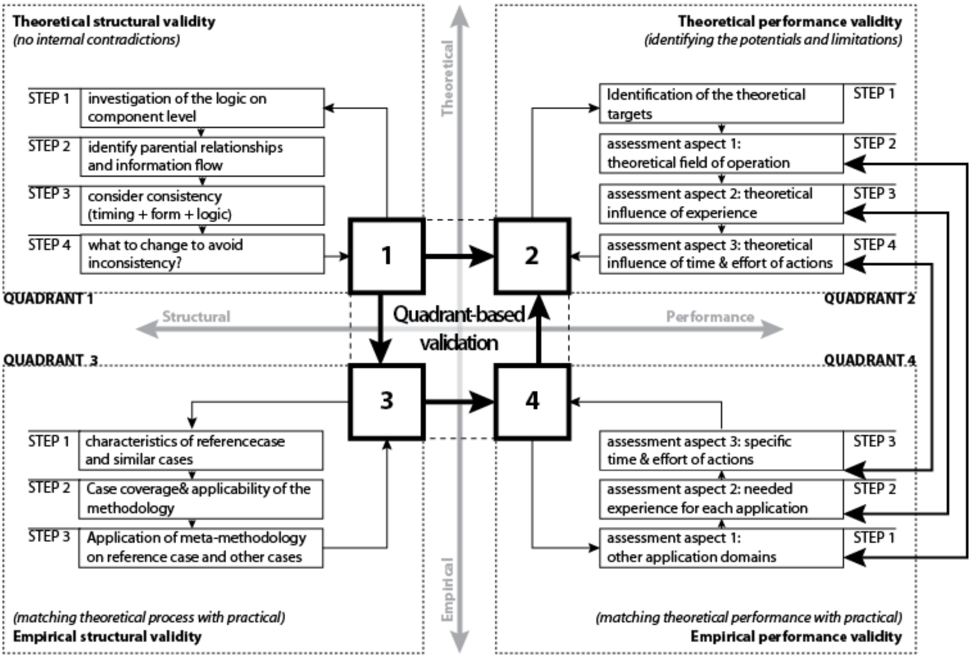

Theoretical structural validity
(no internal contradictions)
STEP 1
investigation of the logic on component level
STEP 2
identify parential relationships and information flow
STEP 3
consider consistency (timing + form + logic)
STEP 4
what to change to avoid inconsistency?
QUADRANT 1
Theoretical
Theoretical performance validity
(identifying the potentials and limitations)
Identification of the theoretical targets
STEP 1
assessment aspect 1: theoretical field of operation
STEP 2
assessment aspect 2: theoretical influence of experience
STEP 3
assessment aspect 3: theoretical influence of time & effort of actions
STEP 4
QUADRANT 2
1
2
3
4
Quadrant-based validation
Structural
Performance
QUADRANT 3
QUADRANT 4
STEP 1
characteristics of referencecase and similar cases
STEP 2
Case coverage& applicability of the methodology
STEP 3
Application of meta-methodology on reference case and other cases
(matching theoretical process with practical)
Empirical structural validity
Empirical
assessment aspect 3: specific time & effort of actions
STEP 3
assessment aspect 2: needed experience for each application
STEP 2
assessment aspect 1: other application domains
STEP 1
(matching theoretical performance with practical)
Empirical performance validity

Figure 11. The quadrant-based validation with its dedicated executions steps

Conclusions

To conclude the promotion research, we could formulate different propositions on the following aspects: (i) on the DSDM and its single-phase methodologies, (ii) on the overall research approach, (iii) on the reference case, and (iv) the quadrant-based validation method and validation outcome for the DSDM.

Proposition 1:

In order to achieve an optimum support and efficiency, the proposed DSDM has been developed as a multi-phase stakeholder-oriented designerly methodology, which offers suitable procedures, instruments and methods for three single-phase methodologies: (i) critical collective reflection, (ii) modular abstract prototyping, and (iii) surrogate-based prototyping.

Proposition 2:

Concurrent development of the parts and the whole of the methodology and applying it to an evolving reference case, lends itself to a short cycle learning process.

Proposition 3:

Breaking down the research project into a sequence of interconnected research cycles not only helps to structure the work, but also to find the necessary and sufficient scope and balance of the research topics and activities.

Proposition 4:

DSDM can be applied to all cases of interactive software development showing a similar structure and (performance) target as the reference case.

Proposition 5:

The methodology of CCR facilitates improved collective requirements engineering and framework conceptualization through the direct reflection of expert-stakeholders on the proposal demonstrated by the software developers

Proposition 6:

MAP offers the possibility for a rapid development of modularly configurable and presentable content. It also supports focused demonstration to stakeholder groups and their decision making process, by using abstract prototypes.

Proposition 7:

The proposed SBP methodology allows fast and cost effective tangible prototyping for functionality and usability testing, based on a composition of surrogate-software means.

Proposition 8:

By using platform-based SBP, it is possible to go beyond the conventional concept of pure component-based design. A platform-based SBP reduces the problem of interfacing of heterogeneous components.

Proposition 9:

DSDM proved to be a valid methodology for the development of interactive software products that have complex functionality, user requirements that are rather uncertain and unclear in the beginning of the process, and that are complex due to environmental aspects, required modeling and data processing.

Proposition 10:

The adapted quadrant-based validation is a valuable approach for the validation of software development methodologies. In addition, it has a large application potential and is flexible enough in single-case, reflexive, context-dependent assessments.

Samenvatting

Software ontwikkelen met een ontwerpmethodologie

Achtergrond

Dit PhD onderzoek vertrekt vanuit het domein van de industriële productontwikkeling. In de laatste decennia, merken we dat dit domein sterk beïnvloed wordt door vier belangrijke trends, die onderling verband houden. Deze trends zijn: (i) verandering van de ecologische, economische en sociale behoeften en (ii) de snelle ontwikkeling en de ongestructureerde verspreiding van geavanceerde technologieën. Deze twee trends hebben tot gevolg dat (iii) er een evolutie is van het product en het gebruik (bv. evolutie van product naar software en diensten die niet tastbaar zijn). Bijgevolg (iv) is er een stijgende nood aan aanpassing van de ontwerpaanpak en -methodieken.

Deze trends zorgen samen voor een karakteristieke verandering in het type producten en in de aanpak hoe producten worden gerealiseerd. We nemen waar dat er een verschuiving is in de aandacht van de ontwerpers van pure vorm, functie, materiaal en de wijze van productie naar het nut, de menselijke ervaring, de bruikbaarheid en wenselijkheid van het product. Deze verschuiving van een functie-georiënteerde benadering naar een consument-georiënteerde en verder naar een ruimere mens- en context-gerichte aanpak, is een belangrijke evolutie.

We identificeren drie generaties van producten waarvoor een verschillende aanpak en methodologieën nodig zijn. De eerste generatie zijn de geassembleerde hardware producten (bv. fiets), pure software-implementaties (bv. Word) en zuiver diensten (bv. kinderopvang). De tweede-generatie producten tonen een groeiende mate van samensmelting van eerste generatie producten. We categoriseren drie soorten systemen: product-dienst systemen (bv. autodelen), *embedded* systemen (hardware en software combinaties bv. wasmachine) en informatiesystemen (software en dienst combinaties bv. stockmanagement). De derde generatie producten zijn complexe systemen en omgevingen en worden vaak omschreven als cyber-fysische duurzame producten en diensten aan consumenten, ofwel *cyber-physical systems*, *internet of things*, enz. (bv. intelligent verkeerssysteem)

Onderzoeksdomein en onderzoeksvraag

In dit onderzoek richten we ons op kennisexploratie en synthese voor methodologie ontwikkeling. Het onderzoek focust op de conceptualisering en implementatie van een ontwerpmethodologie om ontwerpers en ingenieurs bij de ontwikkeling van complexe tweede en derde generatie producten te ondersteunen. We concentreren ons op de ontwikkeling van software als product of als onderdeel van complexe systemen. Dit domein is geselecteerd omdat software de grootste opportuniteiten biedt om behoeften naar complexiteit en evolueerbaarheid te vervullen. Software producten bestaan als zelfstandige applicaties, pakketten, maar ook als embedded software, besturingssystemen of als component van complexe informatiesystemen. Voor dit onderzoek richten we ons enkel op interactieve software applicaties.

In het bijzonder op toepassingsgerichte ontwerp software tools, die worden gebruikt door ontwerpers om hun denken en creatieve processen te ondersteunen, om veranderende real-time gegevens dynamisch te verwerken, voor eenvoudige en effectieve communicatie en data management. Als testcase voor de software ontwikkelingsmethodologie ontwikkelden we een interactief hulpmiddel dat ontwerpers ondersteunt in hun zoektocht naar energiebesparing in huishoudelijke apparaten. Deze software tool werd gekozen als testcase. De case wordt verder in het onderzoek gebruikt als archetype van vergelijkbare tools.

De innovatiewaarde van dit onderzoek zit zowel in de ontwikkelde methodologie als in de onderzoeksaanpak waarbij we gelijktijdig de methodologie en de testcase ontwikkeld hebben. De tot stand gekomen dialectische interactie biedt voordelen voor beide, dankzij de gelijktijdige ontwikkeling van een toepassing in de praktijk is onmiddellijk optimalisatie mogelijk van de methodologie.

Onderzoeksdoelstellingen en hypotheses

Om tegemoet te komen aan de hier boven beschreven trends, moet de methodologie aandacht hebben voor volgende aspecten: (i) gestructureerd proces, (ii) de stakeholders (gebruikers, leveranciers, experten, …) centraal plaatsen, (iii) aangepast zijn aan het denkpatroon van ontwerpers en (iv) ondersteunen met specifieke methodes, instrumenten en technieken. De nauwe relatie tussen producten en hun eindgebruikers benadrukt de noodzaak van een meer intense betrokkenheid van alle stakeholder in het ontwerpproces. Dit stelt ontwerpers en ingenieurs in staat om (i) de behoeften voor verandering te begrijpen en (ii) om te gaan met de uitdagingen van complexe functionaliteiten en snelle realisatie van processen in een context-afhankelijke manier.

Een extra moeilijkheid is enerzijds dat er meestal verschillende soorten stakeholders zijn in een project, zoals bijvoorbeeld eindgebruikers, leveranciers, klanten, marketeers, management, kennis experts, IT-onderhoud, enzovoort. Deze stakeholders zijn betrokken in de verschillende fasen van de product levenscyclus (specificaties, algoritme ontwikkeling, codering en productie, distributie, gebruik, onderhoud, enz.) en de context van het systeem. Anderzijds zijn er geen algemene regels, omdat de noodzaak van betrokkenheid van stakeholders afhankelijk is van de concrete context.

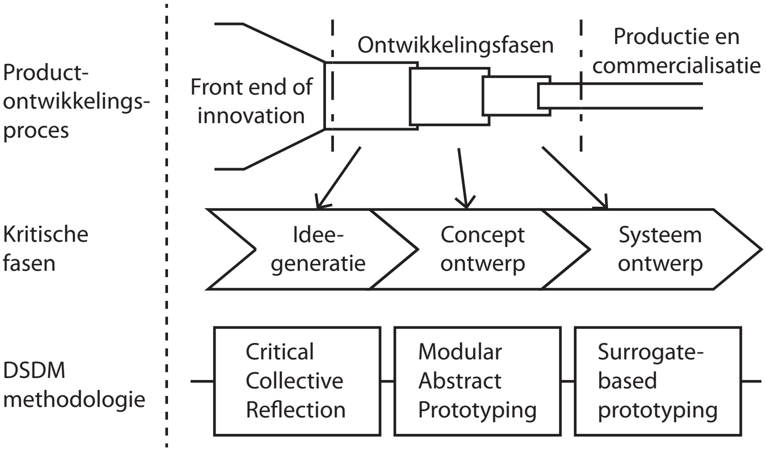
Onze eerste hypothese in dit PhD onderzoek is dat software tools die behoren tot de categorie van de interactieve applicatie software moeten ontwikkeld worden volgens een participatieve ontwerpstrategie waarbij alle stakeholders betrokken zijn. Onze waarneming was dat de menselijke aspecten in het ontwerp van interactieve software vaak niet het gewenste niveau bereiken, in vergelijking met hardware consumenten producten. Onze visie is dat een ontwerp gebaseerde methode nodig is om tegemoet te komen aan het gestelde probleem. Dit leidt tot twee uitdagingen: (i) heroriënteren van het ontwikkelingsproces van interactieve software naar een ontwerp-gebaseerde aanpak met aandacht voor alle stakeholders en (ii) uitwerken van een nieuwe methodologie die de vroege fasen van de ontwikkeling van software bestrijkt waar belangrijke beslissingen worden gemaakt. Onze doelstelling is het verhogen van de bruikbaarheid en de kwaliteit van interactieve softwareproducten. Door het betrekken van de stakeholders in de vroege fasen, kunnen producten worden gemaakt op maat en beter passend bij de behoeften, ongeacht de nood aan extra tijd en inspanningen.

Focussen op mensen en hun ervaringen is cruciaal in de huidige productontwikkeling. Onze tweede onderzoekshypothese stelt dat softwareontwikkeling kan gebruik maken van de principes van participatieve mensgerichte aanpak van hardware consumenten productontwikkeling . De doelstelling is het onderzoeken van de verschillen tussen de ontwikkeling van hardware en software producten en of men niet direct gebruik kan maken van de human-centered (mens-gerichte) ontwerp principes voor hardware consumptiegoederen in de ontwikkeling van software en hoe we specifieke methodologische principes zouden kunnen gebruiken als basis voor de ontwikkelde software ontwikkelingsmethodologie.

Onderzoeksaanpak

Vanwege de verscheidenheid aan doelstellingen en contexten in het ontwerpproces, werd het promotieonderzoek onderverdeeld in vijf onderling gerelateerde cycli (research cycli of RC). In de eerste cyclus (RC1) onderzochten we de behoefte aan betrokkenheid van de stakeholders in de huidige softwareontwikkeling benaderingen en beschrijven we de context van het onderzoeksproces en de referentie case. Tijdens de uitvoering van RC 2, 3 en 4, hebben we de drie meest kritieke stadia van het ontwerpproces bestudeerd (ideegeneratie, conceptontwikkeling en systeem uitwerking). In de RC 5 focusten we ons op de externe validering van de voorgestelde methodologie. De validatie werd uitgevoerd gebruik makend van de *quadrant-based validation* methode. Een methode die bestaat uit vier kwadranten om de methodologie te valideren op een structureel en prestatie niveau en op een onafhankelijke en toegepaste manier. De conclusie was dat de methodologie valide is voor de ontwikkeling van interactieve software producten die een complexe functionaliteit hebben, waarvan de eisen en wensen onduidelijk en onvolledig zijn in het begin van het ontwerpproces en die een hoge nood aan modeleren en gegevens verwerking hebben.

De essentie van de ontwerpgebaseerde software ontwikkeling methodologie



Figuur: overzicht van de DSDM methodologie

De ontwerpgebaseerde software ontwikkelingsmethodologie (DSDM) steunt op volgende principes: (i) context gevoelig en betrokkenheid van alle stakeholders met het oog op het verkrijgen van kwalitatieve verbeteringen en/of veranderingen, (ii) rekening houden met de evolutie van complexe systemen en (iii) gebruik maken van een toenemende mate van detaillering: beginnend met een hoog abstractieniveau (die kan worden ingebed in een rudimentair prototype) en eindigend met een zeer gedetailleerd testbaar prototype van het softwaresysteem. Tijdens dit proces kunnen meningen, ideeën en aanbevelingen van de verschillende stakeholders verkregen worden gebruik makend van prototypes die ondersteunen in zowel de ondervraging als de constructieve activiteiten.

De DSDM bestaat uit 3 afzonderlijke methodologieën die zich elk richten op één van de drie cruciale fasen van het software ontwikkelproces: (i) ideegeneratie en architectuur, (ii) concept ontwerp en integratie en (iii) uitontwikkeling van het systeem. Vanwege de diversiteit van de fasen van het ontwikkelingsproces bouwden we voor elke fase een afzonderlijke methodologie op: (i) Kritische collectieve reflectie (CCR), (ii) Modulair abstract prototypen (MAP) en (iii) surrogaat-gebaseerd prototypen (SBP). Hierna volgt voor elke fase de bespreking van de methodologie.

Fase 1: ideegeneratie en architectuur

Doel van de fase: De kennis van de verschillende probleemdomeinen samenvoegen tot een consistent geheel en het creëren van ideeën op systeem niveau en het scheppen van een conceptueel kader (architectuur) voor een abstracte oplossing. Het probleem in deze fase is de onvolledige kennis van de context en de slecht gedefinieerde en tegenstrijdige ideeën. Stakeholders moeten worden betrokken bij de verwachtingen, behoeften en doelstellingen van de software om de kritieke ontwerpbeslissingen te bespreken.

Naam methodologie: Critical Collective Reflection (CCR) of kritische collectieve reflectie.

Theorie: De CCR methodologie is gebaseerd op het principe van triangulatie: antwoorden vergelijken van het onderzoeksteam met dat van experten om zo tot de beste oplossingen te komen voor alle deelproblemen.

Proces, methodes en technieken: CCR wordt verdeeld in zes stappen: (i) om de complexiteit van software ontwikkeling te behandelen, moeten de verschillende ontwerpproblemen worden geïdentificeerd teneinde de software op te splitsten in beheersbare delen. (ii) *het* ontwerp *opdelen de* concrete deelproblemen waarvoor ontwerp opties kunnen worden gevonden. De methode van morfologische analyse wordt gebruikt om de oplossing opties te vinden en in kaart te brengen in een visueel overzicht. Dit overzicht helpt om ontwerpbeslissingen te maken op het niveau van de individuele ontwerpproblemen en een combinatie van oplossingen te selecteren om te komen tot een concept. (iii) Het nemen van ontwerpbeslissingen is een activiteit die moet worden uitgevoerd enerzijds door het ontwerpteam en anderzijds door deskundige stakeholders. (iv) Het ontwerpteam zet meteen haar conclusies in een eerste conceptueel kader van de beoogde software. (v) Door middel van een geleide expert discussie, kan een collectieve beoordeling worden uitgevoerd op de verschillende ontwerpbeslissingen. Op basis van de ontvangen informatie van de experten- wordt een verbeterd kader gebouwd. (vi) De gelijkenis en het verschil tussen de twee versies van het kader wordt onderzocht en deze conclusies worden meegenomen naar de tweede fase.

Toepassing in case: Tijdens de toepassing van de CCR werden tien belangrijke ontwerpproblemen geïdentificeerd voor welke ontwerpopties moesten worden gegenereerd. Zowel het ontwerpteam als expert stakeholders overwogen deze problemen en opties en ontwikkelden een conceptueel kader met daarin alle ontwerpbeslissingen.

Fase 2: concept ontwerp

Doel van de fase: In de concept ontwikkelingsfase, bestaat software als een functioneel concept. Deze fase focust zich op het ontwerp en de validatie van het conceptontwerp. Validatie is cruciaal in deze fase om ontwerpfouten uit te sluiten. Er ontbreken echter goede modellering, simulatie en demonstratie middelen die gebruikt kunnen worden om het concept te bespreken met de verschillende stakeholders. Daarnaast is de belangrijkste beperking de onvolledigheid en de vaagheid van de informatie die beschikbaar is.

Naam methodologie: Modular abstract prototyping (MAP) of modulair abstract prototypen methodologie.

Theorie: MAP ondersteunt de ontwikkeling van een rijk en compleet abstract prototype dat gebruikt kan worden om feedback te krijgen van alle stakeholders over het gedemonstreerde concept. Dit prototype bevat een abstracte levensechte demonstratie van alle karakteristieke werkings-, interactie- en gebruiksprocessen met inbegrip van de acties van de verschillende stakeholders en de gebeurtenissen in de omgeving. Door de informatie over het software concept in verschillende modules te demonstreren krijgt men een flexibele en gerichte communicatie die (inhoudelijk en vormelijk) aangepast kan worden aan elk type stakeholder.

Proces, methodes en technieken: Het uitvoeringsproces van het MAP kan opgesplitst worden in vier stappen: (i) conceptualisering: bepalen van de inhoudelijke informatie, de context informatie van het software concept die door het prototype gecommuniceerd moet worden. (ii) Het ontwerp van het prototype: een belangrijk aspect is het bepalen van het optimale aantal modules, dit is gerelateerd aan het aantal stakeholders, welke informatie ze nodig hebben (hun interesses) en de gewenste media. (iii) De uitvoering van de testen in sessies volgens de verschillende profielen van stakeholders. Op deze manier wordt de informatie overload verminderd en kan het demonstratie materiaal flexibeler en efficiënter worden ingezet. Na de demonstratie volgt een gedetailleerde discussie en bespreking van de verschillende aspecten van het concept en moeten de stakeholders het concept beoordelen, verbeteringen en wijzigen suggereren. Ten slotte, (iv) moeten deze gegevens geëvalueerd worden door de feedback uit te schrijven en te organiseren in concrete conclusies voor concept verbetering.

Toepassing in case: We identificeerden drie soorten stakeholders, elk met hun eigen kenmerken en interesses: (i) productontwerpers, (toekomstige eindgebruikers van de applicatie), (ii) software ontwikkelaars (programmeurs van de toepassing) en (iii) kennis ingenieurs (voor kennisverwerking en voor de databank met kennis die de basis vormt van de applicatie). Voor deze stakeholders bouwden we zes modules, met behulp van geanimeerde films met verschillende visuele representaties van de werking en inhoud. Tijdens de uitvoering werden, per profiel focusgroep sessies georganiseerd met relatief kleine groepen(6-12 deelnemers). Op basis van de voorstellen werd een verbeterd concept gebouwd dat in de volgende fase verder gedetailleerd kan worden.

Fase 3: uitontwikkeling

Doel van de fase: De uitontwikkeling van alle aspecten van de software: interface, structuur database… ook wel gedetailleerd ontwerp genoemd. Om deze details en het volledige systeem te kunnen testen naar functionaliteit en gebruiksvriendelijkheid is er nood aan een prototype dat snel en tegen lage kosten kan worden verkregen maar dat toch kan worden gebruikt voor de systeemtesten die functionele of structurele modificaties kunnen identificeren.

Naam methodologie: Surrogate-based prototyping (SBP) of plaatsvervangend prototypen methodologie

Theorie: We nemen aan dat het groeiende aantal software producten dat beschikbaar is, gebruikt kan worden om snel testbare prototypes te ontwikkelen teneinde functionele of structurele modificaties aan het einde van het ontwikkelingsproces te beperken. SBP is gebaseerd op het gebruik van surrogaat software als simulatie voor verschillende toepassingsonderdelen van het systeem ontwerp. Deze surrogaten zijn commerciële, in-house of open-source software producten met bepaalde functionaliteiten. De combinatie van deze surrogaat componenten simuleert de werking van het volledige software systeem. Dit vermindert de inspanningen en tijd nodig voor originele codeontwikkeling in de prototyping fase en biedt de mogelijkheid om sneller functionaliteit en bruikbaarheid te testen.

Proces, methodes en technieken: Om de complexe software te kunnen simuleren, vertrekt de methodologie van een opdeling van de functies in sets. Deze sets van functies vormen de basis om verschillende bruikbare alternatieve surrogaat software producten te selecteren. Uiteraard zijn vanwege de meerdere functies, verschillende surrogaten nodig om het gehele systeem te prototypen. Gezien de beschikbaarheid van software producten op de markt, kunnen we aannemen dat er genoeg software vervangproducten beschikbaar zijn voor de SBP, dit maakt het echter wel moeilijk om de beste alternatieven te selecteren. In het algemeen gesproken kan surrogaat software direct worden gebruikt (samengesteld) of door gebruik te maken van de functionaliteiten van de software als programmeertaal (generatief). Surrogaten worden gebruikt om snelle prototyping te realiseren, omdat dit doorgaans minder tijd kost dan software te coderen, aangezien tussen deze componenten alleen interfaces moeten worden gebouwd. De grootste uitdaging van het gebruik van surrogaten, naast de hoge flexibiliteit en het ontbreken van regels over hoe ze toe te passen, is de interfaces uitwerken tussen de verschillende surrogaat onderdelen. Om dit probleem aan te pakken gebruiken we in SBP een platform-surrogaat. Dit platform-surrogaat zorgt voor de interfaces en de communicatie tussen de verschillende andere surrogaat-onderdelen. Door de functionaliteit en gebruiksvriendelijkheid van het prototype te testen, kan het ontwerp een laatste maal geoptimaliseerd worden, alvorens de software definitief geprogrammeerd kan worden.

Toepassing in case: We vertrokken van diagrammen waarin gedetailleerd de interacties tussen de software en de stakeholders staan. Op basis van deze diagrammen maakten we een functie hiërarchie van het software systeem. Het zoeken naar de surrogaten was een zware taak door de beschikbarheid van een onbeperkte hoeveelheid software surrogaten. We besloten om een platform-benadering te volgen om het probleem van interfacing te verhelpen. De eerste stap was dan ook om een geschikt platform-surrogaat vinden. Drupal werd gekozen omdat dit het grootste potentieel biedt naar flexibiliteit en de hoogste overeenkomsten heeft met de functies op hoog niveau. In de volgende stap gingen we op zoek naar geschikte modules door de functie sets te matchen met de 20.000 beschikbare Drupal modules. We identificeerden de hoogste functionele dekking en bouwde de software componenten met behulp van de geselecteerde surrogaat modules. Interface maken was niet nodig omdat dit door het Drupal platform werd opgelost. Functionaliteitstesten werden gedaan door het ontwikkelteam, om te controleren of alle beoogde interacties effectief en efficiënt waren gerealiseerd.

Dankwoord

Nobody can complete a PhD research by only relying on his own force, knowledge, motivation and so on. If I recapitulate this past period of hard working and keeping believing that I will succeed, I realize how important the contribution of some people was. Actually, there were many people, who helped me (directly or indirectly) to become a researcher and to realize this thesis. And although it seems to be impossible to thank everyone in person, I do want to thank some people in particular:

Prof. dr. Horváth, dear Imre, thanks for being my supervisor, to believe in me and to share your passion for science. One day, you told me a joke about a little rabbit that wants to do a PhD, and how fortunate he was to have a lion as supervisor. I cannot recite the joke as you can, but I'm sure I kept the message. Furthermore, I'm certain I had a lion as supervisor too. Thanks for learning me what science is and how to do, structure and organize research activities, and how to write papers. Thanks to have patience to explain methods and approaches over and over again and to learn me to be critical and to push me into our heavy discussions. I will remember our good relationship and I hope we can maintain this with a good Belgian beer..

Dr. Ing. Van Doorsselaer, dear Karine, thanks for being my co-promotor. For me you were a perfect mental coach, so thank you to listen to my plans, ideas, and for supporting me in organizing my mental chaos.

Members of my doctoral committee, thank you for your efforts to read my thesis for the thorough feedback.

Participants of the focus group sessions, thank you for your active participation in heavy discussion sessions. Thanks to Ellen for assisting me in transcribing these sessions.

My fellow researchers at the CADE section of industrial design engineering at the TU Delft, I would like to thank you all for their fair and honest critical remarks during our wonderful discussions. Thanks to Adrie in specific for helping me with the Drupal system and to Eliab for proofreading. Thanks to Liz for our nice talks to express our frustrations and share our problems and successes. I'm sure you will be a good teacher in Colombia and that you will enthusiasm plenty of students. Thanks to Niels for sharing your room with me, for giving me courage and sharing your rich life experiences.

My fellow researchers at productontwikkeling in Antwerp, thanks to Elli, Ann, Sarah, Ivo, Ingrid, Kristof, Dries, Alexis, and Lukas for sharing thoughts and for the nice time together. Special thanks to Elli for being a pioneer in the PhD trajectory at productontwikkeling and to figure out all those ambiguities and problems, and mostly for sharing those with us. Thanks to my teaching colleagues who accepted me to be part of their crew.

Last, but not least, thanks to my family for the support they provided me through my entire life. There is one last person who deserves the biggest thank over all – my husband Gert. Without your love, encouragement and support, I would not have finished this thesis.

Biography

Els Du Bois was born in Antwerp, Belgium on July 8th 1985. In 2003, she finished secondary school with a degree in economics and mathematics. From 2003-2008 she studied product development (productontwikkeling) at the Artesis University College of Antwerp, where she received her master degree in 2008.
In January 2009, she started her PhD research at the Delft University of Technology, in the faculty of industrial design engineering, and in cooperation with the Antwerp University. Moreover, she coaches Bachelor and Master students in both design and research assignments and is involved in other research projects in design at the Artesis University College of Antwerp. Her research interest includes sustainable design, product development, software design, complex systems, and early design and conceptualization.