العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Part

I

Introduction

Chapter 1

Overview

> [N]eglect of context is the greatest single disaster which philosophic thinking can incur.
>
> John Dewey (1931)

Contents

1.1 Introduction

This dissertation is submitted for the degree of Doctor of Philosophy of the Australian National University. It describes exploratory research on the fundamentals of software/system variation, its sources and implications, on the level of requirements. The result of the research is a novel approach to analyse and represent *context* in software engineering, with application to systems in general.

As an introduction, I provide a brief overview of main parts of the research, supported by guiding information to navigate through the thesis. I conclude with a list of work published during the project and a summary of contributions made to engineering research and practice.

1.2 Motivation and research aim

The initial motivation for conducting this research is to explain sources of software/system variety across different organisational goals, different disciplinary approaches, cultural choices, and within personal preferences. In the spirit of the dichotomy set by Simon [1996], as engineers, we are not primarily interested in the knowledge of how the world works, we are more concerned with the knowledge of making a working world. Therefore, we have to enquire about the nature of the world of our artefacts. One challenging goal, in software development, is to maintain a complete account of system requirements. For example, how to avoid system failures as a result of lack of knowledge? How to take advantage of system opportunities, manifested in customers' desires and aspirations? These concerns are summarised by Glass's Law:

> *Requirements deficiencies are the prime cause of project failures.*
>
> [Endres and Rombach 2003, Law L1, pp16-17]

The research is also motivated by the growing interest in product line architectures. In which the aim is to manage a variety of needs within a family of products, while maintaining economical value and achieving sustainable growth. My preliminary research reported in Chapter 2 shows that the source of sustainable functionality of a successful system, lies in its ability to respond to its context. This is summarised by Conway's Law:

> *A system reflects the organisational structure that built it.*
>
> [Endres and Rombach 2003, Law L16, pp81-82].

Further research reported in Chapter 2 shows that *context* as a concept, has not received enough theoretical distillation in software engineering. This is exemplified in the different views by software practitioners about the use and meaning of context on different levels of software development—requirements, architecture, and design. After reviewing views and theories from the literature dealing exclusively with the concept of *context*—literature from various fields of knowledge: architecture and urban design, artificial intelligence, anthropology, linguistics, philosophy—the need emerged for a synthesis of *context* as a separate system concern in software analysis.

This preliminary work has resulted in the following research aim:

> *'to present a model of context that shows when to vary and when not to vary a system. Such a model should indicate the opportunity to vary the system when the context reflects soft demands, and indicate when it is not possible to vary the system because the context has strict demands. The model should also indicate when strict demands are based on conjecture, and when soft demands are based on strong evidence. The model should show different degrees of variation that the system may have through context.'*

1.3 Thesis scope

The scope of this thesis is the development, representation, and demonstration of the theoretical framework of *context* within software engineering requirements. While the broader applicability of the contextual framework to other areas of software development is discussed, the evaluation of the approach within these areas is beyond the scope of this thesis.

The research on context presented here, have general application to human-based systems. Some reference is made to the implication of the work on human-based systems, but demonstrating or evaluating such implications are beyond the scope of this research.

1.4 Thesis structure

Figure 1.1 shows the organisation structure of the thesis depicted in a Unified Modelling Language (UML) activity diagram [Mellor and Balcer 2002]. The thesis is organised in three parts, represented in the activity diagram with vertical lines,

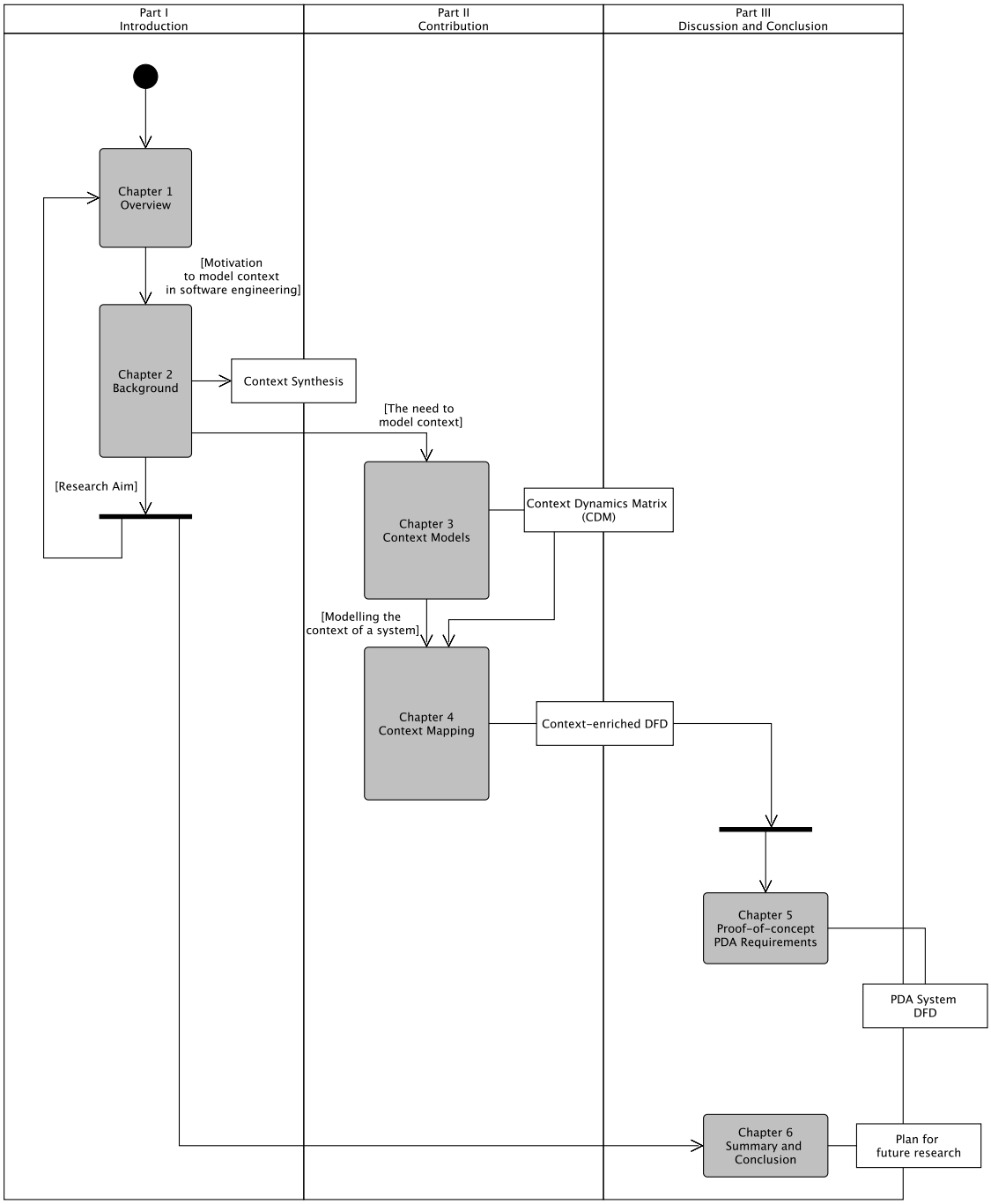


Figure 1.1 – Activity diagram depicting the structure and the flow of ideas and results throughout this thesis.

each part with more than one chapter. Chapters are represented as activities (grey rounded boxes), flow of ideas and conclusions between chapters are represented in arrows and key research contributions are represented by objects (white square boxes).

In the following sections I provide an overview of the thesis with a summary of the findings of each chapter. A more detailed overview of the main research contributions is obtained by reading the introduction of Chapters 3–4 with the conclusion of Chapter 6. A more concise overview of the thesis could also be obtained by reading the preface.

1.4.1 Part I - Introduction

The first part of the thesis (Part I) is formed by two chapters: Chapter 1 and Chapter 2. I present the thesis aim and motivation in this chapter, Chapter 1. Preliminary research is discussed in Chapter 2. In which initial observations leading to realise the need to explain system variety by modelling context is presented. These observations were inspired by the work of Alexander [1964; 1979; 2002] on context and form, Dilley [1999], Scharfstein [1989] on the problem of context, and the discussions of context and patterns in Buschmann et al. [2007]. The collection of ideas thereof, led to the conclusion that a synthesis of *context* is required in software engineering, expressing risk imposed and opportunities offered within the context of each system.

1.4.2 Part II - Contribution

The second part of the thesis (Part II) is formed by two chapters, Chapter 3 and Chapter 4. Chapter 3 discusses context on the level of individual elements, using the *Context Dynamics Matrix* (CDM), a novel approach to model context. Chapter 4 discusses context of individual elements within a system using Data Flow Diagrams (DFD). CDM is used to enrich the context representation of system elements' contexts as represented by DFD.

1.4.3 Part III - Discussion and Conclusion

The third part (Part III) is formed by two chapters. The first chapter, Chapter 5, is the proof-of-concept that describes an industrial case-study to demonstrate the efficacy of the approach. The second chapter, Chapter 6, I summarise the work presented in the thesis and present a survey of related work. I also discuss

limitations and propose future research directions to evaluate and further develop the use of context in software engineering.

1.5 Publications

The contributions made by this thesis are based on the results of preliminary research, some are presented in Chapter 2, and published in the following refereed conference papers.

- Z. Alshaikh and C. Boughton. The Context Dynamics Matrix(CDM): An Approach to Modelling Context. 16th Asia Pacific Software Engineering Conference (APSEC 2009), 2009.
- Z. Alshaikh and C. Boughton. Context centralised method for software architecture: A pattern evolution approach. In 3rd International Conference on Software and Data Technologies (ICSOFT2008), 2008.

1.6 Summary of contributions

The research reported by this thesis makes the following contributions to the existing knowledge of system and software development.

- **A synthesis of context**. A review of context in literature, produces a synthesis of context as two dimensions: influence and perception. The synthesis is based on five themes drawn from the literature that sets the plan for the following chapters to represent the context of systems.
- **Context models**. A novel approach to synthesise and represent the context of elements in terms of context states using CDM. The context synthesis introduced in Chapter 2, is expanded by a force model of influence, and a knowledge model of perception. Both models of context are represented in CDM, which implies sources of system variation when applied to software/system requirements.
- **Context mapping**. To represent the context of systems, the context states of the CDM are mapped to the description of requirements using DFD. The functional view that DFD represents is extended by context states. Thus it is possible for analysts to represent the context of a system as represented by requirement statements, through a context enriched DFD model.

Chapter 2

Background

> In analysis, something that we want to understand is first taken apart. In synthesis, that which we want to understand is first identified as a part of one or more larger systems.
> In the second step of analysis, an effort is made to understand the behavior of each part of a system is taken separately. In the second step of synthesis, an effort is made to understand the function of the larger system(s) of the which the whole is part.
> In analysis, the understanding of the parts of the system to be understood is then aggregated in effort to explain the behavior or properties of the whole. In synthesis, the understanding of the larger containing system is then disaggregated to identify the role or function of the system to be understood.
>
> Ackoff [1999]

Contents

2.1 Introduction

I introduce early attempts to explain sources of system variety, which started by conducting preliminary research, followed by a literature review on context, which led finally to providing a synthesis of views from the literature. Preliminary research was conducted by way of a thought experiment concerning different ways to vary a satellite system. The experiment led me to recognise that within every system's context there exists degrees of influence that either impose certain conditions as imperatives, or vary the system because the influence provides choices. Because the source of such influences is the context of the system, a literature review was conducted on the current state of thinking on 'context,' in software engineering, and other disciplines areas. The review confirms the importance of context to model requirements, and to shape software architecture and design. Other disciplines, such as building architecture, have pointed out that context is the source of forces that shape architecture form. But the literature review has also exposed some views that point to the importance of perception as another dimension of context. This led to conclude that there is a need to synthesise both views, and use them to expose different levels of influence and perception to explain when it is possible to vary the system and when it is not. Thus, the last section of this chapter formulates a conjecture that summarises this novel thinking about 'context,' which becomes the basis for the work developed in later chapters.

2.2 Preliminary research

> A system reflects the organisational structure that built it.
>
> Conway's Law [Endres and Rombach 2003, Law L16, pp81-82]

As part of the preliminary research, a thought experiment was conducted on how to vary the architecture of a satellite system. Conducting the experiment was motivated by the need to explain *why* systems vary on the level of requirements and architecture. The experiment builds on knowledge obtained from previous research [Alshaikh 2006], where three satellite system examples were explored to extract a *unified model*. A unified model is represented in The Unified Modelling Language (UML) that represents similarities among selected systems, which are also modelled in UML. But unified models do not represent variety, as they only capture similarities. Thus, the thought experiment to explore similarities as well as differences, similarities as imperatives and differences as variations. The aim of the thought experiment was to explore those imperative elements among all

satellite systems to understand why they share those particular elements, and consequently, why they vary. Thus, imperative elements are elements that cannot be changed, representing a source of limited variation, and optional elements that can be changed represent sources of variation.

In exploring different ways a satellite system may vary, one imperative element that emerges is that each system is separated into a ground system and a flight system. This separation is a key attribute of every satellite system. The system may vary based on orbit: low-earth-orbit (LEO), micro-satellites, geostationary, deep-space-missions; based on missions: remote sensing, space imaging, space exploratory missions; but what remains unchanged within each of the aforementioned variations is the separation into a ground system and a flight system.

Given this imperative element, is it possible to identify an imperative architecture as well? Put another way, what are the requirements and architecture elements that are chosen and other elements that are imposed as a result of this element? In the thought experiment, choices always seem to lead to one variant of a client-server style.

Because the satellite is separated from the ground station, the separation becomes a requirement/constraint that cannot be changed. But what is the source of such requirement? Such a requirement is derived from the nature of the system itself, and from the conditions that govern its operations. Unlike other elements that the system may share with other systems, if the element of separation is changed, the system would cease to be a satellite system.

But not all systems have the same imperative. In an inventory tracking system, for example, it is possible to vary the architecture across different styles following the variation of all possible systems. The system could follow one variant of a client-server style—an online system—or choose a completely different style such as a purely layered architecture. The difference between the satellite system and the inventory tracking system, is that a satellite system is linked to a fixed imperative element that cannot be changed—that is, the separation between ground station and the flight system. But for the inventory tracking system, conceptually, there is not a similar limitation that applies to all systems of that same domain. Unlike satellite operations, 'tracking inventory' is largely irrelevant to the physical conditions in which it operates. This does not mean that the inventory tracking system is not exposed to influencing factors/forces that are specific to its local environment that might lead to a specific architecture style. A remote inventory tracking system, for example, is similar to the satellite system, because both operate remotely. As a result, imperatives can also vary within

the same system types, such as moving from a local tracking system to a remote tracking system, or from a geostationary satellite to a low-earth-satellite systems.

Comparing the satellite system and the inventory tracking system suggests that both have different system narratives. A narrative describes the system's conditions, aim, environment, and so on. But within each narrative it is possible to identify an imperative element that cannot be changed. Therefore, the source of this imperative element is the system narrative, or alternatively, its *context*.

By following the thought experiment on the satellite system, changing the narrative every time meant that the context was changed. Each time the context is changed, the system may have a different imposing element, at the level of requirements, system structure, and so on. As a result, within each context there are differences between each context and the other. Such differences are the main source of variation. Therefore, a system may vary depending on how its context allows it to vary. Then the ability to vary system needs or structure is directly related to the degree of influence each context imposes on the system. The thought experiment, then, leads to the following observation:

> *The choice of a system's requirements and architecture is limited by the 'degree of influence' a context has on the system.*

But further analysis of the satellite system revealed another aspect, that context also changes over time. The satellite system has two different contexts, before the satellite's launch and after its launch. Looking at both contexts as part of one larger context would mean that the larger context has two states. Within the larger context, even before launch, the satellite is separated from any system that may communicate with it as a ground system, thus the element of separation does not change. But it is possible to identify elements that become imperative only after the satellite system is launched, not before, such as the satellite's orbit. The satellite's orbit is decided before the launch of the satellite as part of designing the mission, in which some satellites will not be able to change their orbit after launch. But because the satellite is not in orbit, it could be changed at anytime. For example, a low-earth-orbit (LEO) satellite leaves each ground station a relatively short communication window to connect to it as it orbits the earth several times a day. Therefore, when the orbit of the satellite is chosen and the satellite is launched, its orbit cannot be changed, thereby adding a new imperative. What this means is, that context elements may behave within predictive states, where in one state it does not influence the system strongly, but as it changes state its influence on the system starts to change as well.

The thought experiment leads to two main conclusions:

- That systems are the result of their context. But they owe their ability to vary, to the degree of influence that their context exerts in different states.
- That systems may vary over time as the state of influence of their context changes. Therefore modelling the behaviour of a system context, by modelling states of influences, offers system analysts a way to plan how to vary systems ahead of time.

2.3 What is 'context'?

> . . . don't ask for meaning, ask for the use.
>
> Wittgenstein [1974]

The word *context* originates from the Latin verb *contexere* that means to weave together. Weaving together directs the attention to an important aspect of the meaning of context, that context weaves elements. Therefore, other related terms are typically used: environment, circumstances, conditions, state of affairs, setting, frame of reference, and factors. But if context by definition means 'other things,' then how do things move from being *in* context to be *out* of context?

Scharfstein [1989] provides a definition and a solution at the same time by defining context as: "*that which environs the object of our interest and helps by its relevance to explain it*." By this definition, Scharfstein distinguishes three elements of any context: an object, relevance, and purpose (to explain). Therefore, elements move in and out of context in relation to each other. Therefore, context according to Scharfstein is relational within a specific purpose: to explain, to describe, to design, and so on. But by using the word 'environ,' the definition does not limit context to specific types of elements, either tangible or abstract. Context, then, is still open to a wide range of possible elements/circumstances.

Similarly, Alexander [1964] recognised that the context of an ensemble is an element of design that cannot be fully described, because attempting to produce a full description of context is an endless task. But Alexander approaches this problem by limiting context by *force*. According to Alexander, what is relevant is the 'force' (influence) of the context on an object of design. The result of this 'force' is undesired outcomes, or what he calls misfits. Alexander is not concerned with the definition of context per se, similar to Wittgenstein's (1974) approach that emphasises the use of context instead of asking what it means; Alexander is rather concerned with the effect(s) of context.

By comparing Alexander's approach to Scharfstein's, they agree that context

has to be limited for analysis. To Alexander, what is relevant to 'form,' is context and its 'force' causing 'stress' on the ensemble, to Scharfstein, it is explanation. Both definitions, however, create serious difficulties in approaching the concept of context. Alexander through 'force' ties context closely to 'form' thereby creating a duality between context and form—Alexander refers to it as context-form. Scharfstein definition, on the other hand, leads to relativism, which at its extreme, does not help to explain anything [Scharfstein 1989].

Other approaches have recognised the importance to limit context, but they have tended to apply very stringent limits. Dey [2001], in context-aware systems for example, limits context to 'information'. In linguistics, Halliday [1977] divides context into three elements: field (e.g., activities), tenor in the form of the relation between participants, and mode (e.g., written or spoken). Based on this model, Halliday founded the *Systematic Functional Linguistics (FSM)* approach to semantic analysis. Fetzer [2004] provides a sociocultural definition of context comprised of individuals' physical, physiological placement, knowledge, and intention. Goodwin and Duranti [1992] follow a similar approach in anthropology, where context is divided into elements: the setting or the physical world, knowledge, language, and non-verbal signs. In all of these approaches context is a reflection of the concerns of the discipline (linguistics, anthropology, and so on).

So what is left out of the analysis when context is limited by such parameters? According to Ackoff [1999], system problems in the real world are not solved by any single discipline. Thus such approaches that limit context by the discipline promote building artificial boundaries that do not exist in reality. Therefore, van Dijk [2008] recognises that context should become a multi-disciplinary problem. Therefore, what the definition of Scharfstein [1989] and Alexander [1964] suggest, is that context is a product of relevance. That is, whatever is part of the problem should become part of the context, and should be included in the solution. If the objective is to understand, then context is whatever provides an explanation; if our objective is to design, then context is whatever results in comfort, robustness, safety, and so on.

What I mean by '*whatever*,' is that context cannot be listed by fixed elements. Thus, context is not only in the elements themselves that surround an object of interest, manifested in its attributes, but what happens as a product of two or more elements. Consider the example of an architect that plans to design a house near a tree. If the architect holds the view that the context is what surrounds the object of interest, in this case the house, he/she might only consider what is relevant to the house, what helps to make the house feel better, the view from inside more pleasing, and so on. But if the designer holds the view that the context is the product of the house and the tree together, the aim of designing that part of

the house beside the tree will change. The design will not only be concerned with how the house looks beside the the tree, but how the tree looks beside the house as well, how the house and the tree enhance each other's beauty. This way the architect needs to consider the attributes of both elements of the design and how they interact when they are put together. An architect can consider the house only, but the result is very likely less satisfactory.

From the views introduced about context so far, it is possible to recognise that context is either without clear limits, or over limited by preconceived views. Without a limit to context, context becomes 'a theory of everything' [Dijk 2009], and with a strict limit it becomes a special theory that only represents part of reality. My answer, then, to the question 'what is context?' is to provide a model to represent context, rather than a theory to explain it. A model that provides a way to describe how and when things move in and out of context. Such a model of context should not provide descriptions based on relevance alone, but based on significance. It should also guide us to use 'context' effectively, regardless of the question of meaning.

2.4 Context in software

In software engineering, little attention is directed to provide a formal definition of context beyond the synonyms introduced earlier. Thus the discipline's approach to context may be interpreted by examining how modelling approaches *used* context to solve system problems. By examining modelling approaches two themes could be identified: realising context as the boundary of the system, and realising context as common-sensical. Therefore, following Wittgenstein's advice, to ask about the use not the meaning; a survey is presented on the use of context in current approaches to software development, on the level of requirements, architecture and design.

2.4.1 Context in requirements

Context in early requirements approaches is typically associated with the task of setting system boundaries, but in later approaches context became identified through the narrative of scenario-based requirements. In setting system boundaries, the term 'context' is used explicitly to develop context diagrams in structured analysis approaches, for example. Later, with the emergence of object orientated analysis approaches, the use of the term 'context' became less common. But context as a concept, continued to be used implicitly within the the scenario-based requirements, or what could be identified, following Scharfstein [1989], as a

	Abstractionism	**Contextualism**
Role of description	Abstractions are powerful and general	Particularities are as informative as generalities
Design criteria	Design integrity	Contextual fit
Origin of requirements	Prescriptive recommendations	Current practice
Role of users	Management	End-users
Community of practice	RE and software engineering	Computer Supported Cooperative Work (CSCW) and Human Computer Interaction (HCI)

Table 2.1 – Key discriminators between abstractionism and contextualism by Potts and Hsi [1997].

common-sense approach to context.

Similar to what Scharfstein [1989] suggests, that we are more able to understanding context in practice rather than theory. The scenario-based approach does not formally recognise the aim of identifying the context of the system as its aim, analysts may resort to their common-sense instead to from a sufficient understanding of the context. Efforts, nonetheless, remained directed towards identifying contextual requirements formally. For example, Potts and Hsi [1997] show differences between abstraction approaches and contextual approaches—as Table 2.1 shows—and call for a synthesis between the two approaches, combining the best features of both.

In what follows a review of the early uses of the term context in structured analysis, as part of setting system boundaries, are reviewed. This is followed by a review of the common-sense approach to context that became popular in last two decades.

Context as boundaries

The work of DeMarco [1979] is perhaps the first explicit use of context, as a concept, in software requirements. By setting the boundary of the system as its context, DeMarco's approach abstracts data inputs and outputs, and represents data flow going through a series of processes, which ultimately forms a Data

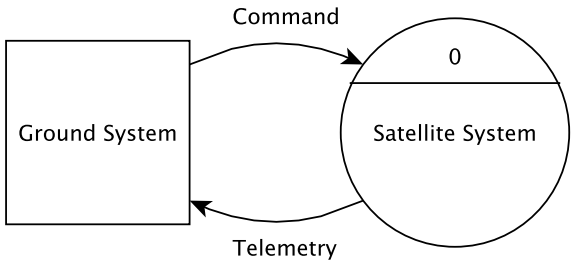


Figure 2.1 – A DFD context-diagram of a satellite system.

Flow Diagram (DFD). The set of data, data flow, and data processes, help to understand the interconnected processes/processing at different levels within a system. The approach depicts a high level view of the system in the context diagram—representing level zero—by identifying the scope and boundary of the system, and the system's interaction with external entities. Figure 2.1 shows an example of a DFD context-diagram of a satellite system interacting with the ground system as its external entity. The process of identifying boundaries and managing scope is a difficult task that demands a series of refinements and revisions with the involvement of stakeholders [Yourdon 1989]. It is the purpose of the context diagram to show relevant system terminators or external entities that interact with the system, and show data flow between the system and the external entities.

In order to focus on identifying important processes and data flow within DFDs, the context diagram does not include any information that either indicates the frequency of interaction between the system and an external entity, or the size/amount of data being transferred that has an effect on the quality of service. Hence the context diagram is a device for setting system boundaries and identifying interacting external entities, ignoring any other requirements related to quality or constraints. In some cases, to understand the context diagram better, an event list is constructed to better describe stimuli and responses of the system, allowing a more dynamic view. Real-time extensions to DeMarco's approach were made separately by Ward and Mellor [1986] and Hatley and Pirbhai [1988], whereby, for the latter, the concept of response times was included with the frequency of system inputs and outputs. Both extensions include the concept of event/control flows as separate from data flows. Both methods, however, also retained the context diagram, but with the inclusion of event/control flows.

But the use of context diagrams is not limited to representing data flow. Jackson [1995b] uses context diagrams to describe the relationship between the 'machine' (system) and the 'world' or the application domain. Unlike DeMarco's

context diagram, Jackson's diagram does not show any description of the interaction(s) between the system and other elements within the application domain. But beyond the representation of context as boundaries in the simple sense, Jackson expresses his understanding of context by using the example of building a bridge. In the bridge example, Jackson describes the importance of understanding the problem domain (context), where the engineer is able to see and feel the environment directly:

> You're an engineer planning to build a bridge across a river. So you visit the site. Standing on one bank of the river, you look at the surrounding land, and at the river traffic. You feel how exposed the place is, and how hard the wind is blowing and how fast the river is running. You look at the bank and wonder what faults a geological survey will show up in the rocky terrain. You picture to yourself the bridge that you are going to build [. . .] [Jackson 1995b]

Yet, Jackson points out that the context is not the problem but what surrounds the problem. However, he does not discuss what should be considered as part of the context and what should be excluded. Alternatively, Jackson extends the context diagram using problem frames. Problem frames, here, are similar to extending the context diagram by processes in the DFD approach. The use of 'frames,' is also common as another term used to mean context, as in the work of Goffman [1975] in sociology, for example. Accordingly, Jackson starts from the problem domain to appreciate the context as a whole, then begins to limit the context by using frames, capturing relevant elements to the problem and application, but on the way, points to certain difficulties:

> The difficulty arises from the relationship between the machine and the world. The machine will furnish the solution, but the problem is in the world. Discourse about the problem must therefore be a discourse about the world and about the requirement that our customer has in the world. Since the world is very multifarious we should expect to find that there are many different kinds of problem. Controlling an elevator is not at all like compiling source programs, which in turn is not at all like switching telephone calls; and none of them is like processing texts in a word processor. Jackson [1995a]

Jackson [1995a], then, expresses clearly that the problems of the world are different, in structure and in behaviour, from the problems of the machine. This is why, Jackson explains, we need domain experts. The use of problem frames and the concept of machine-world of the application's context, represent a serious attempt to address the issue of context, as a separate element within the process of system development, extended further to design within a larger framework, see for example Hall et al. [2008].

	Boundary	Common sense
Terminology	*Is the term 'context' used?*	
	Only at the beginning of the analysis.	Does not use the term.
Layers	*Is context layered?*	
	Represents context in terms of layers.	Does not describe the system in layers.
Relations	*Is context connected?*	
	Maintains related elements connected.	Does not connect its elements formally.

Table 2.2 – A summary comparison between the boundary approach and common sense approach to context.

In Object Oriented Analysis (OOA), use-cases describe a sequence of interactions between the system and external entities as actors [Wiegers 2003]. Even when the term 'context' is not used, similar to context diagrams, use-cases represent interactions between the system, as internal entities, and its context, as external entities. The same concept of the interaction between internal and external system elements underlies most approaches to system analysis.

For example, Executable UML [Mellor and Balcer 2002] uses a similar approach in sequence diagrams and collaboration diagrams. Unlike the (DFD) context diagram, however, sequence diagrams and collaboration diagrams are not used in the early stages of analysis, but rather in conjunction with the more detailed system/class state models [Mellor and Balcer 2002]. The driving force to adapt this method is the need to understand what the user intends to do with the system, rather than asking for what the user wants or receives from the system [Wiegers 2003]. Thus, focusing on the user is part of the user centred design approach, first introduced in Norman and Draper [1986], and later elaborated by Norman [1988]. Unlike Jackson's (1995b) focus on the world and the machine as the context, user centred design shifts the context to the user. The analyst is no longer interested in the external world only, but also in the internal world of users, their intensions, their emotions [Norman 2005], and mainly their mental model [Norman 1988].

Context as common-sense

Due to the focus on the user the common-sense approach to context emerged. Within such an approach requirements are described in a narrative that focuses on the user's interaction with the system. In such a narrative, context is not identified explicitly, but by relating to the user's situation, to which the analyst is able to relate.

For example, Potts [1995] links user intentions to organisation goals using scenarios. Thus, the scenarios' narrative captures user intentions within individual tasks to achieve the system aim. Each scenario has a description of a single instance of an interaction with the system [Wiegers 2003]. According to Wiegers [2003], describing user aims and intentions, rather than what is needed from the system, is a response to the need for designing software to enhance usability. Similar to Jackson's (1995b) bridge analogy, where the engineer imagines the bridge to be part of the scene, the analyst using scenarios can also imagine the users' situation. Such an approach transcribes local system contexts onto the level of users.

The scenario-based technique is used also to describe non-functional requirements, where the concern about users actions is replaced by the concern about quality attributes. For example, quality scenarios by Bass et al. [2003] use general and concrete scenarios to describe non-functional requirements. In general scenarios the narrative does not relate to a specific system. For instance, a general scenario may be to secure data from unauthorised users. To make the scenario concrete, the scenario must specify which data and which level of security. For example, a general scenario might state that 'financial transactions must be fully secured,' a concrete scenario, however, would state that 'credit card details must be secured from all users 99% of the time'. When analysts move from a general to a concrete scenario they have to provide additional information about the specific context where a function or a task is performed. The implications of such information, with other similar statements, is left to the reader's judgement to decide how it may influence system decisions.

Although scenarios-based techniques are widely adapted by a number of approaches [Bengtsson and Bosch 1998, Gheorghita et al. 2009, Kazman et al. 1994; 2000], they represent techniques that address narrow system concerns, and lack uniformity. For example, Ralyté et al. [2010] argues that one scenario approach is not enough, and recommends to integrate multiple scenario-based methods to enhance their ability to represent multiple concerns. Another example is the work by Kazman et al. [2005] that introduces an approach to select the appropriate architecture scenario-based approach from a wide range of choices.

Finally, Table 2.2 shows a summary of the comparison between context as boundaries and as common-sense. Each point is discussed as follows:

- In structured analysis techniques, context is not limited to the beginning of analysis. It is used to set system boundaries at the start of system analysis, but continues to be used for the rest of the analysis without using the term 'context'. For example, when the context-diagram is depicted to build a DFD diagram, it represents all that is relevant to the problem at hand. To obtain further detail, the context-diagram is decomposed into a set of processes each of which influenced by some element of the system context. Each individual process that emerges as a result of the composition, sets its own boundary with other processes of the same level. The break-down of processes continues for several levels.
- In the boundary approach, as in DFDs, the system is represented in terms of layers. But the common-sense approach does not recognise context levels when scenarios are used. The only example of layered scenarios is the use of general and concrete scenarios. But general and concrete scenarios do not represent two layers of context, but rather two degrees of abstraction.
- The boundary approach to context represents connected system elements, as in processes in DFD. But the common-sense approach does not focus on connecting its elements, as in scenario-based requirements. Each scenario, in principle, maintains its autonomy, and may be enhanced by specific techniques to connect scenarios.

2.4.2 Context in architecture

A number of approaches to software architecture realise context, but without necessarily adopting the term, such as: Architecture Tradeoff Analysis Method (ATAM) [Kazman et al. 2000] and Attribute Driven Design (ADD) [Wojcik et al. 2006] among others. But it is possible to classify the use of context in architecture in two forms: context as state, context as boundaries.

Context as state in software architecture is realised when an architecture is either already decided or exists in a running system. According to Kazman et al. [2005] the context of an architecture is derived from three elements: organisational goals, the system state, and constraints. In the review of software architecture analysis methods, Kazman et al. stress the importance of realising context as the first criteria for choosing a software architecture analysis method. Although not identified by Kazman et al. in this manner, the context of an architecture is derived as a state based on the three elements combined. Thus, an analyst must

examine the state of the context of the system's architecture before choosing an analysis method. For example, if an architecture is chosen as a result of a specific constraint, the state of the context changes as the constraint is removed.

Context as boundaries are identified in the traditional sense similar to what is defined by the structured analysis approach, or by defining what is relevant to the architecture in general by setting a conceptual boundary. Bosch [2000] recognises context as boundaries in the traditional sense. Where the context is identified for an architecture the form of interfaces between internal and external entities. Whereby context plays a role in defining functional and non-functional requirements for each architecture interface. This is similar to context-diagrams, where the system is defined by a boundary in relation to external entities. Another approach is to define what is relevant to the architecture by setting conceptual boundaries. For example, the context of software architecture requirements is set according to quality. Bass et al. [2006] only recognise requirements that have an impact on quality. This approach sets a boundary based on the concept that what is relevant to architecture is to achieve quality. As a result, the concept is used to decide what to include and exclude as part of the analysis. An architect then uses this concept and selects from requirements what elements fit a predefined classification. But it is not clear how to determine when a requirement is or is not significant. According to Bass et al. [2006] determining such requirements are based on experience and judgment.

There are efforts in the area of software architecture that arrive at more precise characterisation of what is relevant to the context of architecture. For example, Bass et al. [2006] suggests the following classification of system elements that have architecture significance: *a*) quality attributes; *b*) volume of functionality; *c*) similar requirements for a family of related systems; *d*) choice of technologies; and *e*) deployment and operations. It is possible to consider this classification as a start to move away form the concept of setting system boundaries to a more property type classification. In an attempt to identify common sources of influence within the architecture context, further work by Bass et al. [2008], Kazman and Bass [2005] seek to identify common quality attribute scenarios in relation to business goals empirically. Although the study of Bass et al. [2008] failed to find a correlation between system domains and identified quality attributes, the study represents an important attempt to categorise the context of architecture based on empirical data.

Some representations of architecture context still use traditional analysis artefacts, such as the context information diagram [Taylor et al. 2010] and system context diagram [Fairbanks 2010]. But other approaches represented context elements differently. Bruin et al. [2002] use feature-solution graphs (FS-graph) to

represent architecture knowledge. The FS-graph combines two spaces: a feature space and a solution space. The graph represents architecture knowledge as connections based on stakeholders' views, in which the latter is (part of) the context. Another approach is presented by Clements et al. [2002] using utility-trees. Utility-trees capture important quality attributes and architecture risks according to stakeholders' classification under the guidance of system developers. Developers use utility trees to capture stakeholders' decisions while applying the ATAM approach. A utility tree assigns to each quality attribute, concrete quality attribute scenarios. Stakeholders prioritise each scenario according to two dimensions: importance level, and perceived risk of achieving the scenario.

Both FS-graphs [Bruin et al. 2002] and utility-trees [Clements et al. 2002], represent a significant departure from traditional system analysis methods. Both approaches capture contextual knowledge of the system and part of the application domain—Bass et al. [2008], Kazman and Bass [2005], for example, used data obtained from architecture evaluations using utility trees in their study of quality attributes and system goals. But both approaches are narrowly focused. For example, utility-trees only focus on quality attributes without extending the focus to the other five areas identified by Bass et al. [2006]—that is, constrains and similar requirements. The success of utility-trees should promote using them for concerns other than quality attributes.

2.4.3 Context in design patterns and pattern language

Design patterns are based on building patterns introduced by Christopher Alexander [1977], deriving a language for design from common designs of houses and cities. The concept of context and its relation to patterns is borrowed largely from Alexander et al. [1977]. But software patterns have not matured enough to form a complete pattern language that developers can use to design software systems completely based on patterns [Gamma et al. 1994]. In a speech addressing the patterns community, Alexander [1999] mentioned that design patterns of software lack two attributes: they do not work together to solve multiple design problems, and they do not aim to improve human life. While the latter is improved by the recent developments in the use of patterns in usability, see for example Dearden and Finlay [2006], the former remains a challenging issue—see for example John et al. [2009] on the failure of integrating a detailed pattern representation to an overarching architecture in an industry context.

Although each building architecture pattern by Alexander et al. [1977] has included a description of the context before introducing the pattern, this strategy was not followed in determining software patterns. For example, patterns by

Gamma et al. [1994] replace context by stating intent and motivation. Buschmann et al. [1996] introduce a pattern through a short sentence as the context of the pattern, then describe the problem. For example, in the pipe-and-filter pattern, Buschmann et al. [1996] summarise the pattern's context in a single sentence as: '*processing data streams*,' followed by an example. Both approaches seem to share the same concept, that the pattern's context is manifested in its goal, based on how the designer intends to use it.

Recently more attention is given to rethinking the role of 'context' in patterns. Buschmann et al. [2007] argue that context descriptions must be precise, and designers must avoid general descriptions that can be easily omitted. Buschmann et al. gives the BRIDGE pattern as an example of how designers apply it to the wrong context as a result of an imprecise context description. Buschmann et al. [2007] also discuss the problem of context in relation to a pattern, whether context is part of a pattern or not. This discussion seems to be a boundary issue similar to what analysts would make when they draw a context-diagram, the question then becomes what is part of the system and what is not. But to follow the concept of a pattern language by Alexander et al. [1977], the context of a pattern should be another pattern within the language, or a choice of patterns.

Alexander [1979] points out that patterns do not exist without a pattern language. In which the whole, represented by the language, gives meaning and purpose to the part [Alexander 2002]. The context of a pattern then is the system as a whole, or other patterns within the design. Smaller patterns should be integrated with larger patterns within a pattern language. Instead of describing the context of patterns, Alexander's approach models context by other patterns. This is exemplified at the start of each pattern in Alexander et al. [1977], where each pattern refers to other patterns that may be used by it. The structure of the language is formed such that smaller patterns fit within larger patterns—the order of patterns in Alexander et al. [1977] is nested, starting from larger to smaller patterns, thus emphasising this concept.

Recently, some observations have been made on the level of detail needed to describe a pattern in relation to its context, which confirms the recommendations made by Alexander et al. [1977]. For example, John et al. [2009] observes that even if a pattern is identified, it is not possible to ignore other components within the larger context that influence or are influenced by how the pattern is to be structured, even if those components are not essential to the pattern:

> Although a particular UML diagram can be drawn, doing so necessitates depicting and arranging components other than those that are essential to the pattern and thereby impose themselves on the architecture designer. [John et al. 2009]

Therefore, the implementation of a pattern is limited by the level of description detail. As long as the pattern is described for a particular context, the more the pattern fits a particular context, harder it becomes to apply without modifying it. Gamma and Beck [2010] make a similar observation on patterns' density:

> Designs with high pattern density are easier to use but harder to change. We have found that such a high pattern density around key abstractions is common for mature frameworks. The opposite should be true of immature frameworks—they should have low pattern density. Once you discover what problem you are really solving, then you can begin to "compress" the solution, leading to a denser and denser field of patterns where they provide leverage. [Gamma and Beck 2010]

The concept of context in patterns and pattern languages is limited to single patterns, manifested in each pattern's description. The observations of Gamma and Beck [2010] and John et al. [2009] suggest that implemented systems have a significant influence on the implementation of patterns. They also suggest that the context of a pattern is both external in the form of purpose, and internal in terms of system's components, which forms a mix of a system of patterns and non-patterns elements. Thereby, it is important to regard context more fundamentally when addressing design problems. To answer the question posed by Buschmann et al. [2007]—whether context is part of the pattern or not—context is both part of the pattern and is the pattern. Context is all there is.

2.5 Context in other disciplines

Morris [1999] comments on Scharfstein's (1989) definition—context "that which environs the object of our interest and helps by its relevance to explain it"—that context goes beyond meaning in the simple sense of the word, to the extent that the traditional delimitation between analysis and interpretation is broken.

Context manifests itself in many fields of knowledge, with varying prominence. A number of approaches recognise context not only as a problem, but also as a solution. In one sense, context regresses endlessly posing a challenge to the intellect, in another, it is rather natural and we are more able to use and deal with it in practice than in theory [Scharfstein 1989]. In anthropology for example, understanding phenomena through social and situational context led to a paradigm shift in the discipline [Scharfstein 1989]. Another similar shift resulted from the challenge against the universalistic nature of evolutionary theories in the late nineteenth century and early twentieth century [Scharfstein 1989]. In linguistics, the acceptance of the pervasive nature of context-sensitivity in natural

language created a movement that opposed the view of language as a formal system represented in context-free grammars [Dilley 1999]. The movement led to formal semantics, which as Recanati [2004] puts it: "... [is] a very active discipline whose stunning development in the last quarter of the twentieth century changed the face of linguistics".

Although the concept of context demonstrated an important role in the development of several fields of study, embracing the concept itself, as will be shown in sections to follow, comes with difficulties of its own. Therefore, a discussion is presented from a number of disciplines that focus primarily on context as a course of study. The discussion first starts by exploring *context* as a problem, then *context* as a solution, and finally *context* as 'form.'

2.5.1 Context as a problem

The problem of context, according to Scharfstein [1989], lies in the intellectual burden that it puts on one's comprehension capacity. A burden so heavy, it destroys the understanding it is supposed to enhance. Dilley [1999] points out that *context*, in its own right, is difficult. It is linked to equally problematic issues, like 'meaning' and 'interpretation'. Another aspect of the problem stems from the fact that context is both an abstract and general concept, shared by multiple disciplines, making a complete theory of context is a transdisciplinary enterprise [Strathern 1987]. Thus, the problem of context is addressed according to three main analytical difficulties: the regression of context, the shell problem, and the problem of relevance.

The problem of regression: regression is the result of attempting to contextualise an element through other elements that are also contextualised by other elements, in which this contextualisation process continues indefinitely. For example, Harvey [1999] came to realise, in analysing the immediate social context of linguistic interaction, that questions about origin/identity (who people were), questions about utterances (what is being said and in what language), and the situational setting (what is the occasion), were outcomes with prior origins, not starting points. Harvey explains the complexity of such an analysis in a bilingual culture, in which one language implicitly stands as the context for the other, forming an *implicit* context, leading to context regression.

In social science, there are three approaches to the regression of meaning outlined by Dilley [1999]: external context, internal context, and a mental context. External context regresses outwardly, where meaning is obtained from the external world. Internal context is based on language or text being the source of meaning, and that nothing exists outside of text. Mental context exists in the mind, as part

of an internal intention or a psychological state. It is possible to regard abstract ideas as part of this context. But none of these approaches—or as Dilley [1999] calls them contextual moves—provide a solution to the regression problem. In each of these approaches/theories, arguments are made to limit context either internally or externally. For example, when Derrida [1998] states that '*nothing exists outside of the text*,' he excludes the influence of the external world. Because the producer of text is separated from the text itself, Derrida argues that the producer cannot be replaced by any interpretation or reasoning of our making in the producer's absence. The result is a closed system of text. That is, text always refers to other texts, and truth and meaning only exists within the text. For example, typically in order to know about the producer in a way to know more about the context of the text, the knowledge is obtained from other textual sources: memoirs, other texts produced by the same producer, and so on. Any meaning derived from outside the text itself is doubtful. With this argument, Derrida has delimited context from regressing externally to the world, to regress internally in the text.

The shell problem: is the result of attempting to be thorough in understanding context. Thereby leading to total contextualisation, where everything becomes the context of everything else. In this case a 'twist' of context occurs, where the parameters of the problem are turned inside out [Scharfstein 1989]. This issue is referred to as the shell problem. The shell problem results from the context becoming the new problem, while the old problem, or its contents, become the new shell or context [Dilley 1999]. In interpreting text, for instance, a similar problem occurs—in what is known, according to Ricoeur in [Dilley 1999], as the hermeneutic circle—where starting from the text to understand the context leads to using the context to understand the text. Text is bounded by meaning and meaning is bounded by context, yet *context* is boundless. Whereby any definition of context can itself be contextualised by means of a new context, and the process is open to infinite regression [Dilley 1999].

The problem of relevance: Scharfstein [1989] draws our attention to the issue of relativism, as another philosophical difficulty resulting from the reliance on context. The dependence on context is a kind of limited relativism, whereas relativism, itself, is hard to limit, referred to as the 'unboundedness of context' [Dilley 1999]. Contextualism leads to extreme relativity, which consequently leads to extreme individualism [Scharfstein 1989]. By recognising that each individual case has a unique context, it is possible to justify anything.

But the use of relativism relates to limiting context in a way other than limiting its regression. According to Culler [2009], *relevance* limits the contents of context,

since context's contents have no limit. Culler gives examples of major shifts in reading literary texts as a result of redefining what is relevant:

> Major shifts in the interpretation of literature brought about by theoretical discourse might, in fact, be thought of as the result of the widening or redescription of context. For example, Toni Morrison argues that American literature has been deeply marked by the often unacknowledged historical presence of slavery, and that this literature's engagement with freedom [...] should be read in the context of enslavement, from which they take significance. [Culler 2009, pp92]

The problem, then, becomes how to identify what is *significantly* relevant from what is not. For Derrida [1998], deconstruction allows choosing relevance within the text as part of a critical reading process where the text is turned against itself. By limiting the critical reading to text the reader may perform relevance play, experimenting with what maybe relevant. Choosing relevance is to choose a referent—textual in this case—to reveal weaknesses, hidden themes, and other indications that are exposed by every new reading. This is different from how structuralism approaches relevance. Context in structuralism is renewed or changed by the movement of time—events in the external world for example—as the context changes, the text also changes. But deconstruction reproduces meaning in the text with every reading by relevance. Every time a text is read, a new connection to another text—believed to be relevant—produces a new context, hence new meaning. Foucault [2002] argues that what is relevant—or what is believed to be relevant—changes according to the change of knowledge. Knowledge then directs relevance even in the closed system of language presented by deconstructionists.

2.5.2 Context as a solution

While posing an analytical problem, context is also an intuitive solution. Scharf-stein [1989] observes that we are more aware of context in practice than in theory—this relates to what is previously identified as the common sense approach to context in software engineering.

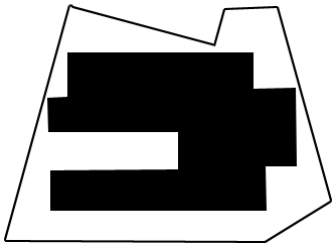
One possible approach to demarcate 'context,' or the domains indicated by context, is to represent context in terms of 'connections'. According to Kristeva [1990], interpretation, in this sense, is an act of 'making connections,' and as a result, disconnections [Dilley 1999]. Such an approach relates the system to its surroundings, it is a result of an interpretation, and by itself, yields an explanation [Dilley 1999]. Therefore context can only be analysed interactively and not disconnected from its application [Harvey 1999].

Wittgenstein [1974] observes that context, or contextualism, comes as part of a 'language game' that people play. It resembles how an agreement about the meaning and the use of a word could be arrived at. Therefore, Wittgenstein is in favour of asking about the use of 'context,' rather than asking about the meaning of it [Wittgenstein 1974]. Malinowski, according to Halliday [1977], derived the term 'context of situation,' alluding to the meaning of words that relate to the culture in which those words are used, a platform, perhaps, where the 'language game' can be played. An example of how the language game and the context of the situation play an important role in any technical discourse is presented by Ozkaya et al. [2008].

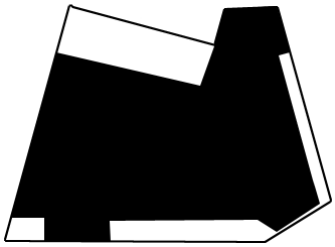
Ozkaya et al. [2008] remark how stakeholders express their quality attributes and concerns (subattributes) via non-standard terms like 'flexibility.' A dialogue with a stakeholder is presented where an analyst, referred to in the dialogue as an evaluator, engages in a conversation about the stakeholder's concerns for a system. When the stakeholder uses the unfamiliar term 'flexibility,' the analyst prompts the stakeholder to explain what is meant by the term. The stakeholder answers: "Well, flexibility has two thrust areas for our product, one from the user perspective and one from the system perspective," [Ozkaya et al. 2008]. Afterwards, the dialogue continues in which the stakeholder is requested to give an example of the 'use' of the term. By engaging in the 'language game' according to Wittgenstein, the analyst is able to arrive at an agreement about the term in question. Such an agreement is reinforced by the fact that both the analyst and the stakeholder share, or perhaps agree to share, a common platform making use of the context of the situation.

Such a platform establishes the term 'flexibility' as a *focal event* [Goodwin and Duranti 1992]. A focal event demonstrates a contextualisation act of the term, in reference to several parameters. According to Goodwin and Duranti [1992] a focal event, or a phenomenon under investigation, can be contextualised through four parameters: social and spatial framework, behavioural environment (represented in gestures or behaviour), language (as context), and the extra situational context (background knowledge and frame of relevance). Dilley [1999] adds two extra parameters: the historical and psychological context. But in order to make effective use of contextual parameters in any analysis, especially given the complexity involved in accounting for such diverse interrelationships, an interpretive conceptual framework of reality must be formulated [Dilley 1999]. Thus, framing context as an object of investigation [Wittgenstein 1974].

In Artificial Intelligence (AI), for example, a couple of contextual frameworks [Beneceretti et al. 2001, Guha and McCarthy 2003] are sought to simplify the knowledge representation of multi-context systems. Similarly, this move to categorise contextual patterns is also adapted by semantic models in linguistics.



(a) *The black shape fails to preserve its frame/context.*



(b) *The black shape preserving its frame/context.*

Figure 2.2 – Context and form in two cases: the 'A' form is not responding to its contextual structure, while the 'B' form responds better to its contextual structure. The two figures are inspired by an example that appeared in [Alexander 2002, pp57-58] of two ways to design an irregular shaped building.

Semantic models help to explain patterns of inter-contextual variability, according to one of three models: the semantic model, the pragmatic model, and the index model [Cappelen 2007]. Cappelen [2007], however, suggest that these explanatory models are not enough. What is needed, given the different areas of discourse such models must explain, is an 'arsenal' of models and data-gathering procedures.

One example of representing context as connections to reduce contextual variability is exemplified in the framework 'Ambiguity Contexts' in AI introduced by Guha and McCarthy [2003]. In this framework, contextual variability is reduced through making connections with other relevant events. The statement 'He went to the *bank*,' for example, is dependent on the meaning of the word 'bank,' either referring to a financial bank or a river bank. Only when another statement is uttered: 'He got *money* from the bank,' is the denotation of 'bank' clarified.

2.5.3 Context as form

According to Alexander [1964], form is the manifestation of context. In which form has to respond to its context, and when it is implemented it becomes part of it. Thus context, in general, is made of an ensemble of forms, inevitably becoming a context for yet another form [Alexander 1964]. In a meeting room for example, the room has to interface with the floor plan of a building taking its form relative to other forms: the office across the hall, the hall itself, the height of the roof, and so on. After the form of the meeting room is finalised, the meeting room becomes the new context for items to be fitted within the room: the meeting table, chairs, etc.

But the context-form regression is a cause/result of several interrelated form-based and non-form-based patterns. The pattern of crowd movement may

influence the way a road is designed between a point of departure and a point of destination. Therefore, Alexander [1964] points out that a pattern of some sort is to be found, or looked for, in the 'earliest' functional origins of a problem (context). Alluding to the idea, that patterns of form are deeply rooted in patterns of life. This inspired the software community to discover such patterns beyond artefacts of design [Coplien and Harrison 2004, Fowler 1997, Grne 2006], and in software-system thinking [Novak and Levine 2010].

What is particularly relevant to form is the demands placed by the context, recognised for example by [Skjeltorp and Belushkin 2004, Thompson 1966] as forces. According to Alexander [1964], form achieves fit by resolving contextual forces. According to this view, it is possible to replace context by force. A force then becomes recognised as a result of its effect(s), or misfits. For example, if a new tool is first used, say a swiss army knife, the user takes notice of signs of irregularities: failing to cut a string because the knife is not sharp enough, the handle is too small or too large, and so on.

But Alexander [1964] uses misfits to address a more difficult problem, how to describe the desired attributes of form within a particular context. Instead of describing all desired outcomes of a developed system—also referred to as fits—Alexander suggests producing a list of undesired outcomes, or misfits. Alexander gives two reasons for listing misfits. First, misfits are more concrete. They are the result of unresolved forces, and when a designed artefact is put into its environment—its context—it is expected that misfits start to arise. As such, Alexander argues, misfits are more concrete, thereby forming an easier way to *perceive* a force. Second, misfits are typically fewer than fits. Alexander argues that when context interacts with form, fit becomes the rule and misfits the exception. For example, listing the number of fits between a coffee table and a person becomes an endless process. But if the table is a bit too high or a bit too low, too large to fit into a door, or too heavy to carry, it becomes easier to observe and describe. For a house, an office that is too small for a large desk, a window too high to enjoy the view while sitting on a couch reading a book. Misfits standout.

Recently, however, Alexander [2002] developed the context-form relationship further by adapting a holistic stance towards form, in which the development of form is ascribed to a 'structure preserving' process, also called the unfolding process. Form, in light of this process, responds not only to its immediate context but also to the context of the form as a whole. Alexander presents examples from nature where this concept has revealed itself persistently in all areas of life. The form holistically, therefore, maintains its structure intact throughout all stages of growth/change, where the parts transform to preserve the whole. Alexander argues that preserving the structure not only serves functional purposes, but it is

also aesthetically appealing.

To demonstrate the structure preserving concept two figures are depicted: Figure 2.2-(a) and Figure 2.2-(b)—inspired by an example that appeared in Alexander [2002]. In the two structures of Figure 2.2, the black shape resembles the 'form' and the white space and the boundary resemble the 'context'. In Figure 2.2-(a), the black shape is not relating to the context. It seems that such a shape is a result of an inward focus, which fails to respond to the changes to its context. In Figure 2.2-(b), the shape reacts to the context differently, where the black shape responds well to it.

2.6 Synthesis of Context

From the ideas reviewed in the literature on the use of context in software systems and other views from other disciplines, it possible to identify some key themes on context. These themes summarise the different ways context is used in various disciplines, and suggest key concepts that any model should consider when it attempts to represent context. Some of these identified themes also confirm the observation made on context as the primary source of system variation through its influence. In addition to the degree of influence context has on a system, perception emerges as another dimension of context. Perception also directs what is believed to be and what is truly influential. Therefore, in order to arrive at a complete model of the system's context, an analyst must perform a synthesis of context based on influence and perception.

I present, first, the main themes derived from literature. Second, I introduce the main contribution of this chapter, which is a synthesis of identified contextual themes in realising context in two dimensions, influence and perception.

2.6.1 Themes from literature on context

The main themes identified from literature are derived from software and other disciplines that suggest that these themes apply beyond the concern of analysing (software) systems. Each theme confirms the notion that context in reality is more sophisticated and more complex than realised in theory. This assumption is reinforced by software engineering examples presented earlier, in particular, the acceptance and use of the common sense approach to context. Table 2.3 gives a summary of the main themes presented here.

Theme	Related sources
Context is a set of connections	Contextulisation is the act of making connections [Kristeva 1990] Data flow connected processes [DeMarco 1979]
Relevance is directed by knowledge	A system of signs is directed by knowledge [Foucault 2002] Knowledge of language directs reference [Cappelen 2007, Guha and McCarthy 2003]
Context regresses endlessly	Extending context beyond boundaries produces new insights [Culler 2009], Contextual moves [Dilley 1999], Context-diagrams [DeMarco 1979].
Context has states	Goals, architecture state, and constraints [Kazman et al. 2005] Connections and disconnections [Kristeva 1990]
Context has influence	A misfit is identified as any stress on an ensemble resulting of the interaction between context and form. [Alexander 1964] John et al.'s (2009) observation on the influence of a system of patterns on individual patterns.

Table 2.3 – Summary of themes that appeared in the literature on the use of context.

Theme 1: *context is a set of connections.* Connections are derived from Kristeva [1990], characterising the process of contextualising/interpreting as a process of making connections and disconnections. This is also derived from realising context not as objects, but as the result of the interaction between elements. Then such interaction is captured in the form of connections. Context is also expressed in terms of connections in the context-diagram and DFDs at large, as defined by DeMarco [1979]. But compared to connections in DFDs, where DFDs refer to connections in terms of data flow, Kristeva's idea of connections is more

abstract.

Theme 2: *relevance is directed by knowledge*. Foucault [2002] directs relevance within a system of signs through knowledge. Therefore, contextualisation must be guided by a knowledge framework. This relates to how to direct connection in a process of making connections according to Kristeva [1990], which is also indicated by models following Cappelen [2007]: semantic, pragmatic, and indexical models; also AI context model by Guha and McCarthy [2003].

Theme 3: *context regresses endlessly*. Context requires a guiding process that enables a natural regression from one context element to another. But this regression must be limited at some point by a *rational* decision that postulates the absence of relevance based on knowledge. According to Culler [2009], extending context beyond traditional boundaries provides opportunities for redefining the context of the problem, hence producing new insights. Redefining boundaries is also referred to by Dilley [1999] as contextual moves. Therefore, context-diagram represent mainly the act of limiting context, as an analyst sets the boundary to limit context from extending endlessly.

Theme 4: *context has states*. Kazman et al. [2005] mentioned three elements for the context of software architectures: goals, architecture state, and constraint. Thus, different architectures may have different context states based on a certain combination of these three elements. Accordingly, it is possible to generalise the notion of context states to levels of software and system development other than architecture.

Kristeva [1990] also imply the concept of states for context through connections and disconnections. If context is the act of making connections, disconnections is an act of identifying elements that not part of the context. As a result, the system may be formed by a set of connections and disconnection, whereby an element could be either in a state of connection with another element, or a state of disconnection.

Theme 5: *context has influence*. Alexander [1964] identifies the role of context as the main source of influence on elements of design through what he identifies as force. This is also observed by John et al. [2009] on the influence of the architecture on how structure patterns are to be integrated within a larger context. The concept of context and force have been adapted by software patterns as the main source of influence on design decisions, as in the work by Buschmann et al. [2007] for example. This supports the observation made on the satellite system thought

experiment, on the existance of influencing factors that shape requirements and architecture. But what these ideas do not mention is that context has different degrees of influence, and how these influences change over time.

2.6.2 The emergence of *influence* and *perception*

The review of literature confirms what was introduced earlier in my preliminary research (Section 2.2), that context is the main source of influence, which determines the degree of variation for a system. But another concept emerged from literature. The role of knowledge in context. In the preliminary research, system requirements and architectures are recognised to vary based on the degree of influence within context. But influence within context is not always self-evident. Therefore, the role of knowledge becomes crucial for analysts to identify whether influences are actually real. As pointed by Culler [2009], Foucault [2002], and Harvey [1999]; elements of context come into being after they have been hidden, and unperceived, to redirect the understanding of a given context. It is also indicated through the observation made by Ozkaya et al. [2008] on explicating terminology use within the same practicing domain, in analyst's discussion with a stakeholder on the term 'flexibility'.

Therefore, from the role of knowledge in context I propose *perception* as another dimension of context in addition to influence. Thus context manifests through two aspects: the degree of influence, and the degree of perception. Perception of context plays an essential role within all the themes summarised in Table 2.3. For example, preserving structure of the two shapes in Figure 2.2 is not only determined by the level of influence of the frame—whether changing the frame is possible or not—but also on the degree of perception about the frame—whether the frame is actually shaped in this way, and whether its influence is actually what it is thought to be.

By presenting a synthesis of *context* based on two dimensions: *influence* and *perception*, I arrive at the main purpose of the thesis. Based on this novel view of context, the following conjecture is offered, which supports the notion that there is a need to model context in software systems as a separate system element.

> Identifying context through influence and perception exposes—more effectively—system limitations and opportunities to vary system elements, and support system developers to make more effective decisions on the kind of solutions there are in systems, in relation to software and hardware.

Conjecture

2.7 Summary and conclusion

In this chapter, from the preliminary research the role of context is recognised in allowing systems to vary relative to different degrees of influence. The less influence of a context the more one is able to vary its requirements and architecture. This is followed by answering the question 'what is context?', by reviewing different disciplines that approached the question: from using the term synonymously with environment, frame of reference, and setting; to the more formal definition offered by Scharfstein [1989] and Alexander [1964]. This led to the recognition that context lies in the interaction between two or more elements, not in the elements themselves.

Further review of context in software engineering revealed different uses of the term. Two main approaches are identified: first, context is used as part of setting system boundaries, exemplified in the context-diagrams as part of structure modelling such as DFDs. The second, is the common sense approach to context manifested in the use of requirement scenarios. The second approach is indicated by Scharfstein [1989], that context is more realised in practice than in theory.

By reviewing literature in other disciplines, where context is discussed as a separate issue, it is possible to recognise context as a problem, a solution, and as form. Context as a problem is actually manifested in three problems: the problem of regression, the shell problem, and the problem of relevance. This is mainly handled in reality, according to Scharfstein [1989], through common sense. As part of the solution to context, Wittgenstein [1974] observes that people engage in 'language games' where agreements about the meaning or the context of a dialogue is reached informally within a conversation. This is also observed by Ozkaya et al. [2008] in software engineering. Other more formal approaches to context are identified, either modelling context itself through certain parameters as in Goodwin and Duranti [1992] in anthropology, Cappelen [2007] in the study of semantics by using semantic models, or contextual frameworks in AI by Guha and McCarthy [2003]. In context as form, form is the result of its context, as pointed out by Alexander [1964]. Alexander [2002] builds on the concept of context as form through the concept of *structure preserving*, where 'form' is evaluated relative to its ability to preserve the shape of its context. While the structure preserving concept is recognised within a physical context—shape of the land, building forms, and so on—it also pertains to non-physical contexts—laws, and people's desires and ambitions.

In conclusion, from ideas reviewed in the literature it is possible to arrive at a synthesis of context based on two aspects: *influence* and *perception*. The degree of influence is realised within a degree of perception. Perception is indicated by

the concept of common sense by Scharfstein [1989], observations made by Harvey [1999] on implicit context, and use of knowledge to direct contextual relevance by Foucault [2002] and Culler [2009].

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Part

II

Contribution

Chapter

3

Context Models

> For real estate, the motto is "Location, location, location." For software process, it should be "context, context, context."
>
> [Kruchten 2009]

Contents

3.1 Introduction

In the previous chapter I have drawn some general themes from the literature review that support the idea that context is the source of influence on software systems and that context is also realised through perception as a second dimension of context. Here, I build on three of the previous themes identified: context has influence, relevance is directed by knowledge, and context has states. For the first theme—context has influence—the force model is presented. It introduces four levels of force, each level has its own implications on how the system may vary. The force model represents the influence dimension of context. For the second theme, that relevance is directed by knowledge, the knowledge model is introduced. It comprises five knowledge sources as a basis for selecting a relevant influence. The knowledge model represents the perception dimension of context. For the third theme, that context has states, the Context Dynamics Matrix (CDM) is introduced. It describes context in terms of sixteen states derived from joining the force model and the knowledge model. CDM joins both the influence and perception dimension of context in a single view.

3.2 Influence and perception

Influence and perception, as previously indicated, are the synthesis of context based on two views: the definition by Alexander [1964] that context is 'force,' and the definition by Scharfstein [1989] that context is what enhances our understanding. Alexander emphasises the influence of *context* on form, and Scharfstein emphasises the perception of *context* on understanding. Both views of context are intertwined, whereby it is not possible to recognise a force without first perceiving it.

Whenever introduced to a design problem, the first step is to understand the nature of the problem in order to arrive at a solution. As engineers, it is expected to introduce useful machines to the world [Jackson 1995a]. It is based on this usefulness, and goodness, that any work of engineering is judged. This is what Alexander [1964] refers to as the goodness of fit. But perceiving context is not straightforward. According to Boehm and Basili [2001], fifty percent of rework effort is the result of not identifying (perceiving) the correct requirements at the start of system analysis.

The context of the developed system, however, is not limited to the problem of establishing a working artefact. It extends to the concern of sustainability, by addressing needs within a future context that cannot be fully described. In

addition to the functional needs that the system is required to have, the system has to address future needs/context(s). Such needs are manifested in non-functional requirements, such as modifiability, security, robustness, and so on. As a result, the field of context becomes broader, thereby making it more difficult to describe. Alexander [1964] summarises this issue by stating that we are trying to design for a context that we do not fully understand.

But even at the local (present) context, the goodness of fit that Alexander [1964] refers to, is bound by its perception. How to distinguish a misfit from a fit without context? Scharfstein [1989] argues that given a certain set of contextual elements, it is possible to justify nearly anything. That is, given a combination of influences, what could be considered a misfit in one context may be a fit in another. Consider the simple example of a process that checks for user's authentication through a username and a password. Let's assume that a user enters a username and password, and the system refuses to grant him or her access. Is it possible then to determine whether a misfit or fit has occurred? How to determine if the process has failed to perform its function? This context may lead to a dilemma if it is not possible to access the stored password to be compared with the actual entered password to determine if what occurred is a fit or a misfit. If the person has entered the wrong username and password the process would be actually performing its function correctly, and the response by the process would have been a fit. But if the opposite is the case, that is, if the correct username and password has actually been entered, then the process would have resulted in a misfit. To use Scharfstein [1989] argument, knowing if the process and its context has resulted in a misfit or fit depends chiefly on how much we know about the context, or alternatively how the context is described. If the context is described to indicate that the correct username and password was actually entered, then the process has resulted in a misfit, if the opposite is described that the user did not enter the correct password, then the context becomes a fit. This is equivalent of asking the user to make sure that the correct username and password have been entered. Typically, whether the correct or the wrong password have been entered, the issue may be resolved by resetting the password with a new one.

But it is possible to argue against this example, by saying that this only applies in the context of user support, where circumstances surrounding the actual case may be in a way that it is difficult to determine whether the case is a fit or a misfit without further investigation. That user support is different, say, from requirements for example, where a fit and misfit can be well defined for every authentication process. If a requirement states that a user must enter a username and a password, the misfit and fit is clearly defined. To argue against this is to say that this is in fact true. User support is different from requirements, and it follows

that fits/misfits may be different as a result. But this difference actually supports the argument that misfits are bound by context. Because to move from the level of user support to requirements changes the context. The knowledge of fit/misfit on the level of requirements may be different from user support because the context is different. It is possible to say that the context of the process of user authentication on the level of requirements is typically well defined and rarely changes for any context. This last statement itself is produced based on a perceived notion that user authentication is a process that is universal, which applies to all possible cases. But it is possible to imagine an authentication process that only allows unauthorised users to access the system. In this 'context,' users with a username and password are checked and not granted access because they are blacklisted. In this odd case, misfits and fits are reversed. What is typically a fit becomes a misfit, and what is a misfit becomes a fit. Thus in order to change fit into misfits, what is needed is to change the context. The process of authentication has hardly changed. The question of why the process should deny authorised users to gain access, needs another context. And the newly provided context might need another context, and as Scharfstein [1989] points out, this becomes open for an endless regression.

Therefore, misfits are only determined within the context they appear in. But misfits are not all the same. For example, in the authentication process, a misfit of not granting a user access is different than the misfit of a user being granted access but after a long delay. Although both are considered misfits, not granting access at all is more severe than being delayed. In fact, a user that experiences delay might not notice the delay if the whole system runs at the same speed, because delay is relative in this context. But not being granted access is a misfit in almost all the cases of the process of user authentication.

Accordingly, it is possible to observe, from the example, that misfits differ based on context. Some misfits are more tolerable than others. But misfits and fits also depend on the perception of them. Consequently, both influence and perception, must be graduated in order for the context to be identified correctly. Not all influences result in the same misfits, and not all influences are perceived with equal levels of credibility. In analysing a context, as in the case of the authentication process, strong influences must be distinguished from weak ones, as the influence denies access compared to the influence that delays access. It is also equally important to distinguish based on the perception of the influence, have the user been denied access because the user did not enter the correct password, or the user have been acutely granted access, but the user was not able to realise (perceive) that he or she have actually accessed the system.

Therefore, in order to identify context more accurately, two models are proposed: a force model of influence, and a knowledge model of perception. Using

both models it is possible to identify different levels of influence and perception associated with each context element.

3.2.1 A model of influence: the force model

The force model of influence is partially derived from the definition of context by Alexander [1964], what places a demand on an ensemble, recognised as force. But by observing carefully how Alexander identifies such demands, it is possible to distinguish forces based on types of needs. This is particularly clear when reviewing the Indian village study.

In the Indian village study [Alexander 1963]—reproduced in Alexander [1964]—Alexander presents the following classification of needs:

- needs felt explicitly by villagers,
- needs called for by national and regional economy and social purposes,
- needs satisfied implicitly in the present village.

These categorised needs are different in nature. A need felt by a villager, is different from a need enforced by law. The first, is a need pertaining only to one villager, and may be shared by others, but the law applies to every villager.

After identifying the categorised needs, the study produces a list of needs in the form of statements. Each statement, if not satisfied, represents one source of misfit, statements such as '*rules about house not facing south*,' and '*cattle access to water*'. A review of the listed needs, similar to the needs category, also leads to conclude that there are notable differences between them. The need of the cattle to access water is different from the rules about house not facing south, in which the second may result in the loss of cattle, while the first leads to villagers dissatisfaction.

Therefore, differences among category needs, and differences among the list of needs themselves, have particular significance in resolving forces. To resolve the forces within the context of a village, design must take into account the differences between these needs. Consider the authentication process example. A user that seeks access may face two misfits: the process does not grant the user access, or grants the user access but after a long delay. According to Alexander's model both misfits are the same. But it is left to the analyst's common sense to recognise the difference between the two misfits. If the process does not grant access the system becomes dysfunctional for the user, but if the delay occurs it is still functional but not to the liking of the user. Both misfits influence the user differently. Therefore,

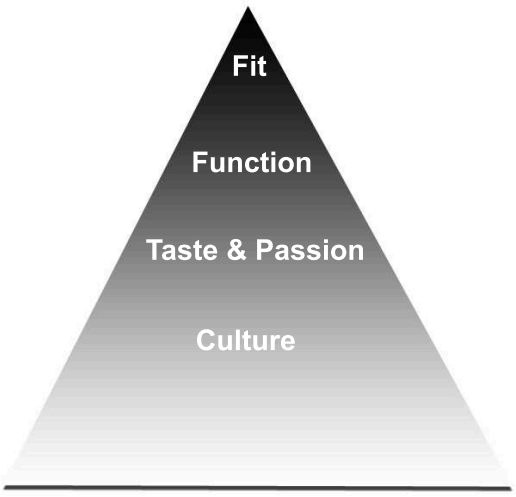


Figure 3.1 – Levels of forces, at the top is the force of fit which has more strict consequences, and as the pyramid gets wider the consequence of the force becomes less definitive and choices become more tolerant to change.

what is missing from Alexander's theory, is a model for the context of forces that exposes such differences.

To properly assess misfits and fits, it is necessary to produce a general model that describes the differences among forces. Alexander [1964] has put forward a model that recognises the existence of forces that influence form, and proposes that in order to achieve a fit between form and context, each force must be resolved—by transforming misfits into fits. But the influence model presented here goes a step further, by recognising that fits and misfits have different degrees of significance, and so do the forces that cause them.

By introducing the concept of influence that includes different levels of forces using the force model, first introduced in Alshaikh and Boughton [2009], the approach distinguishes between four forces. Figure 3.1 illustrates different levels of force. In what follows, a description of each force is presented.

Force of fit[1]

Fit requires elements to obey strict measures that cannot be compromised. Such strictness is typically expressed by parameters with well defined values, type of

[1]The term 'fit' here, is not what Alexander [1964] have termed as fit and misfit. As mentioned before, the concept of fit by Alexander is more general, while fit here applies only to particular contextual instances.

attributes such as: capacity, size, speed, distance, etc. In the authentication process, a strict force is placed on the user to match the username and the password in order for the user to be granted access. Even if the user enters the password with one less character, the process—or the model that executes the process—will still deny the user access. In this process, and because it forces the user to be precise about what the system requires from the user in order to enter, it also does not allow an alternative less restrictive point of entry.

What this means for users is that they will not be allowed, according to this influence, to vary how they are granted access. They have to pass through the same level of influence, a level of fit. Even if the system provides different ways to access the system, the force on the user will remain fit, whether the user needs to enter username and password, or provide a secret answer to a question, use a thumbprint authentication, or any other equivalent mechanism. A misfit as a result of this force would mean that the user will not be allowed to use some or all of the system's functionality.

Force of function

If the influence is a force of function, it is then expected that a specific goal is to be achieved, without necessarily specifying how the goal is achieved. Unlike the force of fit, where the goal is achieved through precision, a force of function is not usually associated with any precise measure. For the authentication process, the way the process grants access to authorised users after a correct username and password is entered, is a matter of function. The system is under a force of function to allow the user access in any way possible. The process under this force may vary how fast it performs its function, but it must allow the user to access the system in all its variations. A misfit as a result of a force of function would mean that the user did not have access to functionality, although there is in principle, multiple possible ways for the system to achieve its functionality.

Force of taste-and-passion

When under the influence of a force that is taste-and-passion, choices are limited according to preference not necessity. A force of taste-and-passion demands satisfying a preference. Unlike fit and function, not meeting the demands of a force of taste-and-passion should not result in a functional failure, but failure to meet users' demands, that may lead to customer dissatisfaction. If the authentication process grants access to the user, but the process takes too long, it is possible to say that the system is not able to provide a service in a timely manner according

to what the user regards as fast or slow. The degree by which the system is able to vary its service speed is sensitive to the user's preference. But knowing that the system is under a force of taste-and-passion enables the system to vary its service if needed to continue to provide functionality. A misfit under the influence of taste-and-passion would not necessarily mean that the functionality is not granted, but it is possible that the functionality is not granted according to the preference of an individual.

Force of culture

Culture forces a choice of tradition and established norms that may be against, at times, the personal preference of individuals. In this case, such a choice is made as a result of a cultural force. For example, if a system grants access to users after checking their username and password in a half a second, and takes a full second to grant access using the same process but somewhere else, does the system incurred a misfit for both cases, only the last one, or none of them? If the force is culture on both processes, the force must be weighed relative to the speed that normally takes to gain access for each place. That is to say, depending on the culture of the place where the system is operating, it is possible to regard that a full second is the best speed compared to other speeds that take two seconds. In other cases, such as the speed of half a second, it may be regarded as uncommonly slow compared to other functions that perform in half of its speed twice its function.

Accordingly, under a force of culture, it is not possible to vary how to achieve a goal unless accepted by the culture. The evaluation of whether a misfit is considered so, is also relative to where the misfit has occurred. But in all cases, violating the measures of what is accepted for a certain culture or not, does not mean that the function has failed. Consequences of not obeying the standards of a culture might result in serious misfit, but nevertheless, a misfit that is not universal.

3.2.2 A model of perception: the knowledge model

Perception in context stems from a number of factors influencing observation. Much criticism in the philosophy of science is drawn against induction as a method of inference, because of the limitations of observation—see for example Feyerabend [1988]. Inductionalists are challenged/weakened on the grounds of failing to account for the right initial conditions (context) of an experiment when relying on observation [Popper 2006].

Perceiving context is realised on different levels, pertaining to theories about

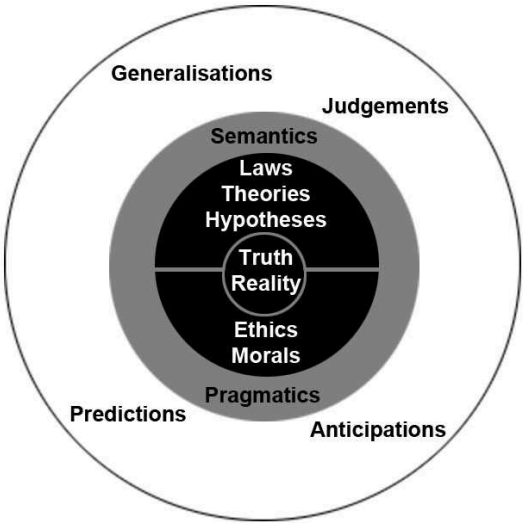


Figure 3.2 – The four parameters of perception represented in three rings, with a split core.

phenomena or concepts about meaning, analysis of events, interpretation of actions, state of affairs, and so on. In semantics, for example, the use of Indexicals Kaplan [1989] pronouns like: 'he' and 'I,' and references to time: 'now' and 'then,' are all references that create implicit contexts. In the utterance: "I am here", the 'I' is an implicit reference to a person not mentioned in the utterance by name, also 'here' in regards to place. Both cases have an implicit connection to someone or somewhere. It is not until 'I' and 'here' are clarified, that the reference becomes explicit.

Raising doubt about the validity of a statement that has generalities, stereotypes, and definitive judgements is another example where context is implicit. Berry and Kamsties [2004] refer to them as language errors. A statement like: "All lights have switches," not only raises the question whether 'all' is accurate (100 percent not merely 90 percent), but also whether each single light has its own on-off switch or all lights have single or perhaps double switches. Berry and Kamsties [2005] reported that when this statement was presented to clients for clarification, clients had different interpretations.

Therefore, similar to modelling influence, a model for perception is presented. The knowledge model is partially based on Goodwin and Duranti [1992] contextual categories that divide context into three parameters/themes:

- Setting: the setting in which the event under investigation occurs.
- Behavioural environment: non-verbal signs associated with an event. It includes spatial orientation and posture when analysing a talk, and the

actions of others involved in an event.

- Language as context: the language itself as data invokes context and provides context for other events.
- Background knowledge: how a local event is influenced by participants drawing from their own experiences stemming from culture and beliefs.

According to Goodwin and Duranti [1992], the study of an event is to determine how it invokes and provides context in a discourse. As Figure 3.2 illustrates, the model identifies four layers and five sources of knowledge. Drawing from Goodwin and Duranti [1992], it is possible to recognise truth-reality as the highest level of knowledge derived from the real world setting, theories and values (laws, ethics and morals) as part of the background knowledge, semantics and pragmatics (language as context, behaviour environment), and judgements (experiences identified as part of background knowledge).

Knowledge in the model is recognised in the form of layers. There are expectations of how events occur, drawing basically from judgement and experience, without having solid supporting evidence. Then a deeper level of knowledge is obtained from text, or a conversation, without having a deeper understanding of the theoretical framework from which the text is drawn. A deeper form of knowledge, is obtained from interpreting text while having a strong grasp of the theoretical framework or the ethical and moral arguments that support a decision or a choice. At the core is the basic realities of a situation where a particular event occurs. Identifying and verifying elements of text in reality—as a requirement statement—strengthens the knowledge about the text and provides firm grounds to base decisions on. In what follows, a description of each level is provided starting from truth-reality, the strongest form of knowledge, and ending by judgement.

Truth-reality

Truth-reality points to events and actions reported with certainty, as it pertains to the highest level of confidence. Reality points to tangible elements that can be verified and examined, such as implemented systems that have been used and understood for a long time. In order for a force to be based on truth-reality, the force or its effect(s) must be demonstrated. This is equivalent for the authentication process when the entered password is compared with the stored password to determine if the misfit is the result of a wrong password or a fault in the system. When the misfit is verified, whatever conclusion reached would be based on truth-reality.

Therefore, when a force based on truth-reality is identified, it states that the knowledge about the force is highly reliable, and is easy to verify, yielding consistent results.

Theory

Theories about the world are truths accepted with a certain degree of rigour within a vocational community. Such theories are well documented and have established a level of credibility. Some organisational goals are local theories of practice that are well understood and accepted by a particular group. Theory also constitutes a well formulated set of statements describing the ambitions and aims of a group of people. For an authentication process, it is possible to provide a model of the process that described how the system performs its function. Any influence identified using this model would be based on theory. The same model may include the code that performs this process, or the results of past successful operations from the system log. An analyst may use this information to formulate a better understanding of how the authentication process functions, and may be able to identify if a misfit is possible given the available information.

When a force in a system based on theory is identified, it is expected to have a high degree of reliability and may be applied to wide range of situations. A theory, typically, would have a considerable level of description, and familiarity within a certain group. Even if the theory, the model, or the principles, is not widely known, it is possible to provide resources explaining its underpinnings, and show its implications.

Values

Values form the basic beliefs and code of conduct within a society, or within individuals forming a group in a society. They are at times expressed and enforced by laws and regulations. Such rules and regulations are documented and well recognised. For example, a user may be granted access to the system based on law. coffee table design is required to fit quality standards enforced by law. This may also include how long it takes the user to gain access to the system. A piece of legislation may not allow the system to deny some users access to part of the system for longer than a certain period. A force of fit for example, in this case to fit the time period, is based on values—based on the values that a community hold for obeying laws.

When knowledge is based on values, a high level of conformity is achieved, conforming to one aspect of a society's context, its laws and regulations. When

values are compared to theory, theory varies between application domains, but values may vary between organisations and governments.

Semantics

A statement that is not part of a theory or values, and to open for interpretation, is identified as a semantic statement. Readers have different semantic interpretations of the same statements, thereby reflecting lower conformity. For example, words describing the process of authentication within a requirement statement might not be precise. Thus leaving room for interpretation, it may be misinterpreted, or regarded as lacking validity.

The knowledge model follows the definition put forward by Carnap [1942] of the difference between semantics and pragmatics, in which pragmatics is in reference to the user's language and intensions, while semantics is the study of statements as abstract expressions without considering the user. But semantics is considered here to include pragmatics whenever possible as part of the same category. Therefore, in the case of requirements, based on a semantic interpretation, statements are identified without having access to the producer. To interpret the statement's semantics using pragmatics means that the meaning is derived from the user directly, with or without a documented statement. But whenever possible, it is preferred to interpret statements taking into account stakeholders' intentions.

A force identified through semantics is more sensitive to the system's context than other forces linked to theory and values. There is always a chance that a statement is rephrased, or edited for corrections, resulting in a change in meaning.

Judgements

Judgement is a private interpretation or expectation not supported by any semantic statement. Judgements are useful whenever ambiguities arise, as they draw from experience, representing deductive reasoning and intuition. While they are easily challenged, they are also often overlooked, and mistaken for facts. For example, if the analyst makes the judgement that a misfit from the process of authentication is a result of a novice user who does not know how to use the system. It is possible that the analyst may have based this judgement on experience with the system, and new users, that leads to the conclusion that the user should try again.

Judgements are based on conjecture, and are easily misguided. Therefore, while it is important to have the freedom to explore implications drawn from the

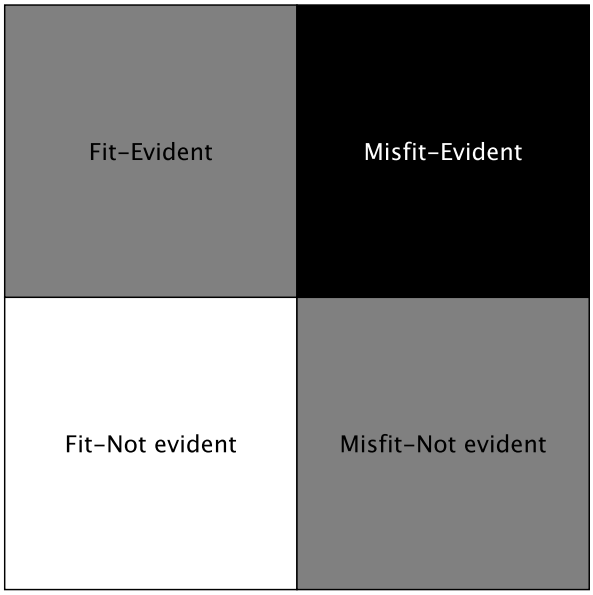


Figure 3.3 – Four state matrix of Alexander's fit/misfit model, showing two extra states when perception is added as evident or not evident.

understanding of a system, these implications must always be identified for what they are.

3.3 The Context Dynamics Matrix

To represent the two dimensions of influence and perception as context, context has to be described in terms of states. In which context is the result of both dimensions. Whenever a context element's influence is recognised, it is recognised under a certain level of perception. Each time the influence changes, the context moves to a new state, and similarly, if perception changes, the context moves to a new state. These states are represented by the Context Dynamics Matrix (CDM).

In Alexander's model, the perception of context is fixed by recognising misfits, arguing that misfits are self-evident. This results in two states, either the context and form become a state of fit or misfit. But as argued previously, misfits are not self-evident, as misfits themselves are recognised within a context of their own, states become four instead of two. Misfit has two states either a misfit is evident, or not, and fit has two states, either a fit is evident, or not. Each state may be represented in a four state context matrix (4-CDM).

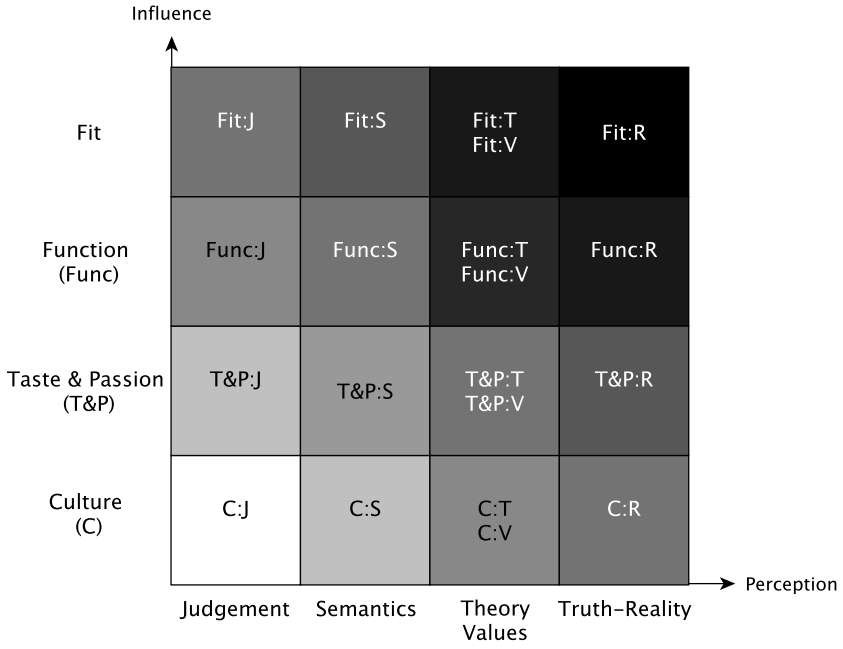


Figure 3.4 – The 4x4 sixteen state Context Dynamics Matrix (CDM). Showing the two dimensions influence and perception.

For example, in the Indian village study, the requirement '*cattle access to water*' indicates a fit when cattle have access to water, and represents a misfit when cattle do not have access. Therefore establishing a connection between cattle and water that has either a state of fit or misfit. But using the matrix, cattle not having access to water must establish evidence of misfit, and when access to water is accessible as fit, then fit must be also confirmed. Failing to establish fit or misfit results in the context state of an unconfirmed misfit and the context state of an unconfirmed fit.

To arrive at a more precise representation, a sixteen state matrix (16-CDM) based on influence and perception models is proposed. In the context of cattle and access to water, the influence of water on cattle must first be identified, function for example, and the level of perception is also identified, say, semantics. Then the context of cattle's access to water becomes function based on semantics. Therefore, fit and misfit is assessed within each context state. A misfit as a result of a function influence is different from a misfit as a result of a culture influence.

Combining the influence and perception models in Figure 3.1 and Figure 3.2, respectively, results in the sixteen state matrix (16-CDM). Figure 3.4 shows specific context states defined along the influence and perception dimensions. The matrix shows the most strict and most confirmed state at the right upper corner, fit based

on truth-reality (Fit:R), and ending by the least strict and least confirmed at the lower left corner, culture based on judgement (C:J).

To demonstrate the sixteen state matrix, the requirement 'rules about house not facing south' of the Indian village is used. According to the rule, and using Alexander's model, if the house faces south it becomes a misfit until changed to another direction. But following such an approach leaves some questions unanswered, for example: how accurate should the measure of south be? or how strict is the requirement? These questions may be raised and answered by modelling the context of the requirement. Using the matrix (16-CDM) the requirement's context has one of the following influences:

- Fit: south could be absolute. If the house is not oriented accurately it may lead to serious consequences.
- Function: avoiding facing south is only one way of achieving a goal, such as avoiding wind or sunlight.
- Taste-and-passion: avoiding facing south is based on villagers' preference.
- Culture: not facing south is a tradition held by villagers. While villagers may follow this tradition, some might choose not to.

Each of these influences could be perceived under the following conditions:

- The influence is based on judgement: with no supporting evidence based only on conjecture. For example, in the case of fit, by observing that all houses in the village avoid facing south.
- The influence is based on semantics: supported by a written text or by talking to villagers.
- The influence is based on theory: supported by credible evidence, the rule is generalised from a number of houses for the whole village. For example, in the case of fit, a theory is formed on the effects of houses facing south based on a number of incidents. The theory is generalised to be a rule for the whole village and other similar villages, thus supporting the need for accuracy in implementing this rule.
- The force is based on values: following the rule, or not following it, is enforced by the municipal authority. In the case of fit, even if effects of the misfit is not observed, the law still has to be obeyed.

- The force based on truth-reality: the rule is confirmed for individual cases. In the case of fit, for instance, the effects of not following the rule accurately are observed directly in the village and confirmed to be true for a particular house.

Having these states associated with the requirement, differentiates fits and misfits based on context. For example, knowing that a house facing south may result in real/true (severe) consequences, puts the requirement as a priority over other possible conflicting needs. Alternatively, knowing that facing south is particularly not preferred over other directions—that is, north for instance—but is possible to have some exceptions; thus leading designers/architects to avoid generalising this requirement for all houses. Accordingly, fits/misfits without these states are not concrete. Some misfits are relative to local and very specific cases, and other misfits are general to all cases within the Indian village. In Alexander [2002], similar emphases on local context is made, by appreciating and using, differences within the local context to generate living structures. Here, context is either identified as a local attribute as in taste-and-passion, or as a general attribute as in fit based on theory.

3.4 Summary and conclusion

The chapter started with a discussion of the concept of force presented by Alexander [1964] that results in misfits if context forces are not resolved. The advantage of misfits, as Alexander argues, is that misfits are self-evident. But as demonstrated by a number of examples, misfits are themselves realised within a context, and depending on the context, the result of a force is realised whether to be fit or misfit. Furthermore, fits and misfits based on the context where they appear, have different degrees of significance for a given design/system problem under analysis. Following these two observations about context, the context fits and misfits must be recognised within two models of influence and perception.

Influence represents the context of a force, while perception represents the knowledge of its influence. To model the influence of context, the force model is devised. The model divides influence into four forces: fit, function, taste-and-passion, and culture. To model perception, the knowledge model is devised. The model divides knowledge into five categories that occupy four levels: truth-reality, theory, values, semantics, and judgements. To capture the influence and perception of context in one view, the Context Dynamics Matrix (CDM) is devised. CDM represents context in sixteen states based on the force model and the knowledge model.

3.4 Summary and conclusion

What is presented so far is a theoretical framework to represent context on the level of individual system elements, where each context state only represents one instance of the relationship between two elements in a given time. It is the aim, however, of this approach to include a wider view of systems' context, by representing multiple elements. Accordingly, it is possible to use context states to analyse the context of multiple system elements.

Chapter

4

Context Mapping

Contents

4.1 Introduction

In the previous chapter, I have addressed three of the five themes previously presented in Chapter 2 on context, that context has influence through the force model, relevance directed by knowledge through the knowledge model, and context has states through the CDM. Here I build on what was previously presented, and address the remaining two themes on context, that context manifests in the act of making connections, and that context regresses endlessly. Both themes may be addressed by mapping context states to multiple elements within a system.

In software engineering, the most serious attempt to map context, by addressing the issue of how to limit its regression, and represent it as connections, is manifested in the use of DFDs. In building architecture, Alexander [2002] has extended the concept of form synthesis, which he first presented in 1964, to introduce a general process for architecture, which he refers to as the unfolding process. Both DFDs and the unfolding process, share some common characteristics on how to approach context in general. But because the concept of context in DFDs, as argued before, is approached as system scope and setting boundaries, it has limitations. Similarly, while Alexander has introduced the unfolding process as a general process, it has only been applied to building architecture. Thus what will be attempted here, is to extend the mapping approach by DFD, with context models of influence and perception previously introduced in Chapter 3, and apply the unfolding process to map the context of systems.

4.2 Mapping context

A review of DFD and the unfolding process is presented, because both approaches have emphasised heavily how the concept of context is used within software and architecture, respectively. But although both approaches address different concerns, DFD for software, and the unfolding process for building architecture, they share significant similarities.

Thus before the similarities between DFD and the unfolding process are discussed, each approach is introduced separately. The discussion of DFD, followed by a discussion of the unfolding process, will provide a brief review of both approaches. Finally, a discussion of how the approaches relate to each other is presented, thereby exposes points of similarity, which could be used to introduce a new way to represent software systems.

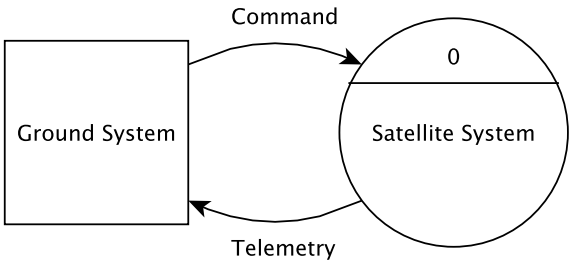


Figure 4.1 – A conceptual DFD context-diagram of a satellite system.

4.2.1 Context representation in DFD

DFD, typically, associates context with the context-diagram of the system that represents the scope of development, by making decision on what is included in the analysis and what is not. But as the analysis proceeds to represent internal system elements, at DFD level zero (DFD-0) for example, the term 'context' is not used. Yet, the concept of setting boundaries at DFD levels other than the context-diagram is still applied, where each process has a boundary that excludes other processes. Thus processes within the main process represented by the context-diagram, at a lower level of representation, are individual context-diagrams in their own way.

Overview of DFD

DFDs limit the description of systems to data flow and process on two levels. First, on the level of context-diagram, only one process is represented, where the diagram shows how data flow between the main process and external entities. Second, on all other levels, starting from DFD level zero, the main process is divided into further processes, where data flow between processes at the same level, or flow from higher levels and externally, such as the level of context-diagram. For the purpose of this overview, what follows uses the example of a satellite system represented in DFD, to discuss how systems are described on the level of context-diagram, then on the level of DFD level zero.

Context-diagram of the satellite system: when an analyst represents data-flow at the context-diagram level, the context becomes limited to setting the system boundary based on other contextual information inferred by the diagram. Information such as decisions, people, goals, function activities, sequence of events, relationships, synthesised in a long process of trial and error [Yourdon 1989]. As

Figure 4.1 shows, what results from this process is the interaction between the system and external elements, called terminators, through data communication.

But by setting the boundary of the system in this way, two assumptions are made. First, that the satellite system, represented as the main process, is the focus of the analysis. Thus the ground system—differentiated from the satellite system by the square shape—is an external element, thereby not considered for analysis, but rather represents a source/sink of data. The second assumption that may be made, is that the ground system is already developed, and as a result it is well understood. But if the ground system is in fact not built, the diagram would not change, because it does not have a way to communicate to the analyst if the ground system is already developed or not. This brings back the shell problem, where understanding the ground system is one way to understand the satellite system.

Another concern that the analyst considers, besides setting the boundary of the system, is to define how the system interacts with terminators through data. The diagram shows the satellite system as the central process that receives commands from the ground system, and sends back telemetries. But typically, the real communication between the two diagram elements is more complex. For example, the ground system could send different types of signals to the satellite, some of them could be summarised as commands, others might not fall into that category. For example, as the command to upload a new update of the satellite software, which might be represented separately. The same thing applies to telemetry. Therefore, when the context-diagram represents how the ground system and the satellite system communicate, it abstracts out a lot of detailed information.

Generally, the concept behind context-diagrams representing the context for any system, is summarised by the following:

- *Focus on the big picture, the context is not in the detail*. By extracting the most fundamental features of the system, the context-diagram focuses on the big picture, rather than listing every relevant detail. This solves the problem of information cluttering that might distract the analyst from understanding the system's mission.

- *Select a view of the system and describe its context*. Data and process is only one view of the system, thus its context is not the entire context of the system but of that view. For example, the diagram does not consider the order in which data is sent between the system and terminators. It also does not consider the influence of states on actions. Questions that the context-diagram of the satellite system may not answer include: does the ground system send the same commands in all of its states? or does each

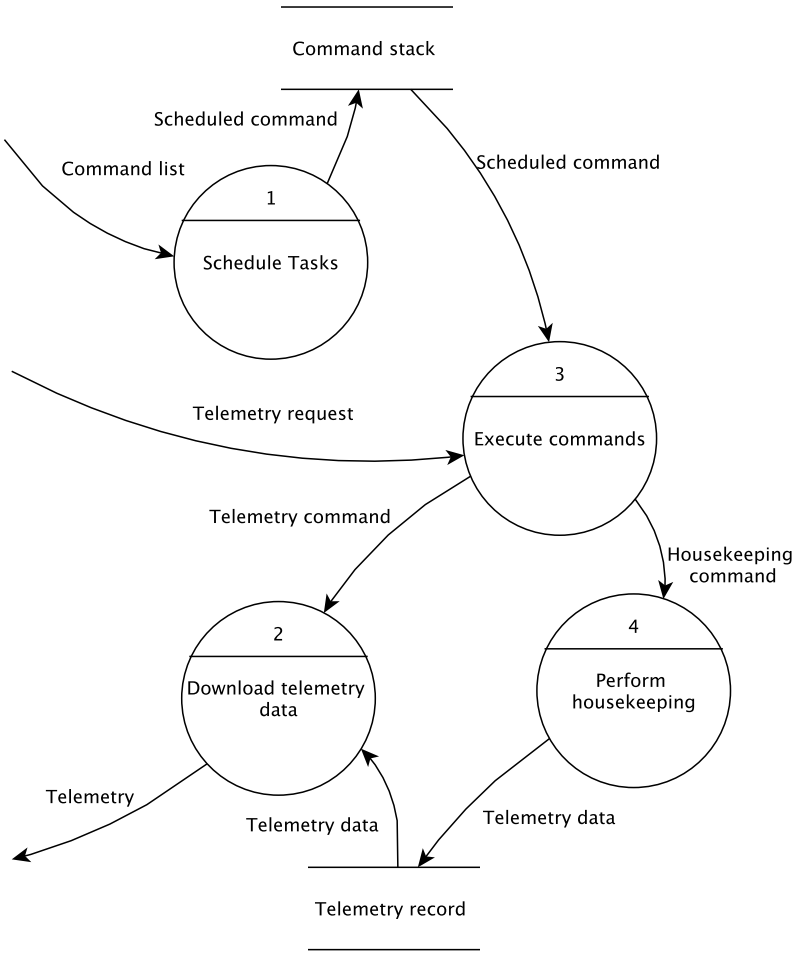


Figure 4.2 – Satellite system data flow level zero.

state have its own commands? Similarly, does the satellite system accept commands in all of its states or particular commands for particular states?

- *Focus on the system of interest and ignore any interactions between terminators*. The context of the system is only understood from the direct interaction between the analysed system and external elements or terminators. Other interactions are not relevant, because they are not directly related to the analysed system. Although such interactions are part of the general meaning of the word 'context,' and might have an influence on decisions within the system, they are excluded from being represented in the diagram.

DFD level zero of the satellite system: Figure 4.2 shows the data flow and processes at level zero. The diagram describes a series of major processes within the satellite system, mainly scheduling commands for future execution and

downloading telemetry requests. At this level, the diagram starts to represent, in more detail, what was abstracted out of the context-diagram. The single data flow represented by the context-diagram as 'command' is now divided to two data flows: command list and telemetry request. From the process 'schedule tasks,' it is possible to recognise that the ground system sends commands that the satellite executes according to a predefined sequence set by the ground system. This is different, for example, from 'execute commands' where the command is executed without a schedule. But all of this information is only interpreted from a general understanding of the context of the system that the mission provides, or building from background knowledge of similar systems, combined with what the diagram indicates through the names of each process.

But the diagram still follows the same approach to context. The context of each process and data flow is determined in relation to other neighbouring elements/processes. For example, the process 'execute commands,' in a way similar to the main process in the context diagram, sets a boundary for what is included in the process and what is not. The context of the process is understood, primarily, from data flow, such as 'telemetry request' as the context of 'execute commands,' and to a less degree, from other processes such as 'download telemetry data.' The diagram also excludes terminators, but indicates them by data flow that comes and goes to and from outside the figure, at the previous level of context. By comparing the context-diagram to DFD, at this level, DFD shows further detail. For example, the DFD shows data storage that was not shown in the context-diagram, and also breaks down command data flow into command list and telemetry request, that was shown as command at the context-diagram level.

An additional point to add, is the more important role that semantics play in DFD levels that proceed the context-diagram in how analysts understand the system. Because the context-diagram has only one process to describe, the name of the process and its data tells a less interesting story about what the system is all about. But at level zero, names of processes and data flow, allow the analyst to expand on the understanding of the context of the system through to semantical referrals. For example, 'schedule commands,' or 'download telemetry data,' expand on the understanding that the diagram gives, compared to what the context-diagram represents, which indicates that a 'command' is sent and 'telemetry' is received. Because processes start to have actions and more complex sentence structures, they open a new dimension to the context of the system that may add meaning and clarity.

Limitations of DFD

The main limitation of DFD, as presented previously, is that its concept of context is synonymous to the concept of setting boundaries. This is not only shown by the context-diagram, where the term context is used to denote the act of scoping the system, the same act of setting boundaries is followed by other processes at level zero, and within process decomposition and detailed expansions. Each process has internal elements, which are not shown at the same level, and other neighbouring processes that are external to it. Other limitations are discussed as follows:

- *Elements share the same level of perception*: perception, as context dimension, is not represented in DFD on all levels. Elements within the diagram may show the same perception level, if any level can be derived at all. Note that perception, here, should be understood within what was introduced in Chapter 3. Because by decomposing elements, to show further processes and data flow, from one DFD level to the next, enhance the perception of the system as a whole. But such enhancement when it occurs, is not represented by the model itself. For example, it is not possible for analysts, when they construct a context-diagram, to share their doubts or confidence in the state of affairs of how elements relate to each other within the diagram.

- *Has limited reference to external context*: except in limited semantic references, DFD elements have no reference to the system context other than the data and process that the diagram represents. Accordingly, the DFD represents a closed system. For example, if the DFD represents a process that verifies an entered password, it may not refer, in the diagram, that there is a possibility that the system may reject the entered password. In the case if the rejection is not represented, it may only be realised by the analyst through the semantic reference of 'password,' where the analyst would know what to be expected when a password is entered—either it is accepted or rejected. But without this semantic reference to the meaning of 'password,' and what operations are associated with the meaning of the word, that possibility of rejection may not be obvious.

- *Does not have any reference of where/how system elements may vary*: looking at a DFD diagram it is not possible to decide what or how to vary system elements, notwithstanding semantic references from data flow or process. But a well represented DFD that does not indicate implementation technologies, may allow system designers to vary the design of the system around the same DFD representation. But the DFD does not indicate if or where within the diagram that the system may vary.

4.2.2 Context using the unfolding process

Alexander [2002] introduces the unfolding process as a stepwise approach to design based on a close observation of changes in context. Unlike Alexander's previous work, the unfolding process distinguishes between the context of local elements and the context of the system as a whole. As mentioned earlier, the concept of structure preserving introduced by Alexander maintains a relationship between the local context and overall context of the process of building. The ultimate aim of the process is to support this relationship, in which the local context becomes harmonious with the global context. Alexander goes further to suggest that the role of the local context is to support the global context. The result of this process is to achieve building structures that have more life.

In addition to maintaining the relationship between the local and the global context, the process places a strong emphasis on observing changes in the local context each time a new element is added to it. The process, then, recognises changes to the context that results when elements are added or removed. According to Alexander, the process continues to be a series of adaptations and transformations, that evolves piecemeal with each new element added.

The unfolding process is based on two fundamental concepts: the concept of centres, and of piecemeal incrementation. But the two concepts are related, because as the process adds or removes centres continuously it: increments, transforms, adapts, and differentiates the living structure piecemeal. While this process is ongoing, it also preserves its structure as a whole.

The concept of centres

Alexander [2001] identifies 'centres' as building blocks of the unfolding process. A structure is enhanced, and made more whole, as the unfolding process performs certain moves that transform the structure. Alexander argues that these moves are countless within nature. The examples that Alexander gives to support his argument, are drawn from natural phenomena, observed in biology and physics, and within physical form that were created, mostly, by traditional societies. Much of what these examples appear to show, Alexander argues, is a complex system of centres and centres of centres that have a geometric order, a configuration of some kind, which makes structures appear to be more whole, more living.

But to answer 'what are centres?' does not seem to be a trivial exercise, because the answer includes the use of the concept of wholeness, yet another problematic concept. Yet, Alexander solves this enigma by stating that a centre is what results when the sense of the whole is combined with the parts within a certain

configuration. The result is a centre that acts as the whole, and centres that act as parts. For example, the satellite system is a centre formed by other centres, such as the data flow of 'telemetry request,' and the process 'schedule tasks'.

But the reason that Alexander uses the term 'centre' instead of 'whole,' has particular significance to the issue of the 'unboundness of context' identified as one of the problems of context in Chapter 2. Alexander recognises that the term 'whole' has a sense of markedness that cannot be accurately identified for each centre. This is because it is not possible to set an absolute boundary between elements of analysis. Thus the use of the term 'centre,' implies that the analysis should always recognise that the context regresses endlessly.

Centres also relate to what Goodwin and Duranti [1992] have identified as focal events, and focal points [Alshaikh and Boughton 2009][1]. But there is a subtle difference between a centre and a focal point. A centre exists in relation to other centres within the context independently from the scope of analysis or the attention of the analyst. But a focal point is the result of the awareness of a centre. Thus when a centre becomes fixed as the focus of

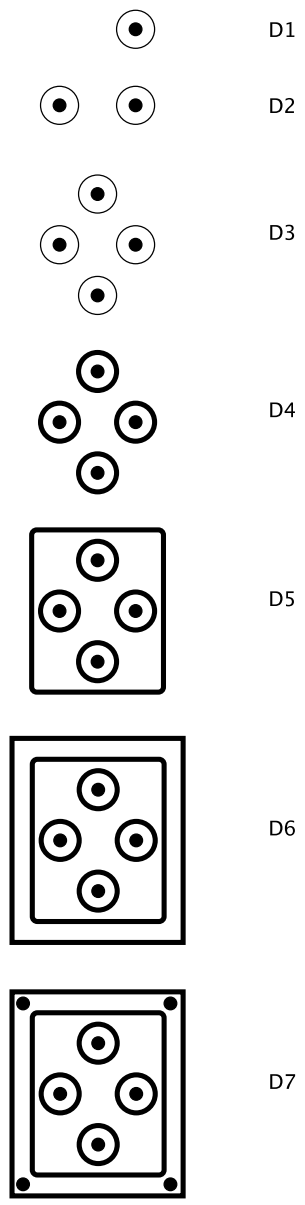


Figure 4.3 – The use of THICK BOUNDARY and LOCAL SYMMETRY to enhance the design of the ornament.

[1]When reviewing the part that Alexander [2001] first introduces the concept of centres, at the end of the section, Alexander remarks that he sees centres as focal points within "a larger unbroken whole". This shows that there is a kinship between what I identify as context and what Alexander [2001] recognises as a whole. But there is yet another difference between the use of the term 'whole' and 'context.' The 'whole,' according to Alexander [2001], what might be identified as the larger context, to be distinguished from the immediate context. Thus the role of mapping context is to be able to capture the larger context within a system by cumulating local contexts

analysis, it then may be recognised as a focal point.

Accordingly, when the focus is set on a centre, the analyst must evaluate the strength of the centre. Alexander presents the unfolding process as a series of transformation acts, where latent centres are transformed into strong centres. But the evaluation of the latency or strength of a centre is not evaluated through the centre itself. on the contrary, it is evaluated through other centres, or what Alexander identifies as the 'whole.' If multiple centres have the quality of life within a structure, the one which has less life would be a weaker centre. Life, as Alexander sees it in his view of building architecture, emerges as a result of a certain configuration within space. Thus Alexander [2001] defines fifteen geometric properties that enhance the life of structures. When a centre is evaluated as latent, an architecture may use one or multiple combinations of the fifteen properties to transform the centre to make it 'more living,' hence stronger.

For example, two of the fifteen properties that form structure transformations that bring life to structures are the the property of THICK BOUNDARY and LOCAL SYMMETRY. To demonstrate how to enhance a centre using these two properties, consider Figure 4.3, which is shown a series of transformations that starts from a simple ornament (D1). The first step to enhance D1 is to transform it by adding another ornament (D2) to form a LOCAL SYMMETRY. The ornament in D2 has more life than D1. Another transformation seeks to enhance D2 further by adding another symmetry. Thus D2 is enhanced further when it is transformed by reconfiguring the ornament by forming a vertical symmetry (D3). In D4, THICK BOUNDARY is used. Each circle is enhanced further by adding a THICK BOUNDARY. But to enhance the circles further as a whole, THICK BOUNDARY is used again in D5. In D6, THICK BOUNDARY is made even thicker, hence transforming the boundary set in D5. Adding another boundary to D5 created a new centre that emerged from the THICK BOUNDARY itself. The new centre is enhanced further by LOCAL SYMMETRY in D7. Notice that the transformation process continues to evaluate each step to identify latent centres, which motivated adding points within the THICK BOUNDARY using LOCAL SYMMETRY. While THICK BOUNDARY transformed D5 to produce the more living D6, it also produced further areas that need to be transformed again. This is typical of continuous adaptation and transformation that the unfolding process follows.

Piecemeal incrementation

The unfolding process performs piecemeal incrementation by continually evaluating the context with each element that an analyst adds. The process starts when the analyst evaluates the 'whole' to identify latent centres, then identifies an action

that transforms the centre into a strong centre. When the new element is added, a new configuration emerges, which is evaluated to be reconfigured again. This process of evaluating and transforming, happens in a gradual manner. The result is that the structure, or the configuration of the whole, runs through a piecemeal process that evolves the system of centres.

Figure 4.3 is a good example of how a structure evolves piecemeal. From D1–D7, each new increment is the result of how the previous step is configured, and how it is evaluated. In D3, for example, it is clear that the four circles have a thin outline. Because the thin outline is perceived to result in a latent centre, the chosen action to make it a stronger centre is to thicken the circle's outline, following the THICK BOUNDARY property. However, because the intention is to enhance the whole, not only one circle is transformed, but all of the circles. This is also followed by the frame added in D5. To add the frame around the circles, does not enhance one centre, but enhances all the circles.

But Figure 4.3 does not necessarily show all the steps that go into moving from D1 to D7. For example, the steps to move from D1 to D2 may be broken down further into sub-steps. The first move is to add a circle beside D1 to form a symmetry. But if the new circle is added without a dot, the whole then becomes a broken symmetry. Even by adding the circle there is more work to be done, the whole is not yet complete. The conclusion that the symmetry is broken comes because after each move, it seems natural to evaluate it, and ask how to enhance it further.

What is maintained for every series of steps, as Alexander [2002] emphasises, is to preserve the structure with each transformation. For example, in Figure 4.3, starting from D3, all of the moves maintain a sense of coherence of the whole that preserves what is there and enhances further. In D4, the boundary of all of the circles are made thicker, not only to one or two, but to all the circles. Similarly, D5 sets a boundary around all the circles, not just part of it. Thereby the process continues to preserve what is done in the previous steps and to enhance it.

4.2.3 The unfolding process and DFD

Although the unfolding process is applied to building architecture, Alexander [2002] argues that it is a process of life that applies to more than building structures. In fact, when the unfolding process is compared to DFD, it is possible to find some parallels. What is attempted here, is to interpret the unfolding process to explore how to apply it to the analysis of systems.

DFD diagrams represent processes that are similar to what Alexander identi-

fies as centres. Within the diagram, a process is part of a larger and higher level process, and in itself, is formed by lower level processes. Similarly, a centre is part of a larger centre and in itself is a centre formed by smaller centres. Thus, the unfolding process suggests that it starts by forming a main centre, then unfold new centres from the main centre. Alexander [2002] gives the example of the Japanese tea house, where the tea house is first placed in a secluded garden. The secluded garden is then divided into two gardens: an outer garden that has a dwelling, and an inner garden that has the tea house. The process then continues to describe what elements are within the outer garden and the inner garden, other than the dwelling and the tea house. Both gardens in this example are centres, which allow other centres within it to emerge, such as the dwelling and the tea house. This is essentially what the analysis of the satellite system using the DFD does. The satellite system, as the main process in the context-diagram, is the main centre for the system. Other processes emerge from this main process at diagram level zero, such as 'schedule tasks' and 'execute commands.'

Both examples, the Japanese tea house and the satellite system, show two significant parallels. First, both approaches allow the analysis to regress endlessly into infinite number of centres and processes. This is either by expanding the boundary of the main centre or process, to regress outward, or to follow what centres and processes emerge internally. Second, both approaches preserve the structure of the whole, as each centre or process preserves the centre or process to which it belongs. For example, in the context-diagram, the main process 'satellite system' has processes within it that preserve its structure. The way the structure of the process is preserved, manifests itself in processes that act coherently when they handle data flow that come in and out of their parent process, in this case 'satellite system.' If a process does not handle a data flow from the main process 'satellite system,' or sends a data flow that the context-diagram does not show, then that process does not preserve the structure of its parent process. Similarly, for the Japanese tea house, the dwelling within the outer garden, preserves the structure of the outer garden by being within its boundaries, not for example to be placed between the outer and inner garden. In both cases, the tea house and the satellite system, the role of preserving the structure is not performed by one centre or process, but by the work of multiple centres within a centre, or processes within a process.

But there is a significant difference between the unfolding process and the typical use of DFD. DFD are typically applied to data and process, while the unfolding process has no such limitation. It is true however, that the DFD is not limited to technology, processes do not have to be computer processes, and data may not be digital but 'material.' Processes may be actions performed by people,

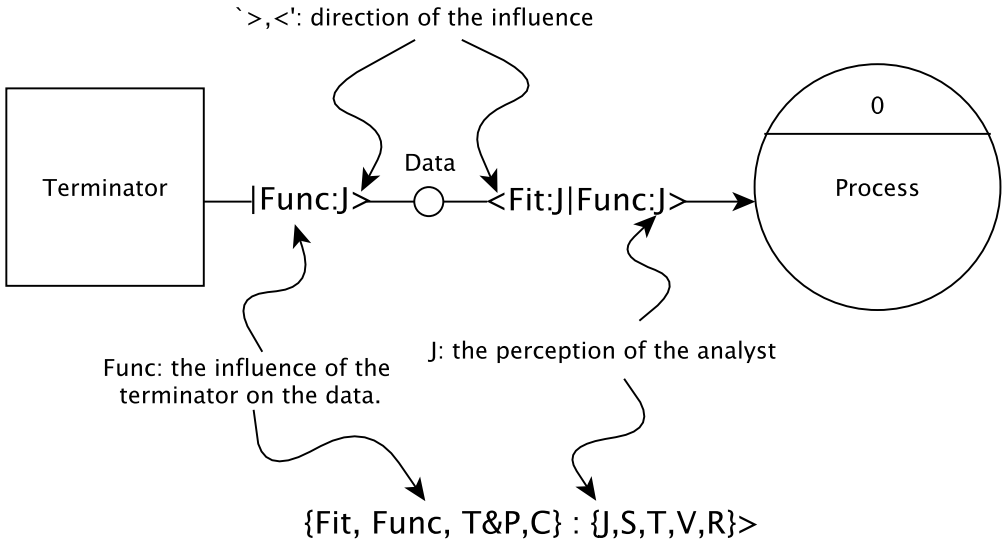


Figure 4.4 – Context states added to a DFD diagram.

and data may be tangible elements, such as office paper. But DFD only represents the functional aspect of the system, and does not represent other system concerns, such as quality, business goals, and constraints. The unfolding process, on the other hand, is detached from the theory of the fifteen properties. As a result, it is possible to apply the unfolding process without using the fifteen properties.

Another notable difference is how DFD limits the analysis from extending externally or internally. The DFD limits the analysis externally by setting the boundary of the system using context-diagrams, and internally by requirements, as the analyst decomposes the system functionally until all requirements are represented. But no such limit is posed for the unfolding process. In fact, the unfolding process does not dictate how to start the process: the design may unfold outwardly or inwardly, bottom-up or top-down. But DFD starts by realising the system from the top at the context-diagram and decomposes the system, in a top-down process. The unfolding process, however, as Alexander [2002] presents it through several examples, may be bottom-up, such as the ornament example, or top-down, such as the traditional Japanese tea house.

4.3 Mapping context to DFD

The concept behind representing DFD, as discussed previously, is to set context at context-diagram level of analysis to identify boundaries and set system scope. But the same concept is also used iteratively for other more detailed levels

of analysis. What resulted from this approach is that the analysis became limited to the functional data-process view of the system. Accordingly, DFD has limited what is relevant to its representation, temporarily perhaps, to data flow and process, without necessarily undermining the importance of other equally important system aspects that are relevant to the system.

For example, while DeMarco [1979] limited the description of requirement through data and process, Hatley and Pirbhai [1988] extended DFD to represent states and events for realtime systems. Originally, DeMarco [1979] presented DFD to describe the system as an idle machine. Issues, such as order of processes, are not identified by the analysis. But Hatley and Pirbhai [1988] describes in terms of states and events in realtime systems. Hatley and Pirbhai [1988], for example, describes the role of sensors and control, in addition to process and data flow, which DFD traditionally represent. Thus, it is possible to determine from the DFD which process is invoked as a consequence to an event.

Similarly, what follows shows how to use context states to enrich the representation of data flow and process when the diagram identifies the influence and perception of context in terms of context states. The approach to use context states is demonstrated by the example of the satellite system represented in DFD.

4.3.1 Enriching DFD by context states

DFD satisfies two themes identified, that context is an act of making connections, and context regresses endlessly. The first, the act of making connections, is manifested in data flow between processes and terminators, and the second, that context regresses endlessly, is manifested in how processes may extend internally and externally. Therefore, when context states are added to DFD, the enriched diagram that results satisfies the five themes of context identified from literature, for example: Foucault [2002], Scharfstein [1989] for perception and kwnoledge, Alexander [1964] for influence, Kristeva [1990] context as connections, Kazman et al. [2005] for context states, and Culler [2009] for context regression.

Figure 4.4 shows how context states are mapped to the original DFD notation. Context states are expressed in terms of the two dimensions: influence and perception. They allow the analyst to express the context of each element within the DFD more accurately. For example, in Figure 4.4, the terminator places a force of function (Fun) based (:) on judgement (J) on data, which flows from terminator to process. In this case, terminator requires from data to satisfy the function goal of sending the data. But because this influence is based on judgement, it might be false. Thus, the state implies that the function *might* be the influence that terminator applies on data. Because it is based on judgement, the analyst may

signal to who ever reads the DFD to verify if the influence is truly function, or accept that there is a chance that this influence may be wrongly identified.

What the use of these context states with DFD suggests, is how an analyst can enrich the diagram with other information other than what the system does. The analyst may indicate that 'data,' for example, may only flow to the process successfully if it flows at a particular speed, otherwise it would not be useful. Such influence is identified in the context matrix as fit. If the analyst chooses to use the original DFD notation, all context states would be expressed as function, even if the analyst's knowledge about the context of terminator and data tells him or her otherwise. Furthermore, the analyst may indicate how the influence was perceived. Is it a personal or professional judgement that has no support from requirements, or is it supported by requirement statements and a well known theory or model?

Each context state may raise questions as much as it provides answers. A context state that shows an influence of taste-and-passion may answer who is interested to have this requirement satisfied in a particular manner. A data to be sent at a particular speed may only be as a result of a stakeholder's preference, not a system constraint.

When analysts add context states to connections, they add other elements external to DFD to the diagram. On the influence dimension, if fit is assigned, it points to objects that enforce the influence of fit (Fit), such as an authentication model. If function (Func) is assigned, it points to a goal that must be achieved by any means. If taste-and-passion (T&P) is assigned, it points to a preference of an individual, either a stakeholder or a system developer. If culture (C) is assigned, it points to the general public, to history, to a large group of people within an organisation, or a community. Similarly, on the perception dimension, when judgement (J) is assigned, it points to the level of perception associated with a person or a group of people. If semantics (S) is assigned, it points to a particular statement, such as a requirement or a scenario. If a theory (T) is assigned, it would point to a series of statements, a model of requirements, a specification, or a system goal, which supports the influence and perhaps describes it in detail. If values (V) is assigned it would point typically to a law or a business rule, and if truth-reality (R) is assigned, it would point to realised objects in reality, current events, or self-evident truths.

Therefore, context states add various concerns that the DFD have temporarily deemed not to be relevant, as analysts wish to describe what the system functions are through data and process. Accordingly, analysts may choose to expand on the functional description of the system using context states, to either extend the whole DFD or part of it. For example, it is possible to extend the context

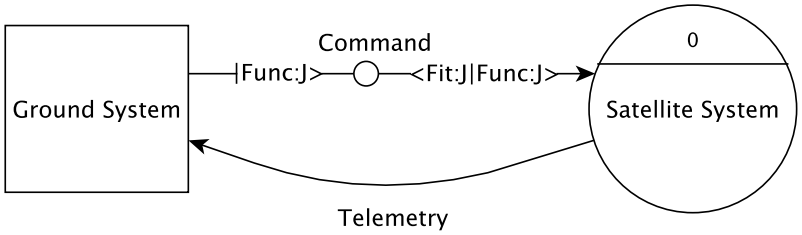


Figure 4.5 – Context states assigned to the satellite system context-diagram

information of the context-diagram alone, or elaborate on the context of parts of the DFD-0. Because what the DFD describes normally are functional aspects of the system as described by requirements, it is possible to classify each element of any DFD diagram to be in the context state of function based on semantics. Thus function based on semantics is the default context state of any DFD diagram. But because that is not always the case, other context states may be useful to use when some system element are influenced by forces other than function, or based on perceptions other than semantics.

4.3.2 The context of the context-diagram

Context-diagrams are typically used to determine the boundary of a system, but do not model context in general. They also limit how to describe the system, because context-diagrams only show a single process, data flows, and terminators. But if analysts assign context states to context-diagrams, without interfering with the process of setting the boundary of the system, it is possible to have a context-diagram that relates more to the system a significantly broader view of context than DFDs typically provide.

Consider the context-diagram of the satellite system example. Figure 4.1 shows the interaction between the satellite system as the main process and the ground system as an external entity. Using the context-diagram as a whole, it is possible to build a partial understanding of the context of the system through data and process. But the diagram only implies the context of 'command' or 'telemetry,' for example. If the context of 'command' is to be identified, its context is formed by its connection with 'ground system,' on one side, and 'satellite system' on the other. Thus the context of 'command' is identified within the context-diagram when it is assigned context states relative to 'satellite system' and 'ground system,' as Figure 4.5 shows.

In Figure 4.5, command has two contexts: its context relative to the satellite system, and its context relative to the ground system. The context of 'command'

relative to ground system, shows that 'command' is influenced by 'ground system,' while 'command' has no influence on 'ground system'. The influence of 'ground system' is function based on judgement. What this means, is that although there is a good reason to identify the influence of 'ground system' on 'command' to be function, but because this fact cannot be supported, or verified by requirements, or any theory or goal, it is recognised according to the judgement of the analyst. Thus the influence of function may be fit, or taste-and-passion, if the judgement is wrong. On the other hand, the context of 'command' relative to 'satellite system,' has two directions. First, the context of 'command' on 'satellite system,' which is function based on judgement. Second, the context of 'satellite system' on 'command,' which is fit based on judgement. The first state indicates that the 'satellite system' has to satisfy the goal of sending the command. For example, if the command requires from the satellite to download data to the ground system, it may download it in any way possible, as long as the data is delivered to the ground system. But because the satellite system applies a force of fit on 'command,' the satellite system will only achieve the goal of 'command,' if the command fits a certain condition. Otherwise, the goal of sending a command will not be achieved. The condition may be, for example, to obey a certain format, such as an encryption protocol. But because both influences are based on judgement, they may still vary.

Thus the three context states that the extended context-diagram show, have specific implications on system variation. Influence implies the degree that the system may vary, and perception implies how the influence is identified. For example, one possible scenario where the force of fit that the 'satellite system' applies on 'command,' relates to the structure of 'command.' If 'satellite system' only accepts one structure for every 'command' it receives, then for the 'command' to be processed or accepted, and ultimately for the goal of sending the command initially to be achieved, it must fit that structure. Accordingly, the context state indicates this context through the influence of fit. Fit, then, implies that 'satellite system' will not accept a command that has a varied structure, from the command structure it knows about. It also indicates that as long as the influence is fit, failing to meet the structure demands that 'satellite system' enforce, should lead to fail to achieve the goal set by 'ground station' for sending a command. But because the force of fit is based on judgement, what is required from the analyst is to verify whether the judgement that the force that 'satellite system' applies on 'command' is true.

Figure 4.6 shows the context states of the context-diagram mapped to the white cells of the CDM: 'command' on 'satellite system' (C<SS), 'satellite system' on 'command' (C>SS), and 'ground station' on 'command' (GS<C). The cells with light grey are the first possible state transitions that may occur for the elements

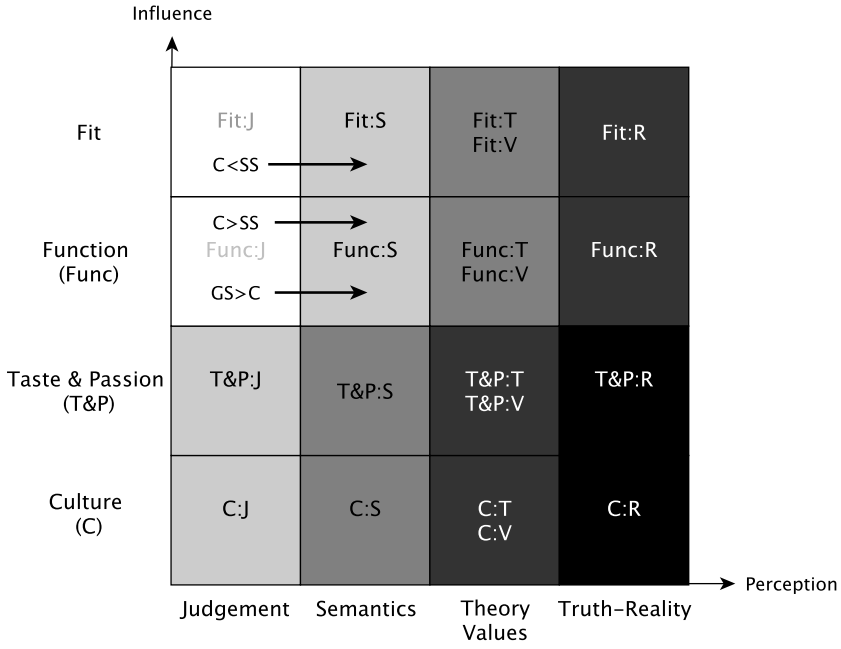


Figure 4.6 – Context states of context-diagram at Figure 4.7 mapped to CDM.

within the white cells. The arrows show the target state transitions. For example, the context state of 'satellite system' on 'command' (C<SS), should transit from fit based on judgement (Fit:J), to fit based on semantics (Fit:S). For context state, if the judgement is agreed upon, it may be documented. But another possible state transition that may result from reviewing the judgement of fit, is to recognise that the judgement was wrong. As a result, another judgement may be made, such as to judge that the influence is taste-and-passion, which leads to a context state transition. When the context state transits to taste-and-passion based on judgement, analysts should aim to transit the context state to taste-and-passion based on semantics.

What analysts should aim to, as they identify the context state of the system, is to achieve at least the default context state of DFD, that is function based on semantics. Other context states, such as taste-and-passion based on semantics, may serve design purposes more than analysis purposes. For example, it is possible to identify a command to be sent at a certain rate of speed. But what DFDs typically represent, is to express the functional aspect of the requirement, which is the function of sending the command. Thus the requirement becomes functional when the rate of transmission is not considered. But with context states, it is possible to indicate that in addition to the functional aims to send a command, there is an additional aim, derived by preference, to send the command at a certain speed. Indeed, other functional requirements may have preferences, but

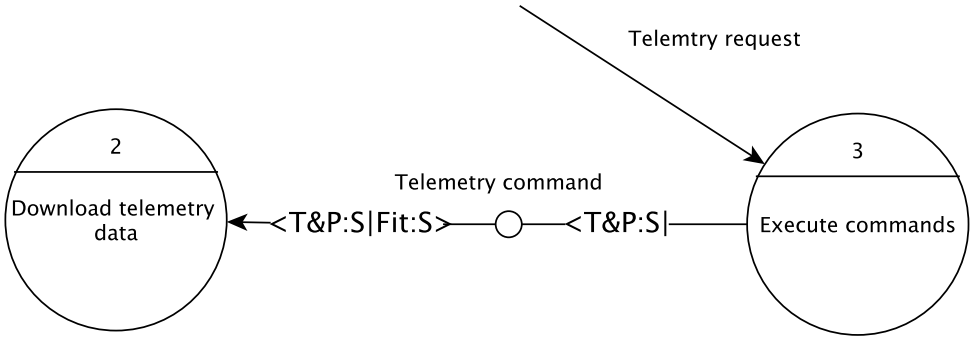


Figure 4.7 – Context state of telemetry command.

not necessarily expressed at the analysis level. What analysts typically do, when they analyse the system in DFD, is to regard these preferences to be irrelevant to analysis, even if they are available to the analyst at the time of building the DFD.

Therefore, in the context-diagram of Figure 4.5, only context states that have states other than function based on semantics are identified. Accordingly, the diagram shows the context states relative only to 'command,' because their states are not function based on semantics, compared to 'telemetry' in the same diagram, for example.

4.3.3 The context of DFD-0

Similar to context-diagrams, DFD-0 continues to describe the system using data and process, but show new elements, such as data storage, which context-diagrams do not show. Accordingly, when DFD-0 is enriched by context states, it is expected to show similar results to what is expected from context-diagrams when they are enriched by context states, themselves. But because DFD-0 are typically more complex, as they show more than one process, the need for context states should be greater.

When the context-diagram process unfolds, it shows the internal processes that the context-diagram hides, and produces DFD-0. At this new diagram level, and other levels that unfold from its processes, process and data store are represented in a similar fashion to the main process and external entities at the context-diagram level. But unlike data flow at the context-diagram level, data flow at DFD-0 and other subsequent levels, have two forms: data that flow between processes at the same level, and data that flow in or out of the diagram, to and from a higher level process. For example, the DFD-0 of the satellite system (see Figure 4.2), shows 'telemetry command' that flows between 'execute commands'

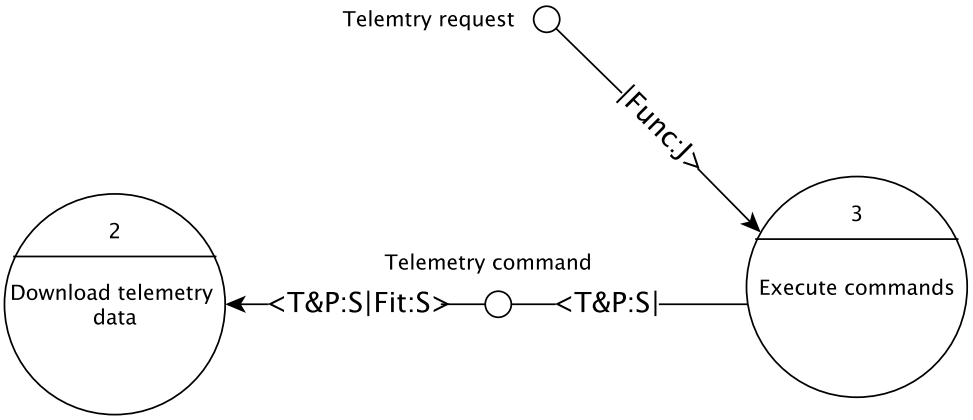


Figure 4.8 – Context state of telemetry request.

and 'download telemetry data,' and other data that flow either from outside the diagram, such as 'telemetry request' to the process 'execute command,' or flow towards outside the diagram, such as 'telemetry' from 'download telemetry data.' Although both types show a subtle difference, they show a difference on how their context states are assigned. Data that flow between processes on the same level have two context states, one at the sender's end and a second at the receiver's end. But in the case of data that flow from or to outside, the diagram only shows the context state at one end, that is the data flow and the process of the depicted level, and hides the process and its context state at the higher level. To illustrate the difference between the context of both data flow types, consider the example of 'telemetry command' and 'telemetry request'.

Figure 4.7 shows the context of 'telemetry command' that flow between 'download telemetry data' and 'execute commands'. The context states that the diagram shows, indicate that 'telemetry command' influences 'download telemetry data,' but not 'execute command'. But both processes, 'download telemetry data' and 'execute commands,' influence 'telemetry command'. Telemetry command's influence on 'download telemetry data' is taste-and-passion based on semantics (T&P:S), which indicates that the demand of the data flow in this case is only preferred. Accordingly, if the preference is about the speed of sending 'telemetry command,' analysts and designers should know that it is possible to vary that speed if needed, because it will not result in failing to process it. Similarly, 'execute commands' applies a taste-and-passion influence on 'telemetry command' based on semantics (T&P:S). If the preference is about the structure of 'execute commands' to achieve better processing speed, for example, analysts may have a better understanding of the consequences when the command structure is changed. 'Download telemetry data,' however, places an influence of fit on 'telemetry

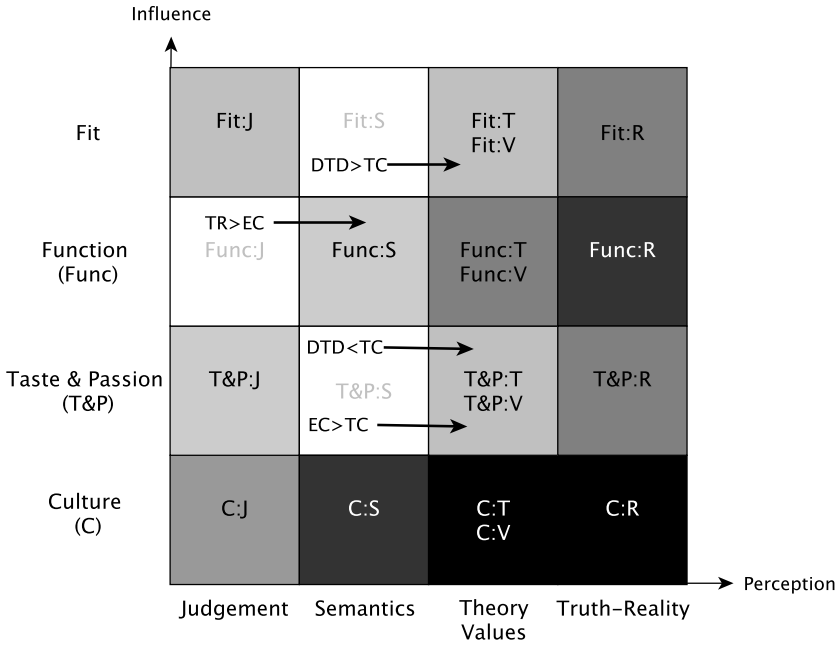


Figure 4.9 – Context states of DFD-0 at Figure 4.8 mapped to CDM.

command' based on semantics (Fit:S). It indicates that 'telemetry command' will not be processed unless it fits a certain demand placed by 'download telemetry data,' such as passing a fit to send test—size of the telemetry to be downloaded checked against destination, for example.

The context of 'telemetry request' is different from 'telemetry command,' however. As Figure 4.8 shows, 'telemetry request' is not recognised fully by a single diagram. Because 'telemetry request' is shared by two diagrams, identified as 'command' at the context-diagram level, and as 'telemetry request' at DFD-0, only the context state that is relevant to the depicted diagram level is shown. Thus the diagram shows the context of 'telemetry request' relative only to 'execute command,' since the external element that sends 'telemetry request,' 'ground system' at the context-diagram, is not shown at DFD-0.

'Telemetry request' places an influence of function on 'execute command' based on judgement. The context state indicates that 'execute command' is required only to execute the command properly. But because 'function' is based on judgement, the context state is declared to alert analysts to confirm the influence. If the perception of the function is not increased to semantics, analysts may at least indicate that the influence may not be identified correctly. Thus, analysts may work to confirm their judgement as they work with stakeholders to identify whether the influence is truly function.

Yet, there are two aspects that are worth noting about context states of elements that share two diagram levels. First, because the data flow at the context-diagram may unfold into more than one data flow, context states vary between the two levels of representation. At the context-diagram, for example, 'command' has different context states than 'telemetry request,' although the latter has unfolded from the former. 'Command' is influenced at the context-diagram by 'satellite system,' but 'telemetry request' is not influenced by 'execute commands' at DFD-0. This is because 'satellite system' influences all commands sent by any external element, but after a command passes this level, and becomes recognised as a specific command, it becomes influenced by another process. The second aspect worth noting, is how the influence of data flow on the main process at the context-diagram may not change when the data flow is represented by a lower level diagram. Because the demands that the data flow place on the system originate, initially, from elements outside the system, such as 'ground system,' they influence the system as a whole, and do not pertain to a specific internal process. In summary, the change in internal processes should not result in the change of the demands placed by external elements, but may result in the change of their influence that they place on external elements.

Figure 4.9 shows the possible state transitions of context states assigned to Figure 4.8, mapped in CDM. The figure shows, for example, that the influence of 'telemetry command' on 'download telemetry data' is taste-and-passion based on semantics, but may need to transfer to taste-and-passion based on theory or values. Although the influence may still transfer back to judgement, if the semantic statement that supports the taste-and-passion influence in nullified, analysts may rather support the influence further by a goal or a business rule. Similarly, other context states may show the same tendency, whenever possible, to transfer context states along the perception dimension. As a first priority, then, is for analysts to identify influences at the level of semantics, according to requirements. A further state transfer that analysts may seek to achieve, is to identify influences beyond semantics, to the level of theory or values. This way, analysts identify the context of the system, further than what typically DFD represent. This move to more precise knowledge about the context, prepare the move from analysis to design.

4.3.4 The context of the unfolding process of DFD

When context states are used to represent the context of system elements in DFD, they may allow analysts to follow the unfolding process on a more granular level. To describe a system in DFD, analysts follow a layered approach that starts at the context-diagram, and virtually has no limit of how many levels it may produce. At

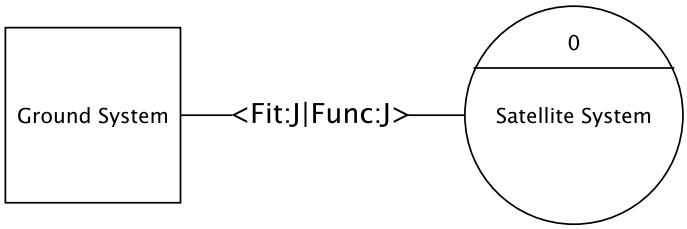


Figure 4.10 – The context of satellite system before adding data flow.

each level, steps are formed by an interplay of adding process then data, or data then process, in no specific order. But with the use of context states it is possible to separate the step of adding process from data by recognising the context state in between.

Because the unfolding process promotes analysts to observe context as closely as possible, context states allow analysts to represent the context of the DFD at each level before and after adding data flow. In the context-diagram, for example, it is not possible to describe the system at a higher level than the context-diagram. But with context states, it is possible to represent the context of process and terminators without using data. The question then becomes, regardless of data flow, whether there is a connection between the system and its external elements, and within what context, before deciding how the system and its terminators may communicate.

As a result, the context-diagram may be built in two steps. First, identify the context between the system to be developed and its external elements. Second, after identifying the context connection, identify how the developed system interact with its terminator through data. Figure 4.10 shows the context of the interaction between the ground system and the satellite system at a higher level, before the diagram describes how both elements interact. What the diagram shows, is how the satellite system relates to the ground system. It indicates that the satellite system demands fit from the ground system, when the ground system initiates an interaction. When the diagram at Figure 4.10 is translated into a context-diagram enriched by context states, the influence of fit placed by the satellite system may be mapped to the influence of fit placed on the data flow by the satellite system, as Figure 4.7 shows.

Similarly, it is possible to represent other processes, at the level of DFD-0 for example, before data flow are added. When processes are separated from data flow, it is assumed that data flow may vary while processes remain less variable. Processes are more stable because they are formed by other processes within. Thus one sign of internal change is the change of inputs and outputs. Therefore, there is

value, especially when the issue of variability is addressed, to represent the context of how processes relate to each other.

The diagram in Figure 4.10 shows a 'pre-thinking' stage of system analysis that reveals the initial context that results from composing 'ground system' and 'satellite system.' This initial context view of the system, allows the diagram to represent the context of the system beyond 'ground system,' hence providing an extended view of external entities. Even at this level, the separation between the 'ground system' and the 'satellite system' show architectural implications. Because elaborating on the specific form that the separation between both system elements may indicate the initial architecture view of the system. As a result, through the view that Figure 4.10 show, it is possible to proceed with the analysis with the following steps:

1. Evaluate the relationship between the 'ground system' and the 'satellite system.' At this step the analysis may involve exploring the external and internal process within the system and neighbouring systems, which starts from the view that Figure 4.10 provides.
2. Translate the requirements to produce a functional view of the system using data and process. This is performed by the traditional DFD view.
3. Use the DFD functional view, requirements, and system goals, to produce a design solution that achieves the functional aims of the system, and other non-functional aims.

4.4 Summary and conclusion

Two of the context themes identified from literature were represented by adapting two approaches to system analysis, DFD and the unfolding process. The first theme, context as an act of making connections, is manifested in data flow that connect processes in DFD. The second theme, context regress endlessly, is supported by the unfolding process, where elements are added in a stepwise process while maintaining a close observation of changes in context. Thus to represent the five themes of context identified from literature, an approach to enrich DFD using context states of the CDM is presented.

Although DFD diagrams share similar attributes with the unfolding process, they limit context to the concept of setting system boundaries. But when context states are added, it is possible to have a model that relates more to the context of the system than what the classical representation offers. Another extension

of how the DFD represents the developed system, is manifested in the analysts' ability to represent the context of the system independently from data flow. As a result, it becomes possible to reason about how the system may vary before and after data flow are represented. The use of context states in this manner, is shown in the example of representing a context-diagram on a higher level by having a process and terminators without data flow, but connected with context states. The unfolding process then is followed, when analysts assign context states between processes first, then add data flow second. Both levels of representation introduce different views on how to vary the system on the level of DFD data and process.

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Part

III

Proof of Concept and Conclusion

Chapter

5

Proof-of-Concept: Requirements of the Voter Mark-off System

> In general, we look for a new law by the following process. First we guess it. Then we compute the consequences of the guess to see what would be implied if this law that we guessed is right.
>
> Richard Feynman

Contents

5.1 Introduction

In the second part of this thesis (Part II) I presented how to represent context of individual elements (Chapter 3) and how to model the context of a system of elements (Chapter 4) when context states are mapped to DFD elements. Here, I demonstrate how to apply context states to the analysis of requirements as modelled by DFD, which shows the functional view of the system. The study is based on the requirements of a Personal Digital Assistant (PDA) based system to mark voters as being voted on an electoral roll for the Australian ACT and Federal parliamentary elections.

5.2 Goals

The main aim of this study is to demonstrate the effect of representing context in terms of states, based on influence and perception, on system decisions in terms of choice of variation. This is done by the following:

- Show how to assist analysts to identify which elements that are possible to vary and how, and which elements that are *not* possible to vary and why.
- Show how to assist analysts by declaring the source of their knowledge about when variation is possible and when it is not.

From previous two points, the following operational questions are answered by the results of the study:

1. Do context states help to identify elements that may be varied and other elements that may not?
2. Do context states identify the basis for possible sources of variation?

By reading the requirements, and applying the context approach, it is possible to recognise that both questions may not be easily answered by reading the requirements alone. But after the requirements are mapped using DFD, and the context is modelled, it is possible to identify how both questions are answered for each system element described by the requirements.

5.3 Setup

In 2001 the Australian ACT and Federal parliamentary elections moved from the manual voting system to the new electronic system eVACS (electronic Voting and Counting System). But before and after the system was implemented, the process of marking voters as having voted was done manually. Recently, the development of a new system to mark voters as being voted electronically has commenced using two devices: a handheld PDA, and a central PDA (C-PDA). The scenario of use is as follows: a polling official is approached by a voter whose name is searched using the PDA until (s)he is identified, and then (s)he is issued with a ballot paper or directed to an electronic voting system eVACS machine, and marked (in the PDA) as having voted. The PDA stores electoral roll data remotely on the C-PDA where it can be viewed later to establish who did not vote and/or who voted more than once.

The study was conducted by analysing requirement scenarios presented in an event-action table, consisting of 87 rows/statements and five columns (number, event, action, conditions, and a requirement code). The table was divide into three major activities: the scenario of accessing the PDA (14 rows/statements), the scenario of marking a voter of the electoral roll (37 rows/statements), and the scenario of data transfer (36 rows/statements). The analysis included two participants: an analyst, and a member of the development team (collaborator). The analyst studies the system through the set of requirements provided by the collaborator, builds the DFD diagram, and assign context states. The results, then, are discussed with the collaborator comparing actual events that occurred during and after the development and implementation of the system. Requirements presented by the collaborator are given in Appendix B.

5.4 Design of Study

The design of the study is divided into two parts: first build the DFD, then assign context states to its elements. In the first part, the DFD is built by drawing a context-diagram, which is followed by DFD-0. When the DFD is built, context states are assigned to DFD elements, mainly process and data flow.

5.4.1 Building the DFD

The analysis that represents requirements in a DFD is built in two stages. The first stage represents the functional representation of the system without

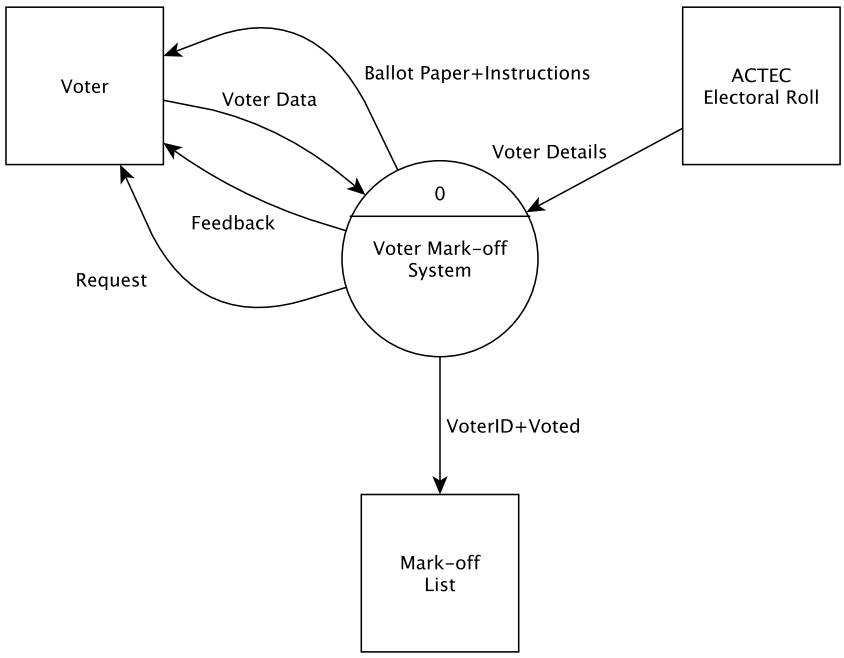


Figure 5.1 – The context-diagram of the voter marking system.

DATA DICTIONARY

Voter Details=FirstName+(Middle Name)+Surname+Address+DoB

Voter ID=Unique ID *generated by the system

Voted=['Yes'|'No']

Voter Data=FirstName+(Middle Name)+Surname+(Address)+(DoB)

Feedback=(MiddleName)+(Address)+(DoB)

Request=['MiddleName'|'Address'|'DoB']

Instructions=*Verbal directions on what to do with the Ballot Paper and where to submit vote.

Table 5.1 – DFD data dictionary.

the implementation decisions mentioned in the requirements, identified as 'the Voter Mark-off System.' The second stage shows the system and some of its implementation decisions that have been made by system developers, identified as 'the PDA-based Voter Mark-off System.' The process of building the DFD, on both stages, starts from the context-diagram, then represents processes at DFD-0.

First stage: Voter Mark-off System

It is possible to distinguish between the implementation details of the system, as a result of decisions made by stakeholders and system developers, from the functional description of the system. The DFD model starts with the context-diagram, which represents the system main process, then decomposes into the DFD-0.

Context-diagram: Figure 5.1 shows the context-diagram of the PDA system, and Table 5.1 shows the DFD data dictionary. The system consists of the main process 'Voter Mark-off System,' and the external entities (terminators), 'ACT Electoral Commission (ACTEC) electoral roll,' 'voter,' and 'mark-off list'. What follows describes the functional scenarios of the system from the perspective of the aforementioned terminators.

- *ACTEC electoral roll*: before the voting commences, Voter Details are stored in the system with a generated Unique ID for each Voter and Voted='No.'
- *Voter*: when the voting commences, Voters provide their First Name and Surname, which are immediately used to search all stored Voter Details (Electoral Roll). If more than one instance of Voter is found and has not voted (Voter='No') then 'Middle Name' is requested or fed back for confirmation. If there are still multiple instances of Voter then 'Address' is requested or fed back for confirmation. Same process applies again for 'DoB' if there are still multiple instances. Once voters are identified uniquely they are marked off as having voted (Voted='Yes') and then given one Ballot Paper with Instructions.
- *Mark-off list*: at the end of election day, 'Voter ID' and Voted for each voter is produced by the system that may be used later for analysis.

DFD-0 Figure 5.2 shows the processes within the main process in the context-diagram, which forms the DFD-0. What follows describes each process.

- *Import Electoral Roll*: receives 'Voter Details' from (terminator) 'ACTEC electoral roll' and generates 'VoterID,' creates 'Voted' and assigns it 'No,' then stores 'VoterID+VoterDetails+Voted(='No') in 'Electoral roll for Mark-off.'
- *Establish Voter Validity*: when 'Voter Data' is received, the system checks for validity. It searches for the details of voter (FirstName+Surname+(MiddleName)+(Address)+(DoB)) and returns with 'Feedback' or 'Request' to the Voter. If the 'Voter' is uniquely identified the 'Voter' is approved (Voter

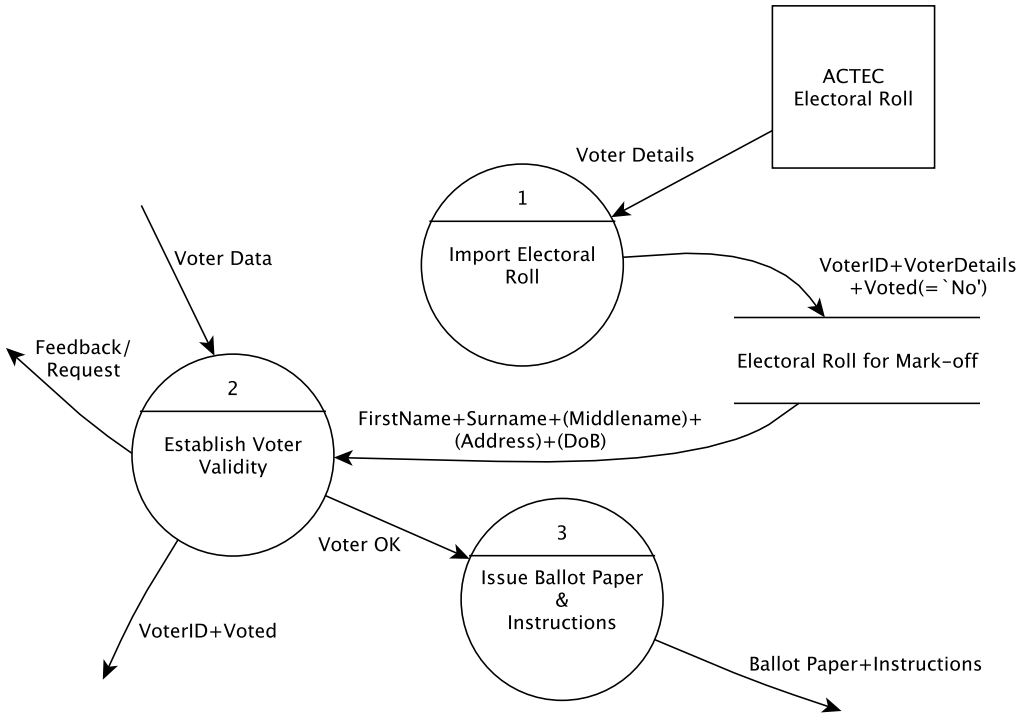


Figure 5.2 – A DFD-0 of the voter marking off system.

OK). At the end of the election the VoterID and voting status (Voted) is produced of every Voter in the stored list.

- *Issue Ballot Paper & Instructions*: when the 'Voter' is identified the process signalled to issue/hand a 'Ballot Paper' and voting 'Instructions' to the Voter.

Second stage: PDA-based Voter Mark-off System

The second level is the result of two solutions of how the system may mark-off voters from the electoral roll. First, to use a PDA to search and mark-off voters. Second, to have a polling official (PO) to operate the PDA. Adding both elements to the system transforms the DFD-0 and as a result the context-diagram. What follows shows how the DFD-0 is transformed, then show the resulting context-diagram.

DFD-0: Two processes are added to the DFD-0 (Figure 5.2), 'Performing PO Operations' and 'Search for Voter.' Figure 5.3 shows the transformed DFD and what follows describes newly added processes.

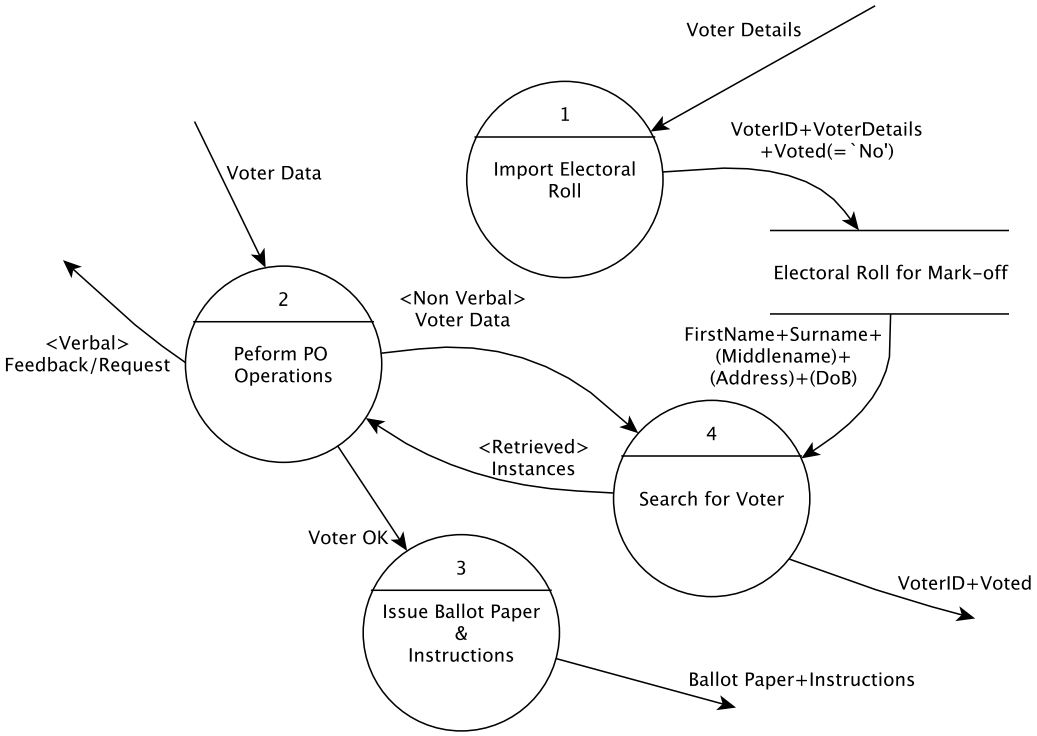


Figure 5.3 – The DFD-0 after adding 'Perform PO Operations' and 'Search for Voter' process, to represent the use of PO and PDA by the system.

- *Performing PO Operations*: communicates with 'Voter' through Feedback/Request and receives 'Voter Data' (verbally) from 'Voter.' Based on search results and communication with 'Voter,' the process identifies sends 'Voter OK' to issue 'Ballot Paper' and 'Instructions.'
- *Search for Voter*: receives 'Voter Data' in text from 'Performing PO Operations,' then the result is sent back to 'Performing PO Operations'—displayed results: FirstName+Surname+ (MiddleName)+(Address)+(DoB).

Context-diagram The context-diagram in Figure 5.4 shows the re-scope of the system, based on what may be developed as part of the PDA, and what is outside the PDA.

- *Inside PDA*: processes that perform operations directly with the data storage 'Electoral Roll for Mark-off,' 'Search for Voter' and 'Import Electoral Roll,' are kept within the boundaries of the PDA, that is the main process 'Verify Voter.'

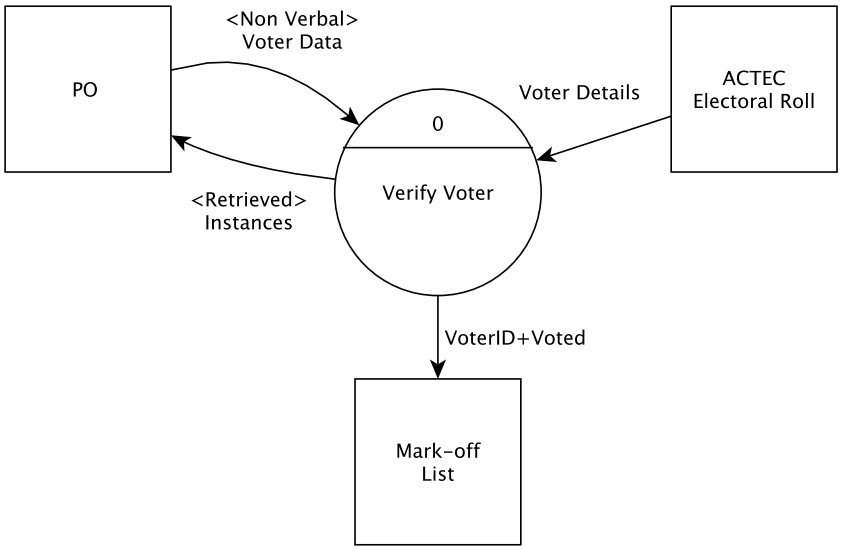


Figure 5.4 – The context-diagram after the re-scope of the Voter Mark-off system.

- *Outside PDA*: 'Perform PO Operations' and 'Issue Ballot Paper & Instructions' are performed manually with the assistance of the PO. The PO interacts with the 'Voter' to transform 'Voter Data' from being verbal to non-verbal. 'Ballot Paper' and 'Instructions' are provided to 'Voter' by PO after his/her details are verified.

5.4.2 Assign context states

The context states within the system of marking off voters may be assigned to three levels. The first level has a single context state that applies to the system as a whole. The second level shows context states of processes and terminators without data flow—refer to Section 4.3.4 for details. The third level shows the context states assigned to process/terminators and data flow.

The assignment of context states starts from the view of the system depicted in the original context-diagram (Figure 5.1) and re-scoped context-diagram (Figure 5.4). As mentioned in Chapter 4, the default context state of DFDs is *function* based on *semantics* (Func:S), thereby allowing to focus on states that deviate from the default context state. To avoid presenting and analysing redundant context states, three representative context states are presented in what follows. The order of assigning/identifying context states follows the order of presenting the DFD diagram. The full list of identified context states are listed in Appendix A.

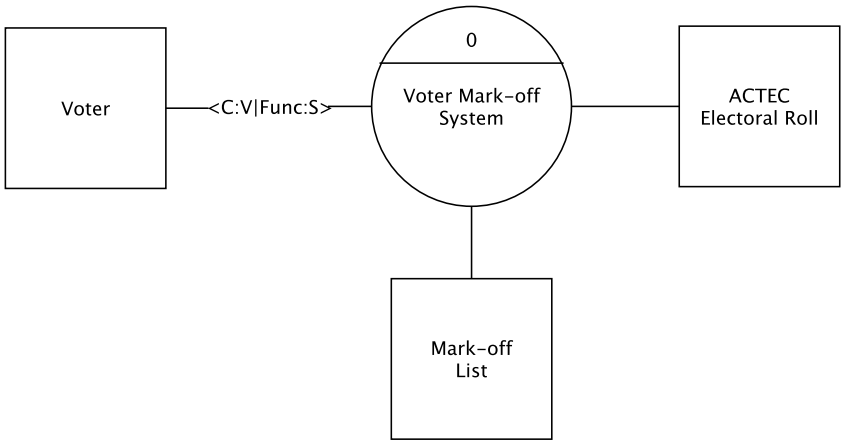


Figure 5.5 – The context-diagram showing only process and terminators. Context states are assigned to the connection between the main process and terminators.

Context-diagram (Figure 5.1)

The context state of context-diagram may be assigned at three levels. The first level represents the context state of the system as a whole. It is typically represented by one context state. The second level represents the context state of terminators and process without data flow. It assumes the context of the elements of the DFD before adding data flow. The third level are context states between data flow and process or terminators. Only the third level may be assigned to the DFD model, as Figure 5.6 shows. For the second level, data flow are removed, and elements of the diagram are connected and assigned states, as Figure 5.5 shows. The context state at the first level is not represented by DFD, but described.

- *Context state of the diagram as a whole*: the system to mark-off voters is part of the larger election system. Before introducing the PDA system, marking voters from the electoral roll was performed manually. The assumption is that the new system *preserved* the structure, the procedures, and the goals of the old system. Because the system new system preserves the old system, the new system as a whole becomes under the influence of culture. But because the fact that the system has preserved the structure of the old system is an assumption, not confirmed by any requirement, the cultural influence becomes baed on judgement. As a result, the context state of the system as a whole is *culture* based *judgement*[1] (C:J).

[1]The perception of judgement here, and in other context states, is my own judgement, as the analyst of the system. In other use of judgement, outside of this study, it may indicate the judgement of a single analyst or a team of analysts.

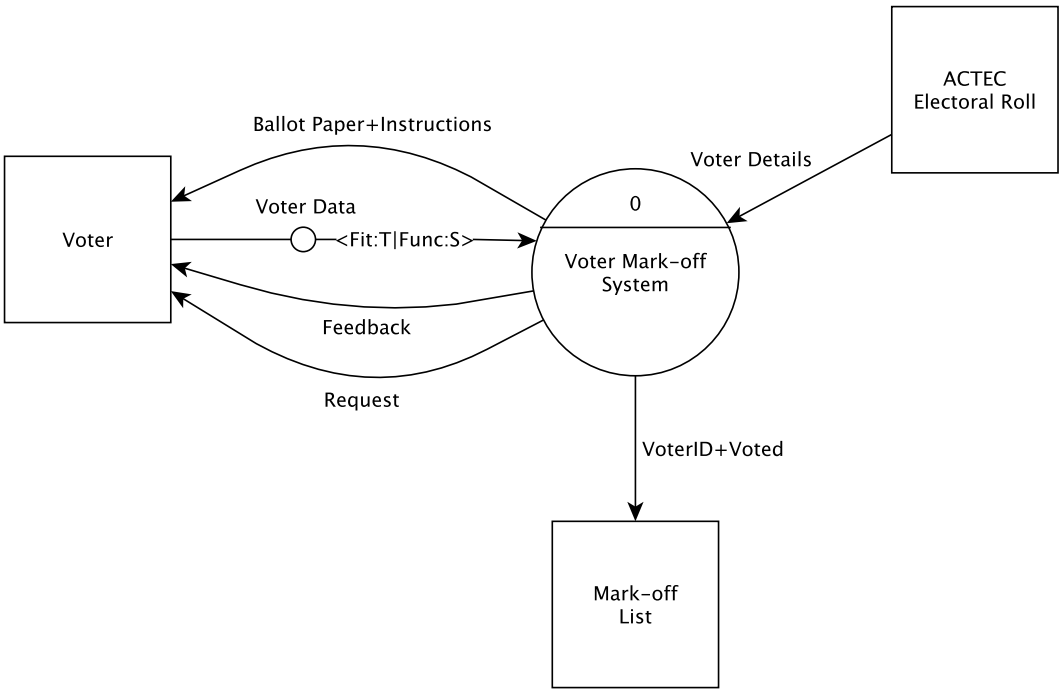


Figure 5.6 – The context-diagram showing the context state of 'Voter Data' and 'Voter Mark-off System'.

- *Voter–Voter Mark-off System*: Figure 5.5 shows the context of both 'Voter' and 'Voter Mark-off System.' Voters are under the influence of 'Voter Mark-off System' when they interact with it. The interaction includes providing personal information, and responding to requests and feedback from the system. The interaction is part of the culture of the system as a whole, which demands from each voter to provide this information. As a result, 'Voter' becomes influenced by *culture*, and because the ACT Electoral Commission defines what should be obtained from voters as part of a predefined procedure (laws/values), *culture* becomes based on *values* (C:V).

  But the influence that 'Voter' applies on the system does not relate to the culture of the system. It relates more to the goal of the voter from interacting with it. Voters require primarily from the system to provide functionality. Accordingly, 'Voter' places an influence of *function*. The influence is derived from the description of requirements, that indicates that voters use the system to be issued a ballot paper to vote. Thus the influence of *function* becomes based on *semantics* (Func:S).

- *Voter Data–Voter Mark-off System*: 'Voter Mark-off System' places an influence of *fit* based on *theory* (Fit:T) on 'Voter Data,' because 'Voter Data' must match the stored data in the system before a 'Voter' is marked off. The search

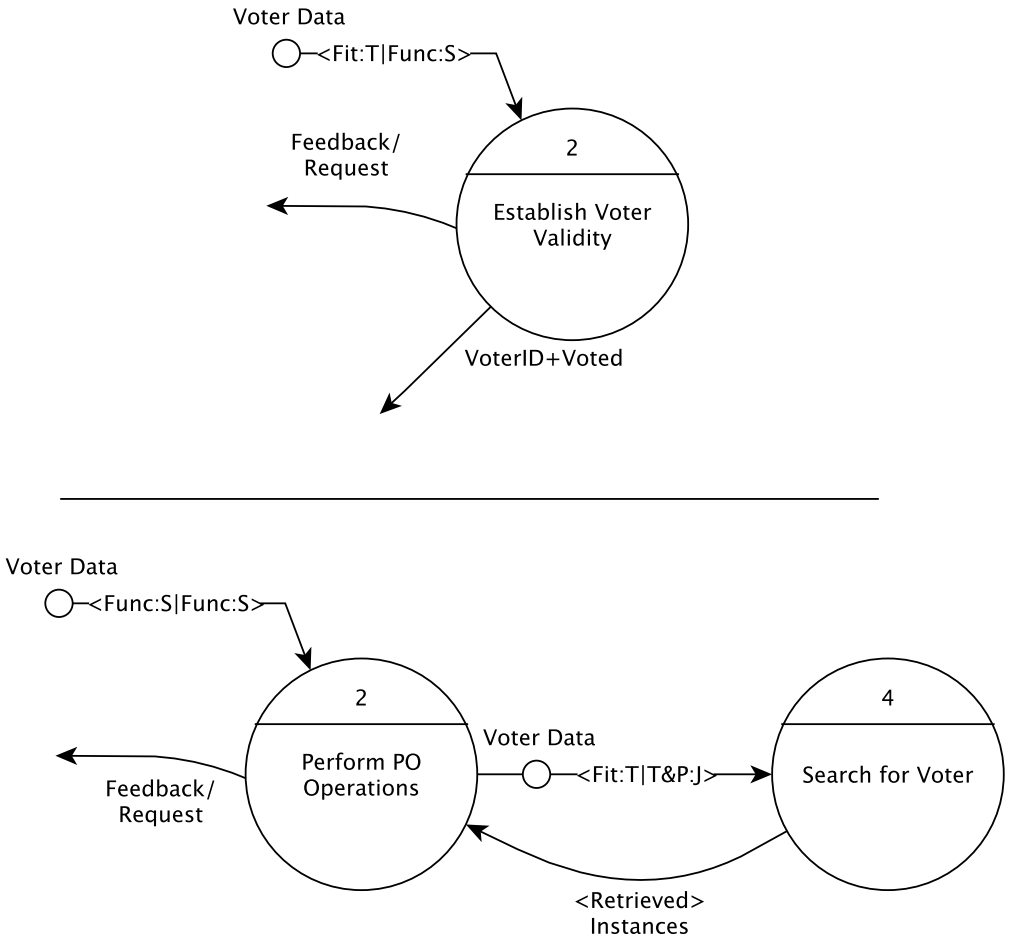


Figure 5.7 – Validating Voter before (above) and after (below) adding PO and PDA.

described by more than one statement in the requirements form a *theory* of how the system operates, which supports that 'Voter Data' is under an influence of *fit*. On the other hand, 'Voter Data' places a force of *function* based on *semantics* (Func:S) on the system. What the data demands from the system, at this stage at least, is to provide the needed functionality to process the data as described by requirements.

DFD-0 (Figure 5.2,5.3)

The context states of the DFD-0 changed from Figure 5.2 to Figure 5.3 after adding PO and PDA. What follows identifies the context state before introducing PO and PDA and after.

- *Before: 'Establish Voter Validity' (EVV) and VD*: EVV places an influence of

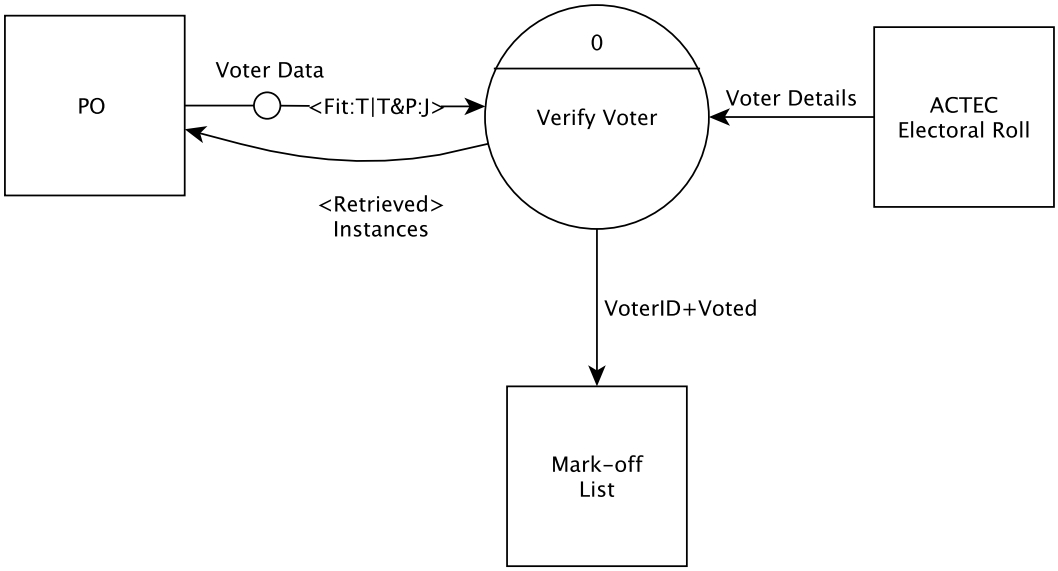


Figure 5.8 – The context states assigned to the re-scoped context-diagram.

fit based on *theory* (Fit:T) on VD—this states is also shown at the context-diagram. EVV places *fit* on VD because it must match the stored data.

- *After: 'Perform PO Operations' (PPO) and VD*: When PPO is added to receive VD, the context state of VD becomes *function* based on *semantics* (Func:S). The force of *fit* is moved to be placed by 'Search for Voter'(SV) on VD. Figure 5.7 shows the change. PPO according to requirements does not match the VD itself. It uses VD to search for it using 'Search for Voter' (SV). PPO, then, demands from VD to be functional for it to be searched using SV.

- *After: VD and SV*: the influence placed by VD on SV is *taste-and-passion* based on *judgement* (T&P:J). VD demands for better performance when SV performs its search. It may also reflect the demands by PPO. Representing the context state of the connection between PPO and SV, PPO places an influence of *taste-and-passion* based on *judgement* as well. The influence that SV applies is *fit* based on *theory* on VD. It is the same influence that EVV placed before introducing PO and PDA.

Context-diagram (Figure 5.4)

Figure 5.8 shows the context state of 'Voter Data' in the re-scoped context-diagram. It shows the same context states assigned to 'Search for Voter' at DFD-0. The influence placed by 'Perform PO Operations' on 'Search for Voter' is moved at the

context-diagram level to represent the context of PO and 'Verify Voter,' which is *taste-and-passion* based on *judgement*.

But if the context of using the 'Verify Voter' (PDA) to search for 'Voter Data' is represented through a direct connection with PO, without the data flow, a different context state is obtained. 'Verify Voter' applies a force of *taste-and-passion* on PO based on *semantics*, because it allows the PO to search the system using more than one element, 'First Name,' 'Address,' 'DoB,' and so on. Similarly, PO may search for 'Voter' without using the system, by going back to the manual electoral roll. But the use of PDA is preferred over the manual system. Accordingly, the influence of *taste-and-passion* is applied on 'Verify Voter' based on *judgement* by PO.

5.5 Analysis

Results are analysed in two parts. First, the implications of context states on system variation are presented. Then further implications obtained after sharing the results with the collaborator are presented.

5.5.1 Implications of context states on variation

The analysis attempts to answer two of the previously posed questions. First, 'Do context states, when assigned to DFDs, help to identify elements that may be varied and other elements that may not?' Second, 'Do context states identify the basis for possible sources of variation?'

Question 1: Do context states help to identify elements that may be varied and other elements that may not?

Context states implied how the system may vary through the influence model's four forces (*fit, function, taste-and-passion,* and *culture*). The analysis identifies also, when applicable, the role of interrelations between the four forces on variation. Influences are identified on three levels, the level of the system as a whole, the context of processes without data flow, and the context of data flow and process.

The context-diagram that Figure 5.1 shows describes how the system functions before introducing the PDA as a solution. The first context state identified is assigned to the context of the act of marking off voters. It forms the contextual underpinning of the system. Because the manual system has been used as part of the election process, it became part of the culture of an existing system. Thus the system, as a whole, the need for it, how it functions, and the success or the failure of

the new system; may all be developed and evaluated under the influence of culture. But because it is under the influence of culture, it is possible to vary the system, as long as the change will introduce improvements over the old system. The force of culture forms the first implication on the opportunity to vary the system.

Within the influence of culture there exists the connection between 'Voter' and 'Voter Mark-off System,' the latter being the main process within the context-diagram. Because the manual system is a 'legacy system,' what it demands from 'Voter' is also part of that culture. Accordingly, when a voter follows the procedures enforced by the system, on how to be marked as having voted and how to obtain a ballot paper, following these procedures and satisfying its demands; is a response to the influence of culture. But the demands of 'Voter' are functional. That is, when voters arrive to receive their ballot papers to vote, they may generally expect functionality from the system. If this was not true, it may not be possible to change how voters are served by the system. 'Voter' then applies a force of function on the system as a result.

'Voter Data' comes under the context of the connection (culture/function) between 'Voter' and 'Voter Mark-off System.' It forms a context connection with the main process when 'Voter' provides his/her details. When 'Voter' sends data to 'Voter Mark-off,' 'Voter Data' becomes under a force of fit by 'Voter Mark-off,' because it must match the data stored by the system. But similar to 'Voter,' 'Voter Data' expects functionality from the main process. The functionality it demands is the functionality that the system offers.

The three context layers together, the context of the system as a whole, the context of processes and processes/terminators, and the context of data flow and process/terminators; may explain the source of variability introduced later. Each context state unfolds to reveal the context within it. In the example of the context states assigned to the context-diagram, the implication of variability may be identified by analysing the three context states together. Because the whole system *is* influenced by *culture*, the influence indicates that the system may allow change to the previous system when the new system is developed. Indeed, introducing improvements to the system is possible, because stakeholders may have saw the opportunity for change. But because the system is also under an influence of *culture*, the system may *preserve* its structure first, as shown by the first depicted DFD, then introduce variation from within. On the level of 'Voter,' for example, it is possible to vary how to mark-off voters from the electoral roll, because they only require functionality from the system—they place an influence of *function*. This is also true on the level of voter's data (see Figure 5.1), yet another source of variation. Because 'Voter Data' places an influence of *function* on the system, it is possible to replace the system with one or more solutions that provide similar or

enhanced functionality. This was shown by the steps that followed that introduced PO and PDA.

PO was introduced first as a process in Figure 5.3, represented as 'Perform PO Operations.' Together with 'Search for Voter,' it replaced the process 'Establish Voter Validity,' shown by Figure 5.2. Figure 5.7 shows the context states before and after 'Establish Voter Validity' was replaced. In both DFDs, 'Voter Data' showed an influence of *function*, which allows 'Establish Voter Validity' to vary as long it performs the required functionality. When PO and PDA were added, the required functionality was divided between PO and PDA. The former communicates with the voter and operates the PDA, and the latter searches for voter's data.

But the opportunity to vary how to mark-off voters was not only indicated by the context of 'Voter Data,' it was also indicated at a higher level, according to the influence of 'Voter.' As previously discussed, the influence of 'Voter' is *function*, which allows the system to vary how to provide voters with the service that they require. It may imply varying what type of data to search for in the system, or imply that voters may not provide any data at all, if the *culture* that influences the system as a whole allows it.

Yet, while context states have implied where the system may vary, they also show where the system may not. Within the overall influence of *culture*, the system has continuously applied an influence of *fit* on the data searched by the system. The same influence did not change even when the PDA was introduced. 'Establish Voter Validity' placed an influence of *fit* on 'Voter Data,' and when 'Perform PO Operations' and 'Search for Voter' were introduced, 'Search for Voter' placed the same influence on 'Voter Data.' When the system was re-scoped, the influence of *fit* appeared to be applied by the main process on 'Voter Data.' If the influence of *fit* to be varied, the change must come from the influence of *culture*. In fact, when the context state of the connection between PO and 'Verify Voter' is assigned at the context-diagram level, the PO does not have to use the system if he/she chooses to. As a result, the influence on PO is *taste-and-passion*. But for the data, even if the system is changed, it must *fit* the stored data, whether on the manual record or the PDA database.

There are two main influences within the *overall* influence of *culture* placed on the system as a whole. The first is the influence of *function* placed by 'Voter' on the system. The second is the influence of *fit* placed by the system on 'Voter Data.' The *function* influence has allowed the system to vary how it provides service to voters, by using the manual system, and later by using both PO and PDA. The *fit* influence placed by the system on 'Voter Data' remained, as shown on different DFD levels, even when the way to enforce the *fit* has changed. Both influences, however, may

be transformed if the culture of the system itself changes, such as not searching for voter data.

Question 2: Do context states identify the basis for possible sources of variation?

Context states have identified how each influence was perceived through four sources of knowledge: *judgement, semantics, theory*, and *values*. With each source, it is possible to imply how reliable the assessment of the system's context is.

Influences were identified under *judgement* when they were not mentioned in the requirements. For example, the context state of the overall system was identified as *culture* based on *judgement*. But because the requirements did not mention anything on the background of the system, which may answer the question whether the functionality is derived from actual processes applied in reality or not. It was assumed that the processes described in the requirements were the actual processes applied by the manual system before the requirements were developed. Similarly, the influence applied by 'Perform PO Operations,' *taste-and-passion* on 'Search for Voter' is based on *judgement* because the influence was not mentioned in the requirements. But the search for voter's data may indicate that the process, if not measured carefully, would take a long time to return results. Accordingly, it is expected that the time it takes for the process to return results may be determined according to the preference of the operator within 'Perform PO Operations.' As in the case of the influence of *culture*, *taste-and-passion* may not be correctly identified if the *judgement* is misguided. Nonetheless, it may alert designers who may implement the system to consider performance when 'Voter Data' is searched for in the PDA.

Contrary to *judgement*, influences based on *semantics* are identified directly from requirements. Because the system description in the DFDs are derived mainly from requirements, most of the influences were perceived under *semantics*. For example, the influence placed by 'Voter Data' in the main process on the context-diagram, is *function* based on *semantics*. The process of entering the data was described within the requirements to indicate the functionality performed by the PO, and have not indicated any other influence. Statements may not indicate other influences, but they also do not limit what might be derived from them. For example, the statement that describes the functionality of the use of 'Voter Data' is described by requirements to indicate an influence of *function*. But it may be used as a reference for an implementation that applies an influence of *culture*. If the 'Voter Data' is described by the data dictionary to have non-standard characters that are culturally related, the influence may still be based on *semantics*, but the

influence may become *culture* not *function*.

A higher perception level than *judgement* and *semantics* is identified through *theory* and *values*. *Theory* is identified for the search of 'Voter Data' for when the influence of *fit* based on *theory* applied by 'Search for Voter.' The *theory* is assigned based on two sources, first, the external source that describes the technology for searching the data. The second source is the internal statements that describe the same influence in more than one place in the requirements. Both sources support the use of *theory* as a basis for perceiving the influence of *fit*. Searching for the voter's data is part of the goal of developing the system. It appears when officials search for names manually, and when it is described by the specification of PDA search. Compared to *theory*, *values* are manifested in laws or regulations that govern the election process. The only *influence* based on *values* appears in the influence placed by 'Voter Mark-off System' on 'Voter.' The demands from the system on voters, in this case, to *fit* their data to the stored data in the system, is based on laws and procedures set by the ACT Electoral Commission. The description of the law, which may include constraints and restrictions, may specify what data may be used, and what data may not be.

The only perception level that was not identified in the system, is *truth-reality*. Typically, in order for elements to be perceived under *truth-reality* they have to be implemented. But because the described system is not developed yet, *truth-reality* does not apply to this level of analysis.

5.5.2 The DFD review

When assigned context states were presented to a member of the team who developed the original system (collaborator), he confirmed what the context states implied in terms of where the system may vary. But he also offered new interpretations of the implications of the context states that were not originally foreseen. The collaborator have confirmed that fit placed by the use of search for the 'Voter Data,' proved to be significant to the success of the system. Furthermore, the collaborator's comments have indicated that context states may also serve as a tool to identify possible misfits.

The development team, at the day of election, had trouble finding few names when they searched the system. There were several reasons behind this, as the collaborator explained, the most relevant to the development of the system was that some names had non-standard characters that could not be searched. As a result, voters had to be marked as having voted using the manual system. This was reflected by the context states on the DFD in the force of fit placed by 'Search for Voter' on 'Voter Data.' It implied that if the name does not fit the characters

stored or used by the PDA, or if the name does not fit the list stored in the system, the search will fail.

The review has indicated that context states may be used to identify possible sources of misfit. Because the DFD shows the context of the system through requirements, different levels of influence reflect different misfit levels as well. Thus when the context states identify a force of fit, it indicates that a misfit may be severe and may lead the system to fail its task. As a result, context states may guide developers to concentrate on how to prevent misfits that occur within the context of a force of fit from occurring, such as failing to search for a name in the PDA. But misfits that occur within a taste-and-passion context are less severe, because in concept, it is possible to use a less preferred choice instead, and avoid system failure. Accordingly, in the case of a taste-and-passion, the system may plan to alternate between ways to achieve its goal, but in the case of fit, it makes an effort to ensure that the system does not fail.

5.6 Discussion of results

Further discussion of the results of the study is presented. It includes a discussion of further implications of context states, first, on the influence dimension, second, on the perception dimension. The first provides further discussion on the implications on the use of context states as a tool to identify/classify system misfits. The second discusses issues with the perception of influences. It explores possible reasons why context states may not be assigned correctly, and how analysts may improve their assessment of context states. The last part presents discussion of possible threats to validity.

5.6.1 Further implications on influence

As indicated by the discussion with the collaborator is the implication of influences on the process of identifying system misfits. When identified, influences form a context of misfits within DFDs. DFDs typically intend to represent what the system should do, but using influences it is possible to identify what the system should not do, and prioritise them. Three misfit examples identified by the study illustrate how influences serve as useful indicators of system misfits. The first misfit identified is associated with the influence of *fit* placed by search on 'Voter Data.' The second misfit is identified by the influence of *taste-and-passion* placed by 'Voter Data' on the system manifested in the main process 'Verify Voter.' The third misfit is battery failure, which has not been represented by the DFDs of the

study. What follows describes each misfit.

- *Fit on 'Voter Data'*: a misfit occurs when the voter's details are entered by the PO and the system returns 'no match found' even though the details are in the system. Note that this misfit is more severe than other misfits because it occurs within the context/influence of *fit*. The PO has no way to know whether the person is suppose to be in the system but he/she is not, or if the voter is actually stored in the system but the search cannot find him/her because of a fault. When this misfit is reviewed with the collaborator it was possible to recognise that three tactics were applied to prevent it. First, POs are given five data fields: first name, middle name, last name, data of birth, and address. Searching with any of the five elements should help to reduce the chances that a mistake in one or more than one data element may prevent the voter from being found. Second, POs are able to obtain search results as they type first letters, instead of waiting for the name to be typed fully until the system starts its search. This tactic enables the PO to speed the process of finding the voter's data, and reduce the chances that the search result return 'no match' because one letter is stored incorrectly. Third, the manual system is preserved as an alternative choice to register voters if the system fails. In few cases where the voter is not found in the manual system, he/she would be asked to declare their identity and sign a declaration that they have voted.

- *Taste-and-passion on 'Verify Voter'*: 'Voter Data' demands from 'Verify Voter' to return search results in a preferred speed. Not being able to meet the preferred speed is a misfit. While it is less severe than other misfits, such as not finding stored data, but it represents a failure to introduce an advantage of using the PDA over the manual system. The way to improve the performance of the system is to limit the fields that the PO may search for. Another way is to allow partial name search. Results may be obtained with the first letters of the voter's name without searching for the whole name.

- *Fit on battery*: the battery must *fit* the time of service that POs need to use the system. If the battery fails, because its power is exhausted or it reaches end-of-life, the system will stop providing service. If this loss of service is not handled properly, by introducing an alternative device to continue providing the service, the system may result in a misfit. To address this issue, the PO may recharge an lose mobility to recharge the device when attached to a power-cord. The PO may also be informed when the device is running on low power to avoid the misfit of the sudden loss of service while the verification process is undergoing.

Identifying misfits in early stages of system development using context states enables analysts to plan for counter measures. Counter measures targeting a specific misfits is equivalent to what Alexander [2001] has identified as the process of strengthening centres. When search becomes identified as a centre, for example, one way to think of its strength, is its ability to avoid misfits. Maintaing context states as focal points may drive the system to achieve targeted quality attributes by measures to avoid misfits under different contexts.

5.6.2 Further implications on perception

Perception level may imply the degree of reliability that an influence has been identified correctly or not. The use of knowledge levels, such as *judgement*, is to allow analysts to share their confidence or doubt about certain levels of influence they have identified. It may allow analysts to identify clearly that some of the influences identified are assumptions not facts. The study of the PDA system shows several context states that may serve as examples where assumptions were declared, and can be easily challenged.

Consider the example of the assumption that the system of marking voters as a whole is under the influence of *culture*. For example, it is possible argue that the system as a whole is under the influence of *function*, because it supports the goal of maintaining the integrity of the whole election process. In fact, this may be actually true. The choice to account for voters may have *function* as its early source of utility. Furthermore, *function* would be the common influence of other systems that mark voters in a similar way. But to argue for *culture*, the way systems that share the same functional influence vary, is to adapt to the specific way that this process is executed. This specific way becomes influenced by *culture*. But because both arguments are logical arguments, they are based on *judgement*, not evidence from requirements (*semantics*) or *theory*. The role of the analyst as a result is to improve the chances of the his/her argument by gathering more information.

When an analyst declares that an influence is based under a certain level of perception, it is expected that necessary effort is made to transfer lower perception levels to higher levels whenever possible. Consider the perception of *judgement*, which assumes no source of knowledge that it identifies except intuition and experience. But in order to formalise this judgement it either should be translated into a requirement or identify an existing requirement that supports it. The move from a perception of *judgement* to *semantics* in the form of requirements, for instance, should go through a process that verifies whether the perceived influence applies to the current system or not. If not, then the influence is totally dismissed, or reverted to its original functional influence.

The process of identifying context states could always be challenged even if the level of perception is high. If an influence is identified under *theory*, for example, the choice of theory may be challenged as well. Especially if more than one theory could be chosen. But act of declaring that the perception is at the level of *theory*, in its simplest form, means that the influence has further elaboration that is worth considering. The influence of *fit* placed on 'Voter Data' is based on *theory* because it is one of the core influences within the system that is referred to widely in the requirements. To argue against this influence based on how it was perceived, means that the links between the concept of search and the statements that form the *theory* must be broken. Within this deconstruction process lies the role of analysis.

5.6.3 Threats to validity

Areas of possible validity threats to the study is identified here. This includes the internal validity, the construct validity, and the external validity.

Internal validity: are the observed findings attributed to the presented approach or to other possible causes? Four factors are maintained to support the study's internal validity:

1. Because the system is already implemented and operational, the study did not influence the results of the project. For example, no results from the study influence the system's requirements either by adding/removing statements or offering new interpretations to existing statements.

2. To ensure the findings were not obtained as a result of a mature understanding of the requirements, the study was only conducted once. Any new information obtained after the study was completed, was not used to alter or improve the results. [2]

3. The only source of information about the system was the requirements document. No other sources were used during the study.

4. The results of the study conquer with the questions and issues faced during development. For example, after the context states were identified, a member

[2]This is true in regards to the process of reading the requirements and identifying context states. But the map was first built using two levels of representation, and was first published in Alshaikh and Boughton [2009]. When the approach developed further—following suggestions to simplify the approach—the approach was reduced to represent context using DFD. Accordingly, when the previous approach was replaced by the new one, the original interpretation was maintained.

of development team, was asked about some of the issues raised by the analysis. The issue with the search of names was a concern shared by the team when the system was development, which was predicated by the context states.

Construct validity: did the study effectively identify variation opportunities? To maintain construct validity to address the threat of researcher's expectancy no contact with the stakeholders or system developers during the study was established. This contributes to the validity of the study as it reduces the chance that the findings were a result of prior knowledge of the system. Furthermore, the analyst has no knowledge of similar systems leading to similar results.

External Validity: could the outcomes of the study be generalised to other cases? While this was a single study, some of the results are generalizable and comparable with other systems because some context states and their associated elements are not limited to mobile-based systems. For example, the context states associated with 'password' and 'search name' may be represented with the same states for other systems.

5.7 Summary and conclusion

The study demonstrated how the context approach could be applied to enrich the DFD representation of requirements. The study, first, constructed a DFD diagram that describes the functional view of the system, starting from the context-diagram, followed by DFD-0. After the DFDs were constructed, context states were assigned to the diagram.

Context states have implied how the system may vary within the limits and opportunities offered by its context. The use of the approach encourages the analyst to ask questions. The use of context states, also enables the analyst to reveal uncertainties about the meaning and implication of some requirement statements, in the attempt to reach to an acceptable level of clarity about the system's context. Context states have also identified significant design considerations, having implications on system specifications and quality attribute concerns such as performance.

When the results were reviewed by a collaborating system developer, the developer indicated that context states may also indicate system fits and misfits.

5.7 Summary and conclusion

Context states have predicted some of the challenges that system developers faced when the system was implemented. As a result, misfits and fits may be prioritised according to influence. *Fit* implies limited variation, which may lead to more severe consequences than a misfit under the *influence* of *taste-and-passion*. This new interpretation of what context states imply, may be the source of further studies on the use of the context approach to identify possible sources of system failure.

Design implications derived from the context states motivates the study to be extended to follow the same process to represent elements with architectural and usability significance. Context states, as the study suggests, represent a useful artefact to represent elements within requirements, and extended them beyond the scope of requirements, to approach various system concerns.

Chapter

6

Summary and Conclusions

Contents

6.1 Introduction

The aim of this thesis is *'to present a model of context that shows when to vary and when not to vary a system. Such a model should indicate the opportunity to vary the system when the context reflects soft demands, and indicate when it is not possible to vary the system because the context has strict demands. The model should also indicate when strict demands are based on conjecture, and when soft demands are based on strong evidence. The model should show different degrees of variation that the system may have through context.'* The thesis aim has been achieved through the model of influence and perception using the CDM, and the mapping of context using DFDs.

I present a summary of what is presented by the thesis on how to indicate implications of variability from context. The summary begins with a review of related work, then present the summary of contribution, followed by limitation of approach, then recommendations for future work.

6.2 Related work

As the main contribution of the research of this thesis, I provide a survey of work related to context states. The survey shows how various techniques have been used that are similar to what is used to analyse context, but either lack generality or have a different focus.

There are several approaches that identify system elements in terms of states from various software system concerns, such as: software architecture, product line engineering, and software metrics. For example, in software metrics, the Goal/Question/Metric approach (GQM) by Basili et al. [1994], measures how goals are achieved through view points. View points have two states, they are either objective or subjective. Similarly, in software architecture, utility trees [Kazman et al. 2000] scale quality attributes according to three levels: high, medium, and low. What makes these examples—and similar ones—different from context states, for one, is how analysts classify the system elements over time. At one time, what analysts who use GQM regard as subjective, may become objective at another time. But most approaches that classify system elements in the same way, rarely explore the dimension of time. As systems develop, they move from one stage of development to the other, as system developers aim to produce a system that functions as required. But if analysts consider, at every stage of development, how they classify elements and change their classification before the system moves to another development stage, they will start to observe what is recognised, here, as

context states.

Utility tree: Kazman et al. [2000] present an approach to classify quality attributes according to stakeholders concerns along two dimensions: the importance of a quality attribute to the success of the system, and the perceived risk associated with achieving each quality target—the assessment by the architecture team of how easy to achieve the target quality. Both dimensions are scaled using three levels: high, medium, and low. The utility tree starts from a single node, representing the utility of the whole system, then the system utility is divided according to identified system quality attributes. Quality attributes—recognised also as quality factors—are divided further into sub-factors. For example, security is broken down into 'data confidentiality' and 'data integrity'. Each sub-factor then is assigned an attribute characterisation. Each attribute characterisation provides a concert of generalised goals. For example, stakeholders may state 'customer details security is highly important,' after refinement, this statement is made more specific 'customer details are secure 95% of time'. Each attribute characterisation is scaled according to the two dimensions: importance of achieving the quality and the perceived risk of not achieving it.

The two dimensions used to scale attributes, while simpler, are complimentary to the context states presented here from two perspectives. First, Kazman et al.'s utility tree involves the perceived risk of the architect team as a second dimension. This is similar to the knowledge model of perception presented in Chapter 3. Architects have to predict how to achieve the target quality goals and assess their prediction to be shared and documented as part of the system. This prediction is also scaled on the degree of risk associated with the architect's prediction. But unlike the knowledge model, architects do not associate or identify any particular perception source to support their prediction. Basically what the utility tree identifies is the architects' judgements, equivalent to *judgement* in the knowledge model. While it is likely that architects would base predictions on pure judgement as a first attempt; judgement is likely to change and become more solid overtime. This change is not captured by the utility tree.

Second, the utility tree represents the level of importance of the target quality as prioritised by stakeholders. This is complementary to the *force model* of influence (Chapter 3). Stakeholders have to assess the level of importance of a certain quality relative to the success of the system. Compared to the force model, the highest ranked quality attribute by stakeholders does not necessarily mean that the system will fail if not achieved. This is because the consequence of not achieving a particular quality is not explicit. For example, in terms of priority relative to stakeholders, the performance of a an online system selling household

merchandise might be the same as the performance of an airport control system, both may have high priority. But in terms of consequence, the consequence of not achieving performance in the online system may be fundamentally different from the performance of the airport control system. The difference is typically realised by the architects' perception of the context, identified earlier as the common-sense approach. Using the force model, architects and stakeholders become more aware of the consequence when communicating using fit and function, for example, rather than high and low.

Feature modelling: feature models are extended from domain analysis by Kang et al. [1990] to be later used to represent variability for products as part of product line engineering [Kang et al. 2002]. A feature is an abstraction of characteristics that share variability and commonalities across products. A feature model represents three types: mandatory, optional, and alternative features. Features may be composed by a number of other features, as sub-features or as an implementation. When features are related, they form a tree structure, the root represents more general features and the leafs represent the more specific local features. Kang et al. [2002] also classify features according to functional and non-functional features illustrated by different features of a house: a flood control feature is a functional feature, and non-functional features are exemplified in characteristics such as cost, capacity, usage, and so on. When features are identified they can be prepackaged as standard items (for later negotiation), and features that are part of a specific custom-made product.

Fey et al. [2002] notes four limitations of feature models: *a*) the model does not explicitly define feature attributes and how each attribute relates to other attributes; *b*) the set of relations used to describe features and products are not sufficient for more complex examples—thus deriving new relations such as pseudo-features and provided-by relations; *c*) the feature hierarchy structure has redundancies that result in inefficient analysis algorithms; *d*) some inter-feature relations increase the depth of the tree structure of the model that could be reduced. Fey et al. suggest that *mandatory* and *optional* are reduced to only *require* relation.

Fey et al. [2002] suggest feature models may be improved by using meta-models instead. The defined meta-model introduces new relations: modify, require, and conflict. A similar extension of feature relations is suggested by Ferber et al. [2002] by identifying other feature interactions: intentional interaction, resource-usage, environment induced interaction, usage dependency, and excluded dependency. Lee and Kang [2004] extend feature modelling by recognising feature dependancies. Such dependencies are recognised during operations, such as

when a feature is used or modified, or when a feature is activated—activation dependancies include exclusion-activation and subordinate-activation.

Yu et al. [2008] indicate that features hide stakeholder intentions. For example, it is not possible to know why a particular feature is in the model. This led number of attempts to link feature models to use cases [Griss et al. 1998, Halmans and Pohl 2003], and mapping features to UML activity and class models [Czarnecki and Antkiewicz 2005]. Classen et al. [2007] criticise feature models for mixing between requirements, domain properties, and specifications. Therefore, Classen et al. [2008] introduces features related to the problem domain—in terms of requirements, domain properties, and specifications.

The main difference between the context approach proposed here and feature modelling is that variations in feature modelling do not represent their sources. Accordingly, based on the relation represented in a feature tree—optional or require and so on—it is not possible to predict the nature of change from its source, for example is it personal or cultural. Even when features are mapped to goals or models, features remain isolated from the real world—as Classen et al. [2007] indicates by linking features to the problem frame approach [Jackson 2001]. Feature model does not identify consequences of variability. What happens for example if a mandatory relation is broken or changed, and how to represent elements that are not realised yet?

For example, Deelstra et al. [2004] report on two core issues drawn from their experience with product families; the first is complexity, the second is implicit properties. Complexity arises from the unimaginable number of possible variations that individuals have to select from. Implicit properties, on the other hand, are unrealised or/and undocumented dependancy points. Therefore, the advantage of recording the source of variation—as realised in the context framework by function, taste-and-passion, and culture—is that analysts do not have to cope with such complexity, as there are more variants than an analyst or a model can list. Analysts, and stakeholders as well, need only to know when they are able to vary, especially in wide variation cases. Implicitness as I argue here, is an unrealised element of the system, that needs to be noted as part of the context of the system. As Deelstra et al. indicate, not identifying implicit properties result in false positives and a larger number of human errors. These implicit properties of a system are not captured by the model. Deelstra et al. also points to the issue of consequence. Software engineers when considering variation choices, do not know all of the consequences during a derivation process. As a result, selection consequences either appear early, or prove to complicate development later.

Goal/Question/Metric Approach (GQM): in Basili et al. [1994], goals are linked to specific measurement techniques through questions. Thus, to verify achieved goals, it is important to know in advance how to measure them. A goal is divided into four elements: purpose, issue, object, and viewpoint. The purpose could be to improve the performance (issue) of data transfer (object) according to customer satisfaction (viewpoint). The question could be 'did the performance improve?'. According to the viewpoint—customer satisfaction—the metric is assigned 'increasing performance by 30%'.

The context approach complements the notion of measurable goals, but differs, in two ways. First, context states recognise the use of measurement through *fit* of the force model—that is, measurable elements recognised in GQM as objective data. Context states differ as it recognises the consequence of failure as another aspect. Both consequence and measurability come hand-in-hand. Second, questions in GQM form a technique to link goals to measurements. The influence of *fit*, for example, serves a similar role. But context states extend questions about goal measurements, to measuring perception as well. The perception model enables analysts to consider the depth of knowledge about a certain influence. It is not enough to identify a goal, verifying the basis of identification is equally important.

On explicitness: the need for explicit information/assumption/rational is not new. Garlan et al. [1995] on architecture mismatch recommend to make architecture (implicit) assumptions explicit. But such implicit assumptions are (typically) not integrated within system models. Lago and van Vliet [2005] recognise the need to document and manage assumptions to cater for unanticipated changes in the environment. Assumptions according to Lago and van Vliet are invariabilities that are unrealised and unaccounted for, which include: technical, organisational, managerial assumptions. These assumptions are the state of affairs that a component operates in, that is its context. But Lago and van Vliet recognise when the assumptions change, new conditions (context) become relevant. But when new conditions arise, even if they are not realised, they are recognised as assumptions, albeit implicit. Therefore, assumptions are either realised facts about the context as explicit and unrealised facts about the context as implicit. But this model fails to distinguish between what is realised as true assumptions but not true anymore, and unrealised assumptions thought to be false but actually true. Furthermore, the model does not capture and distinguish relevant facts from irrelevant facts.

Ven et al. [2006] attempt at documenting knowledge about design rational in software architecture based on three levels: implicit, documented, and formalised knowledge. Implicit knowledge is private knowledge not shared by architects, and

may not be realised at all. Documented knowledge is exemplified in the representation of architecture views. Formalised knowledge is documented knowledge but has clear and concise descriptions of the architecture, captured for example by Architecture Description Languages (ADL). Ven et al. [2006] classification is complimentary to the *knowledge model*. But Ven et al. classification is specifically described for architecture knowledge. For example, the classification excludes knowledge about goals not part of the architecture documentation, such as business goals. It also does not describe the process of recognising new knowledge, how implicit knowledge becomes documented.

Other approaches: Ali et al. [2009] introduces 'goals' as a way of identifying requirements' context using the contextual goal model. The model categorises the goal context into: actors, decomposition (OR and AND), goal activation, means-end, and contribution to soft-goals. The model is based on the concept that goals are the context of the system. This excludes the analyst's view (perception) and the different levels of priority for each goal.

Bübl [2002] uses context properties to describe models. Metadata attached with a model is referred to as 'context'. When a model is assigned a context property, the information in the property describes how and where the model, or an element, is used. Context property may include several fields, Bübl's three general context properties: workflow holds requirement specification added by designers, personal data indicating whether the model holds private information, operational area represents information about where the model is implemented from an organisational perspective. Such added data to the model become part of the context and may provide more information used by developers to clarify where the system is to be implemented, but are typically inconclusive. It is also not clear how descriptions are managed for a model implemented in more than one context. Such choices of context elements to document, or attach to a model, face the question of relevance. How to decide which contextual elements are relevant?

Bruin et al. [2002] introduce Feature-Solution graphs (FS-graph) to capture design decisions that impact quality. An FS-graph is formed by two graphs: a feature graph and solution graph. Features are connected to solutions by edges of two types: selection or a rejection. Depending on the system context edges states are adjusted. But the context of the graph is limited to the internal elements part of the design space, what is regarded as features and solutions. The context approach presented here, is open to any element that is part of the system concern. Furthermore, knowledge about the context is graduated, which is not recognised by the FS-graph approach.

6.3 Summary of contribution

In what follows a summary is introduced of the contributions of the presented research on context and the approach to analyse context to explore how systems vary. The summary includes main contribution of what was presented in Chapter 2 and chapters within Part II.

First contribution: *the synthesis of context*: Chapter 2 introduced a novel view of context as possessing two dimensions—*influence* and *perception*. Early attempts to explain why systems vary have led to identify the influence of context as the main source of software/system variation. Further investigation on the topic have led to identify views from literature that have emphasised the influence of context on system decisions. These view were identified in software engineering, and other disciplines such as anthropology, linguistic, philosophy, and building architecture. Other views from literature have also supported the role of perception. The review of literature concluded that there is a need to arrive at a synthesis of views about context, context as influence and perception. The new view synthesised view should replace the classical view in software engineering, that 'context' as 'setting' system boundaries. Other views within software engineering have shown signs of departing from 'setting the system boundary' view, but without necessarily using the term 'context.' The synthesis of context is supported by five themes identified from the literature. The thesis has attempted to realise each theme within the proposed approach. The identified themes are presented in Table 2.3, and what follows provides a list of themes on context:

1. Relevance is directed by knowledge,
2. Context has influence,
3. Context has states,
4. Context is a set of connections,
5. Context regresses endlessly.

Second contribution: *the Context Dynamics Matrix (CDM)*: Chapter 3 introduced two models of context: the force model for influence, and the knowledge model for perception. Both models are represented in terms of states within the CDM, where a context state becomes an 'influence based on perception.' A state shift occurs if the influence changes or the perception changes. Using the matrix it

is possible to compare context states across different systems independently from system elements. As a result, analysts may use context states to show where elements within the system may vary.

The ideas presented in Chapter 3 follows three of the five themes of context identified from the literature—previously presented in Chapter 2. The knowledge model of perception follows the first theme, that relevance is directed by knowledge. The force model for influence follows the second theme, that context has influence. The CDM follows the third theme, that context has states.

Third contribution: *Mapping context states to DFD*: Chapter 4 presents a way to build the context of the system as a whole in terms of individual context states using the CDM, by assigning context states to DFD. By adding context states, analysts may reflect their own understanding of how the system functions. They may also expand, by identifying context states of processes and data flow, on how the DFD view of the system may or may not vary.

Chapter 4 supports two of the five themes of context identified from the literature. Mapping context states to DFDs by representing the context of data flow and processes/terminators follows the theme that context is a set of connections. The theme that 'context regresses endlessly,' is followed by the unfolding process [Alexander 2002] supported by DFDs. The mapping of context states completes addressing the themes that Chapter 3 did not address. It encompasses, in a way of synthesis, the identified views about context from literature represented by the five themes.

6.4 Limitations of contribution

The research presented by this thesis is limited mainly by the study presented in Chapter 5. The study has shown how the approach to model context provide a series of implications on how the system may vary. But the results and conclusions of the study have the following limitations:

- The PDA study presented in Chapter 5, was conducted on an industrial project within a controlled environment. Although it was conducted on a real set of requirements, a more realistic evaluation of the approach is to perform the analysis as the requirements are interpreted and discussed. Much of the contextual transformations that the context states capture occur during the process of reading and understanding the system through requirements as they are analysed.

- The analysis and modelling of the context of the PDA requirements are derived from scenario-based statements within an event-action table (Appendix B). Each statement was derived from a different format of requirements, which indicates that the requirements have evolved into the present state. Therefore, the study becomes limited to the requirements in its final scenario-based format. As discussed previously in Chapter 2 the scenario-based technique is part of what was identified as the common-sense approach to context. Thus the scenario-based technique may have more contextual detail, not provided by other techniques, that makes the process of modelling context possible. A more comprehensive study of the requirements of the PDA system would require the use of the initial set of requirements used before arriving at the scenario statements.

- The evaluation of the approach involved only one analyst, and limited interaction with external stakeholders. Thus the study demonstrates the use of the approach through a single view over one iteration. While the may be used by analysts individually, the analysis is unlikely to be conducted in isolation. Accordingly, the study shows limited evidence on how the context states improve the communication between system developers or the understanding of the requirements.

- The PDA study demonstrated how context states imply sources of system variation. But the study did not show whether the system in reality behaved in accordance with the implications of context states. In order to collect such evidence, the study must first identify for each case what change had occurred and within which context. But because the context states are identified for the context of systems under development, it was not possible to obtain such information since the system had already been implemented.

6.5 Overall conclusion

The overall conclusion of the research is within the answer to the question posed earlier: what is context? The proposed answer seeks to be pragmatic. Instead of focusing on what 'context' is, it is more useful to show how context could be used. Thus the use of the CDM, extending DFD, and the implications that all of these techniques have on how to vary the system; is only part of what could be gained from the use of the concept. Thus far, what was explored by the implications of modelling context on variation, is only a narrow aspect of what the a system could offer when its context is modelled. The difficulty of how to interpret a system's context states, is to maintain focus on a single view to provide a clear interpretation

of results.

When context states of the PDA system was presented to a member of the development team, it was possible to draw from the map possible implications other than variation. It was surprising, and at times confusing, how to draw two seemingly unrelated conclusions from the same element, without identifying a conflict. For example, under the influence of *fit*, an element's opportunity to vary are limited. But *fit* also implies that an element under its influence may fail because it was not able to comply with the demands of *fit*. It is not clear, however, whether there is a necessary correlation between the implications of variability and failure, with other context states other than what is observed for *fit*. Under the influence of *taste-and-passion* there is the ability to vary because the consequence is assumed not to be severe. But it is possible to view that the result of not being able to vary under *taste-and-passion* is undesirable, at times, as not being able to satisfy the preference that created it in the first place. The trade-off becomes whether to vary to achieve functionality but loose preference, or not achieve functionality and satisfy a preference. Underlies the ability to vary within the influence of *taste-and-passion* and *culutre*, is the assumption that achieving functionality is the main preference. But when under *fit*, functionality may not be achieved without complying with its measures. Measures of fit and functionality become one.

Accordingly, it is possible to observe that a possible connection between variability and failure may be associated with the concept of misfit that Alexander [1964] proposed. Misfits that occur under an influence of *fit*, are misfits that cause the system to fail. But misfits that occur under an influence of *taste-and-passion* may not cause the system to fail per se, but may lead to undesired result, manifested in customer dissatisfaction for example.

The research also challenges the notion that context is used more effectively through common-sense. The common-sense approach, as argued previously, has shown success in software engineering through the use of scenarios. But because context regresses endlessly, it forms a burden on the intellect that analysts cannot use there common sense effectively with multiple context instances. What the use of context states to enrich DFDs have shown, is that it is possible to gain considerable advantages when the analyst's common sense is guided to build the context of a system through a more complex set of context structures.

Ultimately, what this research has attempted to demonstrate, is the effectiveness that might be gained, if analysts achieve a balance between a purely common sense approach to context and the more formal approach to context as boundaries. In the spirit of Scharfstein's (1989) dilemma of context, the balance

between extreme contextualisim and total abstractionism, is achieved through a stepwise process of synthesis between the influence of context and its perception.

6.6 Recommendations for future work

Future work should address the limitation of the research thus far. This is achieved through the support of the application of the context approach by providing further evidence that supports what has been demonstrated so far. Further evidence may be obtained by establishing an engagement with industry, and provide additional examples of how systems behave in accordance with the proposed context model. There is also the need to make the approach more usable, which is achieved by defining contextual patterns, or alternatively called *archetypes*. Further support of the usability of the approach and to gathering evidence is achieved through the development of tool support. It also possible to generalise the application of context states through two means. First, by applying a general approach to map context states, using context-maps Alshaikh and Boughton [2009] for example. Second, to identify context scenarios over the life time of a project cycle, the whole project, or the life of the system as a whole.

6.6.1 Industrial-scale evaluation

An advantage of the context approach is that it does not necessarily need to replace any current system practices on the level of requirements or design, but it can be used in parallel. This should make it easier to engage with a medium to large scale projects without the need to replace any of their followed practices.

The evaluation should be performed by selecting three to five members to participate in a study while they are involved in the same industrial project for one or two months. They should be from different backgrounds and responsibilities within the project, they could be formed by project managers, designers, analysts, or clients. Each team member is asked to identify context states from requirements, and follow the changes in context as each member perceives them. With the support of an analysis tool that captures the process of assigning context states and following their transitions, each member should submit versions of their view of the system through context states on a regular basis.

The aim of the evaluation is to identify correlation between what the context states identify about the system and how the system behaves under each identified context state. What follows lists some of the main research questions that the such a project may answer.

- How does the context states help team members to take system decisions?
- Can the context states identify contextual thresholds, that motivate system developers to change the system and as result change its context?
- Can the context states provide a reliable indicator of how the system may vary within a particular stage of development, such as requirement analysis.
- Can the perception model be a reliable measure of the success or failure of a development project?
- Can the influence model be a reliable measure of the success or failure of a development project?
- Can the study identify shifts in context states within the CDM that follow a predictable pattern?

6.6.2 Provide additional examples

The context approach requires further examples to support the results obtained so far. The PDA example demonstrated how to apply context states through the interpretation of scenarios based on requirements. Two more examples should be provided that vary on terms of mission and scale.

The application of the approach has been carried in areas outside of the software and systems domain. An example is developed by Chemboli et al. [2010] in the area of course design using workflow management, which presents contextually enriched models using context states. The results of such application of the approach outside the software and systems domain, encourages the application of the context states outside software, such as architecture design and product line engineering. In fact, currently there are some attempts to apply the approach to political science in the area of international relations.

These examples indicate how diverse and wide is the interest in modelling context for various aims. But while the focus should be to select examples from the area of software and system domain, it is possible to extend to examples lie within the domain boundary of the discipline, such as software economics.

6.6.3 Reduce the complexity of contextual analysis

While the process of building context enriched DFD representation of software systems is generally simple, especially for small to medium size systems, it might become a tedious process if applied to larger systems. This is largely due to the

complexity of analysing context itself, as a result of the shell problem for example. Therefore, there is a need to simplify the process further. This was first attempted at the beginning of the research by establishing two levels of representation, a simple one that the diagram shows, and a more complex one within analysis, see for example Alshaikh and Boughton [2009]. But because maintaining a link between both levels was believed to be error prone, and involves unnecessary redundancy, it was replaced by the current approach to use DFD instead.

Recently, however, contextual patterns were easily identified within the process of identifying context states that suggests using an auxiliary set of patterns identified as *archetypes*. These set of archetypes are defined and understood context state examples that may apply to more than one system. An example of such an archetype is the context centre 'password' in the PDA system. In this case the influence that 'password' places on users is fit based on theory (Fit:T), and the force placed on 'password' by the system is taste-and-passion based on theory (T&P:T). The context states of 'password' are archetypical, because most systems that use 'password' have the same context state(s). While it is possible to have a password that does not apply the same forces because it is within a different context, it would not be archetypical, however. Thus using the archetypical method, it is possible to reduce the effort of identifying forces if the decision is made that the context follows a particular example.

While the use of archetypes have not yet been tested yet, the use of this method on selected examples have shown some promising results. It is the hope that by the use of this method, that the context of a domain may be identified through a set of archetypes that can be themselves the target of study across systems. Thus offering a new dimension on the research of context analysis.

6.6.4 Tool support

Tool support enhances the study of system context in two ways, by simplifying the process of building DFD and assigning context states, and assist analysts to maintain a record of when and how the context of the system transforms. A tool, C-Map, is currently under development. C-Map may serve partly as a research assistant tool and as a development tool. The tool provides partial automation of the process of analysing the context of software systems based on textual requirements. Thus the aim in future work, is to develop a fully functional version of C-Map, which should support the use of the approach in industry.

For a more complete tool framework, a second version of the tool should provide support for context analysis of architecture and design. The tool should allow the analysis to proceed from requirements to architecture following a stepwise process.

The tool then would allow analysts to obtain a more in depth analysis of the context of the system as the analysis of requirements extends to other software stages, such as architecture.

6.6.5 Expand DFD to a more general representation

In Alshaikh and Boughton [2009] the approach to use context-maps, an approach represent context states, were presented. Context-maps are based on the concept of focal points, previously mentioned in Chapter 4, in the discussion of centres as part of the unfolding process [Alexander 2002]. But because the context-map approach to represent context states has introduced considerable complexity to the process of context analysis, representing context was replaced by the use of DFD.

Similar to DFD, context-maps show data, represent connections between external and internal system elements, but it does not limit description to process and data flow. Furthermore, context-maps show promising results on how it is possible to expand the analysis to other elements that DFD do not typically represent, such as choices between implementation technologies. For example, in the PDA requirements the use of Bluetooth as a way to communicate between PDA and C-PDA during the election. Using DFD, the communication between PDA and C-PDA is represented independently as processes, and the technology used for communication is not represented. But using context-maps, it is possible to represent such elements.

Yet, one source of the complexity that context-maps may bring to the analysis, relates to the order of which analysts represent system views. Part of the confusion that may be added to the complexity of the use of context-maps, is to decide on the order to introduce system views for analysis. Using DFD to represent the context states of the system, compared to the view of context-maps, may be a step within the order of introducing system views. The move may be introduced as a process to transit from the functional view to other system views, such as architecture.

System views, however, should be independent context-states, because some context states of the design views may be identified even while other views are discussed separately. For example, the context of 'password' in the functional/DFD view of the system should not change when the view of architecture/design is considered. The influence of fit that 'password' applies on 'polling official' should remain unchanged. Future work will focus on how to successfully manage an effective order of introducing system elements for analysis. An effective order may be found in generalising the order of unfolding of analysis that DFD follows, to architecture and design.

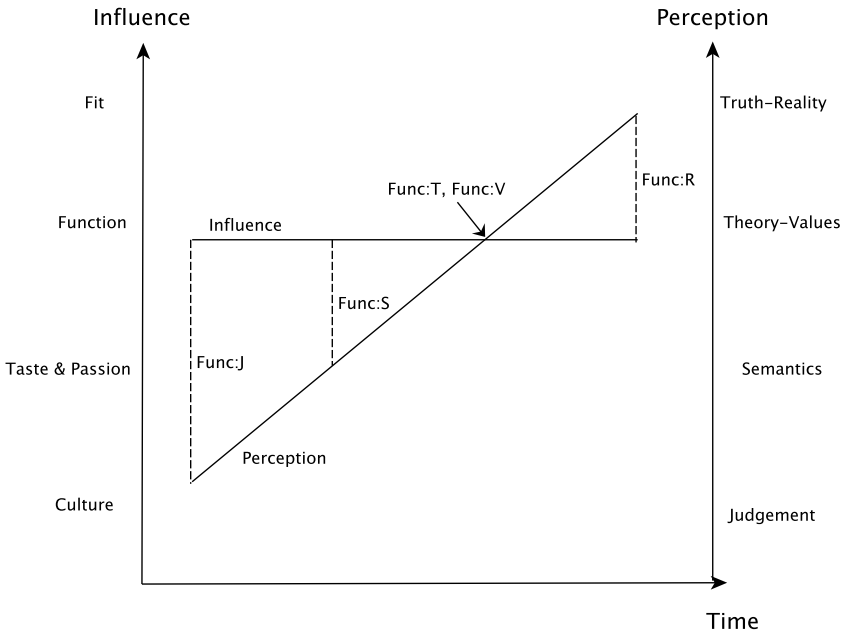


Figure 6.1 – The progression of the context state of telemetry request and execute commands, where influence remains constant and perception increase in a steady progression.

6.6.6 Context state transition scenarios

Context state transition scenarios describe how context states perform multiple transits. A research question that comes as a result of describing context in terms of states, is how context states transit over time, and whether it is possible to identify a predictable series of transitions.

Figure 6.1 shows an ideal scenario considered for the process of software development starting from the analysis of requirements. This scenario assumes that the context state transits within a controlled system process, which reviews requirements and engages with stakeholders continuously in knowledge driven process to understand the system and its requirements. In such process, system drivers and decision makers are always trying to build their system on explicit knowledge and avoid conjecture. By increasing the knowledge of the system, developers strive to increase their knowledge about the system and as a result drive context states to transit to stronger perception levels. Consider the example of 'password' mentioned in PDA requirements (Appendix B).

The scenario that Figure 6.1 presents, includes a graph of the two dimensions of influence and perception on the y-axis, and time on the x-axis. The graph

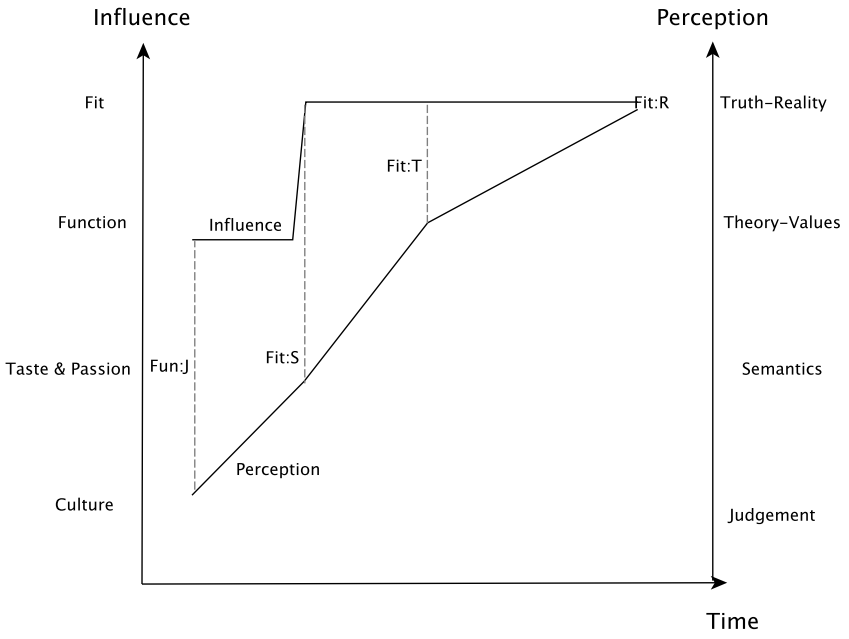


Figure 6.2 – The progression over time of the context state of a system that introduces an authentication mechanism that changes the influence from function to fit.

represents the scenario of the influence of 'polling official' on 'password' recognised as *function* based on *semantics*. On the influence dimension, the graph is flat, it remains unchanged over the time of the project. But on the perception dimension, the the graph shows a continuous rise in perception, starting from judgement at the beginning of analysis until it reaches to *truth-reality* when the system is implemented. What supports the increase in perception is the growing knowledge and confidence about the context of the system. But what are other possible scenarios that may derive both changes in influence and perception?

Consider the scenario represented by Figure 6.2. In this scenario, the context of a system element starts under a force of function, such as the influence on polling official by the PDA system without using an authentication mechanism. But as the system is in the process of writing requirements—notice the increase of perception from *judgement* to *semantics*—a shift in influence occurs from *function* to *fit*. One possible rational for this shift, that the need for an authentication mechanism was not realised at first, but when the requirements were written, the need to allow users access after they provide proper authentication emerged. As a result, the force of *function* applied by PDA is replaced by *fit* on all users.

But is it possible for the authentication scenario to go through another context state shift? For example, it is possible that after the force of function shifts to

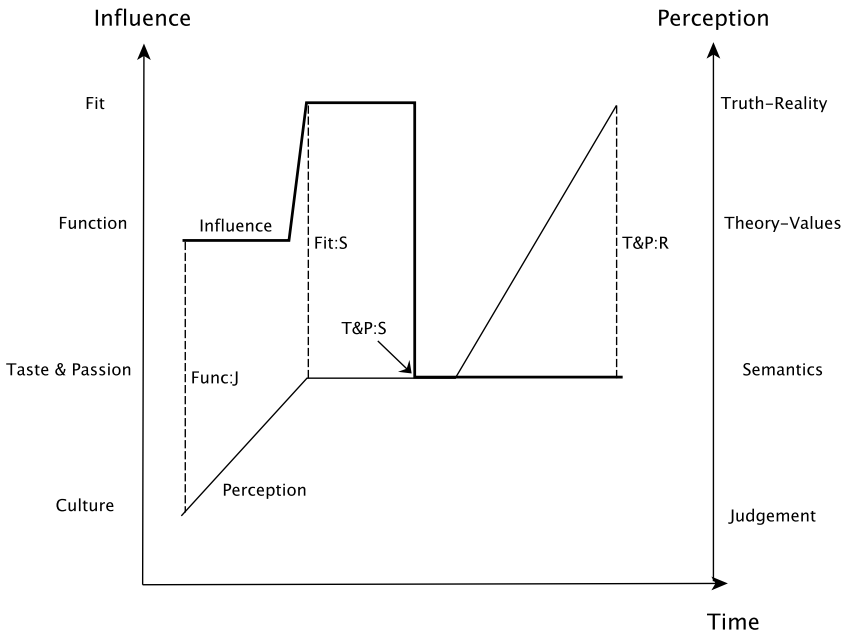


Figure 6.3 – Context scenario showing a series of transitions. Influence is represented by the thick graph transiting from *function*,*fit*, and to *taste-and-passion*. Perception is represented by the thin line transiting from *judgement* to *semantics*, then gradually to *truth-reality*.

fit, stakeholders recognise that they prefer to gain access to the PDA with or without authentication. Users, then, may have different features of the system may be shown to them when they enter without proper authentication, and enhanced features are shown when the authentication is used. As a result, users may be influenced by *taste-and-passion*. Figure 6.3 shows the shift from *fit* to *taste-and-passion* while the perception is based on *semantics*.

Context scenarios may represent a novel approach to describe the dynamics of systems through context states. Further research should aim to explore various context scenarios to try to establish a correlation between system variation and variation of context over time. It is possible also to study the context variation of archetypes, where context states of general examples are explored over the history of their use across multiple systems.

6.7 Closing remarks

Looking at what was achieved during the life of my research project, I expect that the study of software and system's context, along with what remains as serious

research challenges, should provide new insights into the nature of software systems and system development. But the study of context also exposes the need to consider a multidisciplinary approach to the study of context as a complex phenomena.

It is my intention to proceed with this work seeking answers to all the difficult and interesting questions posed by the topic, following the approach presented by this thesis, as long as the approach remains useful, and the questions to be answered remain relevant.

———————*The End*———————

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Part

IV

Appendices

Appendix

A

Proof-of-Concept: *Context States of the Voter Mark-off System*

Contents

A.1 Introduction

What follows are the results of the analysis of the context of the PDA system presented in Chapter 5. *Context states* are presented in tabular format:

Element| <*state*|*state*> |*Element*| *Description*

The table may be mapped to DFDs, whenever applicable, the left element is positioned on the left of the diagram, and the right element is at the right. The *context state* on the left represents includes the influence that the right element places on the left element, and the right *context state* includes the influence that the left element places on the right element. The *description* whenever applicable, describes the influence and the perception. In most cases, where the perception is baed on *semantics*, a reference to the statement from requirements that the influence was perceived under. The reference is code used for the original requirements attached in Appendix B. Some of the influences perceived under *semantics* are obtained from the *verbal* description provided by the system developer.

Context states are presented in the order they appeared in Chapter 5. First, the context-diagram of the manual system. Second, the DFD-0 of the manual system and the after introducing PO and PDA. Finally, the context-diagram of the re-scoped system after adding PO and PDA. *Context states* are presented in three levels. The first level is the context state of the system as a whole. The second level present *context state* of the each DFD without data flow. The third level present *context states* of DFDs with data flow.

A.2 Context states of context-diagram

A.2.1 Context state of the system as a whole

Element	State	Description
Voter Mark-off System	C:J	The developed system is under the influence of the culture of the manual system. The reality of this influence is based on judgement, and may need to be confirmed.

Table A.1 – Context state of the system as a whole

A.2.2 Context states of context-diagram—second level

Table A.2 – Context states of context-diagram—second level

Element	<State	State>	Element	Description
Voter	C:V	Func:S	Voter Mark-off System	Voter demands functionality from the system, but the System will provide its service according to its cultural demands. The functional demands that voters show is described by requirements (M.1), and the demands for voters to provide information to the system is described by the ACTEC as part of its procedures [**?**].
Mark-off List	Func:S	C:J	Voter Mark-off System	Mark-off List may exist before the process of marking off voters commences. Thus it influences the system according to its culture. As a result, the system's demands may only be functional when it adapts to the structure and data defined for the list. Reference for the functional influence is mentioned by requirements (e.g., D.7).
Voter Mark-off System	C:J	Func:J	ACTEC Electoral Roll	The structure of the electoral roll may be defined before the system is developed, which forces it to comply to it. The system, however, only demands functionality from the ACTEC Electoral Roll.

A.2.3 Context states of context-diagram—third level

Table A.3 – Context states of context-diagram—third level.

Element	<State	State>	Element	Description
Voter	—	Func:S	Voter Data	Voter demands from Voter Data to perform the function of marking him/her from the electoral roll based on requirements (M.1).
Voter Data	Fit:T	Func:S	Voter Mark-off System	Voter Data will have access to the system to perform its function, but the system will demand from it to fit to its stored data. The influence of Voter Data is drawn from requirements (e.g. M.3). The demand by the system for the Voter Data to fit its stored data is referenced by multiple statements (M.3–M.9).
Voter	Func:S	Func:S	Feedback	Voter is required to receive feedback from the system, as mentioned by the requirements (M.2). The feedback should inform the Voter with the result of the search.
Feedback	Func:S	—	Voter Mark-off System	Feedback is provided during the process of search. The system demands from the feedback to be informative (functional).

Continued on next page

Section A.2—continued from previous page

Element	<State	State>	Element	Description
Voter	Func:S	Func:S	Request	Voter receives a request that he/she can respond to, such as providing additional data: DoB, address, and middle name. When a match found, the Voter may be asked to provide additional information as a way to confirm his/her identity. Requests are part of the functionality of verifying the identity of Voter, as mentioned by requirements (M.2).
Request	Func:S	—	Voter Mark-off System	The system demands from Request to perform its function to communicate to Voter the needed data, as described by requirements (M.2).
Voter	—	Func:J	Ballot Paper +Instructions	Voter receives 'Ballot Paper+Instructions' from the system after being marked off the list. Voter demands from the 'Ballot Paper+Instructions' to be functional when used to cast his/her vote.
Ballot Paper-Instructions	Func:J	—	Voter Mark-off System	The system may demand from 'Ballot Paper +Instructions' to functional—by checking the ballot paper.
Voter Mark-off System	Func:J	Func:J	Voter Details	Both the system and 'Voter Details' demand functionality from each other when transferred. The details of the this process is not described by requirements.

Continued on next page

Section A.2—continued from previous page

Element	<State	State>	Element	Description
Voter Details	Func:J	—	ACTEC Electoral Roll	'Voter Details' is under the influence of ACTEC Electoral Roll to provide functionality when sent to the system.
Voter Mark-off System	—	Func:S	VoterID+Voted	'Voter Mark-off System' demands from 'VoterID+Voted' to be provide needed functionality to be stored in the 'Mark-off List' according to requirements (D.7).
VoterID+Voted	Fit:J	Func:S	Mark-off List	'Mark-off List' may demand 'VoterID+Voted' to fit to its capacity. But 'VoterID+Voted' demands from 'Mark-off List' to allow it to be stored in the list (see requirements starting from D.7).

A.3 Context states of DFD-0

A.3.1 Context states of DFD-0—second level

Table A.4 – Context states of DFD-0—second level.

Element	<State	State>	Element	Description
Import Electoral Roll	Fit:J	Func:J	Electoral Roll for Mark-off	'Import Electoral Roll' demands from 'Electoral Roll for Mark-off' to perform its function and store the imported data, but 'Electoral Roll for Mark-off' would only accept the data if it fits to its capacity.
Establish Voter Validity/Search Voter	Func:S	Func:S	Electoral Roll for Mark-off	The interaction between 'Establish Voter Validity' and 'Electoral Roll for Mark-off' are based on functional demands to identify search for voter details obtained from the 'ACTEC Electoral Roll.'
Establish Voter Validity/Perform PO Operations	—	Func:J	Issue Ballot Paper & Instructions	'Establish Voter Validity' demands from 'Issue Ballot Paper & Instructions' to produce a ballot when it is notified that a voter has been marked from the electoral roll.
Perform PO Operations	T&P:S	Func:J	Search for Voter	'Perform PO Operations' uses 'Search for Voter' to obtain voter details. It demands functionality when it searches for data. 'Search for Voter' allows 'Perform PO Operations' choices to search for, which leaves the search to be according to preference.

A.3.2 Context states of DFD-0—third level

Table A.5 – Context states of DFD-0—third level.

Element	<State	State>	Element	Description
Voter Data	Fit:T	Func:S	Establish Voter Validity	'Establish Voter Validity' demands from 'Voter Data' to fit to the stored voter name, while 'Voter Data' requires from 'Establish Voter Validity' to provide functionality.
(FSMAD) FirstName + Surname + (Middle-Name) + (Address) + (DoB)	Fit:J	Func:S	Establish Voter Validity	'Establish Voter Validity' demands from 'FSMAD' to fit to the query that requested, while 'FSMAD' requires 'Establish Voter Validity' to allow it access to the process.
Perform PO Operations	—	Func:S	Voter Data	'Perform PO Operations' demands from 'Voter Data' to perform its function to search for stored data.
Voter Data	Fit:T	T&P:J	Search for Voter	'Search for Voter' demands from 'Voter Data' to fit the stored data. 'Voter Data' may require from 'Search for Voter' to preform at a certain preferred performance.
Instances	Func:S	—	Search for Voter	'Search for Voter' demands from 'Instances' to perform its function when it returns results (e.g., M.6.1).

Continued on next page

Section A.3—continued from previous page

Element	<State	State>	Element	Description
Perform PO Operations	Func:S	Func:S	Instances	'Perform PO Operations' demands from 'Instances' to perform its function when it returns results (e.g., M.6.1), while 'Instances' demands from 'Perform PO Operations' to show the results returned by 'Search for Voter.'
Electoral Roll for Mark-off	Func:S	Fit:T	FSMAD	'Electoral Roll for Mark-off' demands from 'FSMAD' to fit to its stored data. While FSMAD demand the functional goal behind sending it, that is, for 'Electoral Roll for Mark-off' to obtain voter details.
Establish Voter Validity	—	Func:J	Voter OK	'Establish Voter Validity' may require 'Voter OK' to perform its function to issue a ballot paper and instructions.
Voter OK	—	Func:J	Issue Ballot Paper & Instructions	'Voter OK' demands from 'Issue Ballot Paper & Instructions' to perform its function of producing a valid ballot paper and correct instructions.
Establish Voter Validity	—	Func:J	Voter OK	'Establish Voter Validity' may require 'Voter OK' to perform its function to issue a ballot paper and instructions.
Voter OK	—	Func:J	Issue Ballot Paper & Instructions	'Voter OK' demands from 'Issue Ballot Paper & Instructions' to perform its function of producing a valid ballot paper and correct instructions.

A.4 Re-scoped context-diagram

A.4.1 Context states of context-diagram—second level

Table A.6 – The context states of the context-diagram—second level.

Element	<State	State>	Element	Description
PO	T&P:S	T&P:J	Verify Voter	'Verify Voter applies a force of taste-and-passion on PO based on semantics, because it allows the PO to search the system using more than one element: 'First Name, 'Address, 'DoB, and so on. Similarly, PO may search for Voter without using the system, by going back to the manual electoral roll.
Mark-off List	Func:S	C:J	Verify Voter	Mark-off List may exist before the process of marking off voters commences. Thus it influences the system according to its culture. As a result, the system's demands may only be functional when it adapts to the structure and data defined for the list. Reference for the functional influence is mentioned by requirements (e.g., D.7).
Verify Voter	C:J	Func:J	ACTEC Electoral Roll	The structure of the electoral roll may be defined before the system is developed, which forces it to comply to it. The system, however, only demands functionality from the ACTEC Electoral Roll.

A.4.2 Context states of re-scoped context-diagram—third level

Table A.7 – Context states of re-scoped context-diagram—third level.

Element	<State	State>	Element	Description
PO	—	Func:S	Voter Data	Voter demands from Voter Data to perform the function of marking him/her from the electoral roll based on requirements (M.1).
Voter Data	Fit:T	T&P:J	Verify Voter	Voter Data will have access to the system to perform its function, but the system will demand from it to fit to its stored data. The influence of Voter Data is drawn from requirements (e.g. M.3). The demand by the system for the Voter Data to fit its stored data is referenced by multiple statements (M.3–M.9).
Verify Voter	Func:J	Func:J	Voter Details	Both the system and 'Voter Details' demand functionality from each other when transferred. The details of the this process is not described by requirements.
Voter Details	Func:J	—	ACTEC Electoral Roll	'Voter Details' is under the influence of ACTEC Electoral Roll to provide functionality when sent to the system.
Verify Voter	—	Func:S	VoterID+Voted	'Verify Voter' demands from 'VoterID+Voted' to be provide needed functionality to be stored in the 'Mark-off List' according to requirements (D.7).

Continued on next page

Section A.4—continued from previous page

Element	<State	State>	Element	Description
VoterID+Voted	Fit:J	Func:S	Mark-off List	'Mark-off List' may demand 'VoterID+Voted' to fit to its capacity. But 'VoterID+Voted' demands from 'Mark-off List' to allow it to be stored in the list (see requirements starting from D.7).

Appendix

B

Proof-of-Concept: *Requirements of the Voter Mark-off System*

Contents

B.1 Introduction

The requirements in the form of an even-action lists of the PDA system is provided here. It is divided into three parts: accessing the PDA, marking an elector (voter) off the electoral roll, reporting, and data transfer. The statements, and the format of the requirements are presented here in their original form that was used to analyse the system using DFD, and obtain knowledge of the context of the system. But as the tables show, each row points to the original requirements that were used to obtain these scenarios. The event-action list tables are republished with a special permission from Software Improvements Pty Ltd.

Accessing the PDA

#	Event	Action	Condition/ Comment	Requirement
A.1	PDA is switched on	System prompts for password.	For any of the non C-PDAs to be used: PDA application has been loaded PDA is switched off	R22, R22.1, R22.2, R24
A.2	Password entered	System checks if password is correct	Password same for all PDAs. Password is of 6-digit form. Password has not previously been entered.	
A.2.1	Password is correct	System displays Home screen containing: Areas to input Family name and Given name Access to reports (statistics) Access to upload (in case auto send fails) Special alpha keyboard Find button	Functions only available when correct password entered. Upload = send Download = receive	R1, R26.1, R26.4
A.2.2	Password is incorrect	System displays error message: *Password incorrect*		
A.2.2.1	An incorrect password is entered (for the 10^{th} time)	System deletes database and turns off PDA	There have been 9 attempts to enter the correct password.	R25

A.3	C-PDA turned on	System prompts for password	For any of the C-PDAs to be used: C-PDA application has been loaded C-PDA is switched off	
A.3.1	Password entered	System checks if password is correct	Password same for all C-PDAs and PDAs Password has not previously been entered C-PDA has permanent power supply	
A.3.2	Password entered is correct	C-PDA system activates Bluetooth and maintains active status for <specified period> per election day		
A.3.3	Password entered is incorrect	System displays error message: *Password incorrect*		
A.3.2.1	An incorrect password is entered (for the 10^{th} time)	System deletes database and turns off C-PDA	There have been 9 attempts to enter the correct password	R25
A.4	PO provides no input	System continues to display current screen	PDA or C-PDA has not been used for less than 3 minutes	
A.5	PO provides no input	System goes to 'sleep'.	PDA or C-PDA has not been used for 3 minutes	R8
A.6	PO wakes up PDA or C-PDA	System displays screen that was displayed when system went to sleep	PDA or C-PDA is 'asleep' for less than 30 minutes Standard operation to be implemented	R8.1, R23

A.7	PO turns on PDA or C-PDA	System prompts for password See A2.1 or A3.1	PDA or C-PDA has not been used for 30 minutes	R9, R9.1, R24

Marking an elector off the electoral roll

#	Event	Action	Condition/ Comment	Requirement
M.1	Elector approaches Polling Official (PO)	No change to display	System has Home screen displayed including special Alpha-keypad (simplified keyboard with hyphen (-) and apostrophe (‘) and large keys.	R1, R5.1, R7, R7.1
M.2	PO asks elector for name	No change to display		
M.3	PO enters both Family name and Given names in full, and then selects Find	System searches on Family name and then Given name.	Stylus is to be used for all inputs.	R5, R10, R10.1, R11
M.3.1	Only one match found (full Family name and Given names)	System displays elector screen with background colour for elector’s electorate and with elector details: Family name Given names (1st in full with 1st two letter of 2nd Given name) Full address details Electorate ‘Voted’ status	All bar ‘voted’ status are Read Only.	R17, R18, R18.1, R18.2, R18.3, R18.5, R18.6, R19

M.3.2	More than one match found (full Family name and Given names)	System displays entered Family name (in full) with 1st Given name (in full) and 1st two letters of 2nd Given name and address of all matching electors. Elector details to fit on one line only. Any elector already marked as 'voted' is to be coloured differently in the list. If list exceeds on-screen space for display, System displays means of accessing remainder of list.	Means of accessing full list to be scroll bar or large arrows (if PALM functionality permits).	R13, R14, R14.1, R15, R15.1, R15.2, R15.3, R15.4
M.3.3	Match only with Family name	System displays Family name with list of all Given names with that Family name and elector address. Elector details to fit on one line only. Any elector already marked as 'voted' is to be coloured differently in the list. If list exceeds on-screen space for display, System displays means of accessing remainder of list.	Means of accessing full list to be scroll bar or large arrows (if PALM functionality permits).	R13, R14, R14.1, R15, R15.1, R15.2, R15.3, R15.4

M.3.4	No match	System displays message 'No match found' with OK button to return to Home screen		
M.4	PO enters part Family name and Given names, either in full or in part, and then selects Find	System searches on part Family name and then Given name.		R10, R10.1, R11, R12, R12.1, R13, R13.1
M.4.1	One match found	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given names (1^{st} in full with 1^{st} two letter of 2^{nd} Given name) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only	R17, R18, R18.1, R18.2, R18.3, R18.5, R18.6, R19

M.4.2	More than one match found	System displays list of matches in alphabetical order of Family name and within each Family name Given name in alphabetical order, and with elector address. Elector details to fit on one line only. Any elector already marked as 'voted' is to be coloured differently in the list. If list exceeds on-screen space for display, System displays means of accessing remainder of list.	Means of accessing full list to be scroll bar or large arrows (if PALM functionality permits).	R13, R14, R14.1, R15, R15.1, R15.2, R15.3, R15.4
M.4.2.1	PO selects one elector from list	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given name (if any) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only	R16, R19
M.4.3	No match	System displays message 'No match found' with OK button to return to Home screen		

M.5	PO enters only Family name in full and selects Find	System searches on full Family name.		R10, R10.1, R11
M.5.1	One match found	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given name (if any) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only	R17, R18, R18.1, R18.2, R18.3, R18.5, R18.6, R19
M.5.2	More than one match found	System displays list of matches in alphabetical order of Given name with address. Elector details to fit on one line only. Any elector already marked as 'voted' is to be coloured differently in the list. If list exceeds on-screen space for display, System displays means of accessing remainder of list.	Means of accessing full list to be scroll bar or large arrows (if PALM functionality permits).	R14, R14.1, R15, R15.1, R15.2, R15.3, R15.4

M.5.2.1	PO selects one elector from list	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given name (if any) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only	R16, R19
M.5.3	No match	System displays message 'No match found' with OK button to return to Home screen		
M.6	PO enters only Family name in part and selects Find	System searches on part Family name.		R10, R11, R12
M.6.1	One match found	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given names (if any) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only.	R17, R18, R18.1, R18.2, R18.3, R18.5, R18.6, R19

M.6.2	More than one match found	System displays list of matches in alphabetical order of Family name and within each Family name, Given names in alphabetical order. Elector details to fit on one line only. Any elector already marked as 'voted' is to be coloured differently in the list. If list exceeds on-screen space for display, System displays means of accessing remainder of list.		R14, R15, R15.1, R15.2, R15.3, R15.4
M.6.2.1	PO selects one elector from list	System displays elector screen with background colour for elector's electorate and with elector details: Family name Given name (if any) Full address details Electorate 'Voted' status	All bar 'voted' status are Read Only	R16, R19
M.6.3	No match	System displays message 'No match found' with OK button to return to Home screen		

M.7	PO enters Given name in part or full and selects Find	System displays message: "Please enter at least part of a Family name" with OK button to return to Home screen.	Note: Family name not entered.	R10.2, R11
M.8	System displays list of electors	No change. System waits for PO to select one elector.		
M.9	Back button selected on Elector screen	System returns to last display of listed electors.		
M.10	Elector is marked as 'voted'	System adds 'tick' to editable 'voted' tick box, adds elector to 'voted' table in database, and adds date and time stamp to database entry. System also displays message: *Issue elector a <electorate name> ballot paper* with an OK button. System starts process to transmit elector details as 'voted' to C-PDA.	Details of elector are displayed.	R18.6, R18.6.1, R18.6.1.1, R18.6.1.2, R18.6.1.2.1 R18.6.1.3 R18.6.1.4
M.10.1	OK button is not selected.	No change. System waits for PO to select OK.		
M.11	OK button is selected	System returns to Home screen ready for input of Family name and Given names.		R18.6.1.5

M.12	'Voted' status of elector is changed to 'not voted'.	System displays confirmation message: *You are about to change this elector's voting status to having not voted. Do you wish to proceed?* with Yes and No buttons	Details of correct elector are displayed with voter marked as 'voted',	R18.6.2, R18.6.2.1
M.12.1	Neither Yes nor No button is selected.	No change. System waits for selection of Yes or No.		
M.12.2	Yes is selected to confirmation message (ie wants to change status to 'not voted')	System removes 'tick' from display, marks elector as 'not voted' and records with date/time stamp of deselection. System starts process to transmit change of status to C-PDA. Home screen is displayed.	Confirmation message: *You are about to change this elector's voting status to having not voted. Do you wish to proceed?* with Yes and No buttons is displayed Note: Trevor's suggestion is that we use an 'audit trail' table to handle any number of voted/not voted selections for any one elector. This table need only contain those electors that have been deselected at least once.	R18.6.2.2, R18.6.2.2.1 R18.6.2.2.2, only valid if audit trail maintained R18.6.2.2.3
M.12.3	No is selected to confirmation message (ie does not want to change status from 'voted')	Elector screen is displayed with elector marked as 'voted'.	Confirmation message: *You are about to change this elector's voting status to having not voted. Do you wish to proceed?* with Yes and No buttons displayed	R18.6.2.3, R18.6.2.3.1, R18.6.2.3.2

M.13	Home button is selected	System returns to Home screen ready for input of Family name and Given names. System leaves status of 'voted' flag as displayed on elector screen.	Elector screen is displayed without any messages	R2
M.14	Home button is selected	System returns to Home screen with search fields blank.	Search results screen is displayed.	R2
M.15	Home button is selected	System returns to the Home screen with search fields blank	Stats screen is displayed.	R2
M.16	Using 'Graffiti pad' PO attempts to enter something on any screen.	No change to display	Graffiti pad is inoperable from all screens.	R6
M.17	Find is selected without entering any name information.	System displays message: "Please enter at least part of a Family name" with OK button to return to Home screen.	Home screen is displayed	

Reporting

	On PDA			
R.1	Stats function is selected	System displays Reporting screen with number of electors with 'voted' flag based on default settings (ie for Today) for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors today: <#>	PDA is switched on with Home screen displayed.	R4, R27.1.1, R27.2.1, R27.3.1, R27.4, R27.5, R27.5.1, R27.5.2, R27.5.3, R27.5.4
R.1.1	Home button is selected	System displays Home screen	Reporting screen for 'Today' is displayed, including 'change period' option	
R.2	Function to change reporting period is selected.	System displays Reporting screen for Change Period with the following displayed: Start date End date Start time End time and OK button	Reporting screen for 'Today' is displayed, including 'change period' option PalmOS has standard date and time selection dialogs. Entering dates and times as a text field is not encouraged in the Palm philosophy	R27, R27.1, R27.2, R27.3, R27.4

R.2.1	Function for 'Today' is selected.	System displays Reporting screen with number of electors with 'voted' flag based on default settings (ie for Today) for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors today: <#>	Reporting screen for 'Change Period' is displayed with option to select the Today period	R27.5, R27.5.1, R27.5.2, R27.5.3, R27.5.4
R.2.2	Home button is selected.	System displays Home screen	Reporting screen for 'Change Period' is displayed with option to select the Today period	
R.3	Start date is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: Start time: End time: Number of 'voted' electors for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total for period: <#> and OK button	Reporting screen for 'Change Period' is displayed with option to select the Today period. The following are editable: Start date: End date: Start time: End time: Note: Use PDA 'calendar' to select date. If entered direct, use standard format of day/ month/year	

R.3.1	End date is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: End time: Number of 'voted' electors for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total for period: <#> and OK button	See R.3	

R.3.2	Start time is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: <entered time> End time: Number of 'voted' electors for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total for period: <#> and OK button	Time is specified in hours:minutes with am/pm	

R.3.3	End time is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: <entered time> End time: <entered time> Number of 'voted' electors for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total for period: <#> and OK button	See R.3 and R. 3.2	
R.3.4	OK button is selected	System Change Period reporting screen is displayed with: Start date: <entered date> End date: <entered date> Start time: <entered time> End time: <entered time> Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors: <#>		

	On C-PDA			
R.4	C-PDA is switched on.	C-PDA System displays Reporting screen with number of electors with 'voted' flag based on default settings (ie for Today) for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors today: <#> Also available is function to change reporting period	C-PDA is switched on and C-PDA Home screen is displayed C-PDA Home screen is stats screen with default settings ie for Today. Display stats in real time if does not require any more effort.	
R.5	Function to change reporting period is selected	C-PDA System displays Reporting screen for Change Period with the following displayed: Start date End date Start time End time and OK button	Reporting screen for 'Today' is displayed	

R.5.1	Function for Today is selected	C-PDA System displays Reporting screen with number of electors with 'voted' flag based on default settings (ie for Today) for: Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors today: <#>:	Reporting screen for 'Change Period' is displayed	
R.5.2	Home button is selected	System displays Home screen	Reporting screen for 'Change Period' is displayed	
R.6	Start date is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date Start time End time and OK button	Reporting screen for 'Change Period' is displayed See R.3 for format of date, and R.3.2 for format of times	
R.6.1	End date is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time End time and OK button	Reporting screen for 'Change Period' is displayed See R.3 for format of date, and R.3.2 for format of times	

R.6.2	Start time is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: <entered time> End time and OK button	Reporting screen for 'Change Period' is displayed See R.3 for format of date, and R.3.2 for format of times	
R.6.3	End time is entered	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: <entered time> End time: <entered time> and OK button	Reporting screen for 'Change Period' is displayed See R.3 for format of date, and R.3.2 for format of times	

R.6.4	OK button is selected	System Change Period Reporting screen with the following displayed: Start date: <entered date> End date: <entered date> Start time: <entered time> End time: <entered time> Electorate A: <#> Electorate B: <#> …….. Electorate N: <#> Total 'voted' electors: <#>		

Data transfer

	PDA data transfer operations - During election			
	Transmitting change of status to 'voted'			
D.1	System adds tick to 'voted' box on elector screen.	System activates Bluetooth connection to C-PDA. System displays message: *Connecting.*	Elector screen is displayed. PO has marked elector as 'voted'.	R29
D.2.1	System receives confirmation from C-PDA that Bluetooth connection is operational	System changes display message to: *Transmitting.* For elector just marked as 'voted' on PDA, copy to C-PDA: PDA ID, Elector ID, 'voted' flag and date/time stamp when marked as 'voted'. System checks if there are any transactions stored in 'unsent' table and, if any, for each transaction transmits: PDA ID, Elector ID, 'voted' flag status and date/time stamp when marked as 'voted'.	PDA is attempting to connect to C-PDA	R29, R32, R33, R33.1

D.2.1.1	C-PDA receives transmission from (trusted) PDA	For each Elector ID received, PDA ID, Elector ID, 'voted' flag and date time stamp are added to C-PDA table of elector IDs, if Elector ID/PDA ID combination is not already in C-PDA database. If Elector ID/ PDA ID already exist in C-PDA table, update date/time stamp when voted, change flag to 'voted' and leave deselection time unchanged.	PDA has connection with C-PDA Note: Elector ID can appear more than once in C-PDA table: initially marked as 'voted' when deselected, ie changed to not voted when marked as voted again.	
D.2.2	No message received from C-PDA to confirm connection is in place.	System continues to attempt to connect. System continues to display *Connecting*.	PDA with *Connecting* displayed has been attempting to connect to C-PDA for less than 30 secs.	
D.2.3	No message received from C-PDA to confirm connection is in place.	System stops attempting to connect to C-PDA and stores transaction data in table of 'unsent' transactions. PDA displays Home screen.	PDA with *Connecting* displayed has been attempting to connect to C-PDA for 30 secs.	
D.3.1	PDA receives confirmation that data transferred has been received by C-PDA.	PDA system stops displaying *Transmitting* message. Data in 'unsent' table is deleted. Home screen is displayed.	PDA has connection with C-PDA	

D.3.2	No message received to confirm transferred data has been received.	PDA system stops displaying *Transmitting* message. Current transaction is added to 'unsent' table. Home screen is displayed	PDA has connection with C-PDA, *Transmitting* (standard PalmOS feature) is displayed. All data in 'unsent' table and current transaction has been sent.	
	Transmitting change of status from 'voted' to 'not voted (ie elector ID incorrectly marked as 'voted')			
D.4	Yes is selected (to request to confirm change 'voted' status)	System activates Bluetooth connection to C-PDA. System displays message: *Connecting.*	Elector screen is displayed with system prompting to confirm elector status is to be changed to 'not voted'. Yes and No options are available	

D.5.1	PDA receives confirmation from C-PDA that Bluetooth connection is operational	System displays *Transmitting*. For elector just marked as 'not voted' PDA copies to C-PDA: PDA ID, Elector ID, 'not voted' flag and date/ time stamp of deselection. C-PDA checks for Elector ID and PDA ID and if a match found changes flag to 'not voted', leaves voted date/time stamp unchanged and adds into table date/time stamp if deselection.	PDA is connected to C-PDA. Note: at this time there will now be two entries for the same elector ID in the C-PDA database table.	
D.5.2.1	No message received from C-PDA to confirm connection is in place.	System continues to display *Connecting.*	PDA has been attempting to connect to C-PDA for less than 30 seconds	
D.5.2.2	No message received from C-PDA to confirm connection is in place.	System stops attempting to connect to C-PDA and stores transaction data in table of 'unsent' transactions. System displays Home screen	PDA has been attempting to connect to C-PDA for 30 seconds	
D.6.1	PDA receives confirmation that data transfer has been received by C-PDA.	PDA stops displaying *Transmitting* message and displays Home screen.		

D.6.2	No message received to confirm transferred data has been received.	PDA system stops displaying *Transmitting* message. Current transaction is added to 'unsent' table. Home screen is displayed.	PDA has connection with C-PDA. Current transaction to change 'voted' status has been sent.	
	PDA data transfer to C-PDA – After election			
D.7	Function to transfer all data is selected on PDA.	PDA activates Bluetooth connection to C-PDA and displays message: *Connecting.*	C-PDA Bluetooth function is ready to receive.	R30
D.7.1	Function to transfer all data is selected on PDA.	PDA activates Bluetooth connection to C-PDA and displays message: *C-PDA not receiving.*	C-PDA Bluetooth function is not ready to receive	
D.7.1.1	Home button is selected	Display Home screen on PDA	Message: *C-PDA not receiving* is displayed.	
D.7.1.2	Stats function is selected	Display Reporting screen on PDA	Message: *C-PDA not receiving* is displayed.	
D.7.1.3	Search function is selected	Display Home screen on PDA	Message: *C-PDA not receiving* is displayed.	

D.8	PDA receives confirmation from C-PDA that Bluetooth connection is operational	Change display message to: *Transmitting. This may take some time.* For each elector marked as 'voted' on the PDA, the System copies to C-PDA: PDA ID, Elector ID, 'voted' flag and date/time stamp when marked as 'voted' together with any deselection date/time.	PDA has connection to C-PDA.	R30, R32, R33, R33.1
D.8.1	C-PDA detects no difference in data for all Elector IDs transferred, that is for each data set transferred there is a match with existing data in C-PDA database.	C-PDA sends message to PDA indicating data received PDA displays message: *Data matches* and OK button.	For each Elector ID transferred, C-PDA has checked PDA-ID, 'voted' status and date/time stamp (s) in C-PDA database.	
D.8.1.1	OK is selected.	Display Home screen.		

D.8.2	C-PDA detects difference for one or more Elector IDs, that is Elector ID may not be in C-PDA table, or data set for a particular Elector ID is not in C-PDA table.	For each Elector ID not in C-PDA table, details are added to C-PDA table. For each Elector ID in C-PDA table with different PDA ID, add another entry to C-PDA table for that elector ID. For each Elector ID in C-PDA table with matching PDA ID update C-PDA entries for voted flag, date/time stamp (s) of vote and/ or deselected.	For each Elector ID transferred, C-PDA checks PDA-ID, 'voted' status and date/time stamp in C-PDA table.	R32
D.8.2.1	PDA receives confirmation that data transfer has been received by C-PDA.	PDA displays message: *Data updated* and OK button.	C-PDA has been updated with data transferred from PDA.	
D.8.2.2	OK is selected	Display Home screen		
	Central PDA data transfer operations – at end of election day			

D.9	C-PDA is placed in cradle (for downloading to PC)	Wait for cradle to be activated	PALM cradle with USB connection to PC with appropriate software to enable synching with C-PDAs. PC has CD burner installed. Both cradle and PC are switched on and ready to send/receive and synchronise C-PDA electoral roll database with PDA's database.	
D.10	Synchronisation button on cradle is selected.	Cradle turns on C-PDA, if not already on, and communicates with PC to download and synchronise databases. C-PDA and PC both display progress of download (standard function). PC checks C-PDA is a trusted C-PDA and data has not been previously downloaded.	C-PDA is a trusted C-PDA not previously downloaded. Note: Because data for an elector comprises an elector ID, PDA ID and 'voted' flag, an elector who has been marked as 'voted' on different PDAs will be considered by the system as two different electors. Data is to remain on C-PDA.	R32, R33

D.10.1	Download is complete	C-PDA indicates both on-screen and with audio signal download is complete. PC indicates on-screen download is complete.		
D.10.2	C-PDA is removed from cradle	PC displays list of C-PDAs and status of download for each.		
D.11	Synchronisation button on cradle is selected.	PC checks if C-PDA is a trusted C-PDA and displays message: *C-PDA is not a recognised device.*	C-PDA is NOT a trusted C-PDA.	
D.12	C-PDA is placed in cradle and synchronisation button activated.	PC checks if C-PDA is a trusted C-PDA and whether data has been downloaded. PC displays message: *C-PDA has already been downloaded.* *Proceeding with the download will overwrite existing data on the PC.* with *Yes* and *No* buttons.	Data from C-PDA has already been downloaded.	
D.12.1	No is selected	Download process is aborted. PC displays list of C-PDAs and their status.	Data from C-PDA has already been downloaded	

D.12.2	Yes is selected	Download process continues. Both C-PDA and PC display progress of download (standard function).		
D.13	C-PDA is removed from cradle	Both C-PDA and PC return to status of waiting for activation.		
D.14	'Export' option on PC is selected	PC creates csv file of electoral roll database on PC: for each entry in PC database, Elector ID, PDA ID, Voted flag and date/ time stamped as voted is provided in csv format there can be multiple entries for the same Elector ID PC displays message: *Insert write once CD*	ACT Electoral Roll application is active on PC with 'Main' screen displayed.	R31
D.14.1	WORM CD is inserted	PC burns csv file to CD and ejects CD		
D.14.2	CD ejected	PC displays Main screen of ACT Electoral Roll application.		

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Abstract

Context is often considered as a source for system change and variation. But the term 'context' has been typically used to mean the act of setting boundaries and setting system scope in software engineering. In this thesis, I challenge this view by suggesting that context should be applied to imply system variation on all levels of software (system) development. It constitutes as a more complex phenomena of how the system interacts with the world. The suggested alternative approach synthesises context in terms of influence and perception through *context states*.

Context states are represented by a sixteen context state matrix, I refer to as *The Context Dynamics Matrix (CDM)*. Context states are the result of two dimensions of context, perception on the x-axis, and influence on the y-axis. Analysts may identify context of a system using the CDM when they identify the influence that an element exerts and assign their perception of how they identified the influence. Both the influence and perception dimensions are modelled using two models. First, the force model of influence, which identifies four levels of influence that an element may apply, each level shows a different implication on variation. Second, the knowledge model for perception, which shows five sources of knowledge about the influence/context. Accordingly, an analyst may describe the context of a system by matching the level of influence with the level of perception to obtain the context state of a given system element. A context state may imply a high or low level of variability, and a high or low level of perception. The use of context states is independent from any modelling view of a system that either describes functionality or system structure.

Because *context states* describe the context of a system independently from the level/view in which they are described, it is possible to map the context states to enrich the description of a given view. Accordingly, I show how to map context states to functional description of systems by assigning context states to Data Flow Diagrams (DFD). Processes and data flow are assigned context states that enrich their description of the system, in terms of levels of variation that the context may imply.

A proof-of-concept is provided to demonstrate how to apply *context states* to the analysis of the requirements of a system from industry. The results of the study show the viability of using context states to describe the context of system, and support the argument to experiment further to evaluate the effectiveness of *context states* in areas of system development not covered by my research.

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Contents

List of Figures

List of Tables

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Notes on the Synthesis of Context

A novel approach to model context in software engineering

THE AUSTRALIAN NATIONAL UNIVERSITY

A thesis submitted for the degree of
Doctor of Philosophy
of
The Australian National University

Ziyad A. Alshaikh

Feb 2011

This document was produced using TEX , LATEX and BIBTEX

I declare that the work in this thesis is entirely my own and that to the best of my knowledge it does not contain any materials previously published or written by another person except where otherwise indicated.

Ziyad A. Alshaikh
25 Feb 2011

Acknowledgements

I wish to thank my supervisor, Clive Boughton, for his continuous support and guidance throughout the long process of producing my thesis. He kept a genuine interest in the topic that I have chosen for my research, even when the work appeared to be esoteric at times. My thanks also goes to Elisa Baniassad, my supervisor and chair of panel, for her advice and constructive feedback on the presentation of my work. Although she had joined my panel in the last year of my candidature, she gave me valuable advice on methodology and style, which helped to improve the quality of my work.

Since the start of my candidature I had the privilege to engage with many people, in valuable discussions about my topic and other related areas of research. Shayne Flint, as a member of my supervisory panel, had helped me shape my thinking about software modelling and system thinking. His work and our discussions during my early years of candidature, have been a source knowledge and inspiration. Len Bass for his constructive feedback, who I met in early 2009, during my visit to the National Information and Communications Australia (NICTA) in Sydney. Stephen Mellor, who on few occasions, provided me with valuable comments on specific aspects of my work. Richard Gabriel who provided me with valuable feedback on my application of the work of Christopher Alexander to my research. Thanks also goes to all the anonymous reviewers who provided me with valuable feedback on parts of my thesis that I submitted for publication.

Thanks goes to my fellow grad students, Agung Fatwanto, Normi Abu Bakar, Zoe Brain, Alvin Teh, and Luke Nguyen-Hoan; who have shown true friendship and support. Hassan Almari for the interesting discussions on software engineering and almost everything else. Special thanks goes also to my office mate and friend Srinivas Chemboli. He has always been a strong believer in my work, and contributed to the development of my approach by providing new areas of application that I did not explore.

I wish to acknowledge my sponsor King Abdulaziz City for Science and Technology (KACST) for allowing me to pursue my interest in research. Special thanks to Prince Turki Al Saud, Vice President for Research Institutes, who supported my scholarship at the time when he was the Director of the Space Research Institute, and for his continuous support since. Mohammed Almajed, Directer of the National Satellite Technology Program, for his friendship and support. Thanks also goes to the Saudi Arabian Embassy and Cultural Mission, for providing logistical support for my scholarship. Ambassador Hassan Nazer and Cultural Attache Ali Albishri, have always shown support both personally and

through their team. Fahad Alotaibi and Abdulaziz Bin Taleb, from the Cultural Mission, both have been good friends, and have administered my scholarship with the help of their team, to provide me with academic advice throughout the process of completing my thesis.

Finally, I like to thank my family. My parents who inspired me, and provided invaluable advice when I needed. My son Abdualziz, too young to fully understand why his daddy spends most of his time busy with work, spending long hours at the office. I'm sure the day will come when he will realise that all I have done was because I love him. Sharing all of this with me was my wife, Wala'a. She has been always there for me when I needed her, she ensured that I have the time and freedom to complete my dissertation, doing whatever she could to help me meet my deadlines. Without her I would not have been able to make it. My love to you all.

Abstract

Context is often considered as a source for system change and variation. But the term 'context' has been typically used to mean the act of setting boundaries and setting system scope in software engineering. In this thesis, I challenge this view by suggesting that context should be applied to imply system variation on all levels of software (system) development. It constitutes as a more complex phenomena of how the system interacts with the world. The suggested alternative approach synthesises context in terms of influence and perception through *context states*.

Context states are represented by a sixteen context state matrix, I refer to as *The Context Dynamics Matrix (CDM)*. Context states are the result of two dimensions of context, perception on the x-axis, and influence on the y-axis. Analysts may identify context of a system using the CDM when they identify the influence that an element exerts and assign their perception of how they identified the influence. Both the influence and perception dimensions are modelled using two models. First, the force model of influence, which identifies four levels of influence that an element may apply, each level shows a different implication on variation. Second, the knowledge model for perception, which shows five sources of knowledge about the influence/context. Accordingly, an analyst may describe the context of a system by matching the level of influence with the level of perception to obtain the context state of a given system element. A context state may imply a high or low level of variability, and a high or low level of perception. The use of context states is independent from any modelling view of a system that either describes functionality or system structure.

Because *context states* describe the context of a system independently from the level/view in which they are described, it is possible to map the context states to enrich the description of a given view. Accordingly, I show how to map context states to functional description of systems by assigning context states to Data Flow Diagrams (DFD). Processes and data flow are assigned context states that enrich their description of the system, in terms of levels of variation that the context may imply.

A proof-of-concept is provided to demonstrate how to apply *context states* to the analysis of the requirements of a system from industry. The results of the study show the viability of using context states to describe the context of system, and support the argument to experiment further to evaluate the effectiveness of *context states* in areas of system development not covered by my research.

Contents

List of Figures

List of Tables

Part

I

Introduction

Chapter 1

Overview

> [N]eglect of context is the greatest single disaster which philosophic thinking can incur.
>
> John Dewey (1931)

Contents

1.1 Introduction

This dissertation is submitted for the degree of Doctor of Philosophy of the Australian National University. It describes exploratory research on the fundamentals of software/system variation, its sources and implications, on the level of requirements. The result of the research is a novel approach to analyse and represent *context* in software engineering, with application to systems in general.

As an introduction, I provide a brief overview of main parts of the research, supported by guiding information to navigate through the thesis. I conclude with a list of work published during the project and a summary of contributions made to engineering research and practice.

1.2 Motivation and research aim

The initial motivation for conducting this research is to explain sources of software/system variety across different organisational goals, different disciplinary approaches, cultural choices, and within personal preferences. In the spirit of the dichotomy set by Simon [1996], as engineers, we are not primarily interested in the knowledge of how the world works, we are more concerned with the knowledge of making a working world. Therefore, we have to enquire about the nature of the world of our artefacts. One challenging goal, in software development, is to maintain a complete account of system requirements. For example, how to avoid system failures as a result of lack of knowledge? How to take advantage of system opportunities, manifested in customers' desires and aspirations? These concerns are summarised by Glass's Law:

> *Requirements deficiencies are the prime cause of project failures.*
>
> [Endres and Rombach 2003, Law L1, pp16-17]

The research is also motivated by the growing interest in product line architectures. In which the aim is to manage a variety of needs within a family of products, while maintaining economical value and achieving sustainable growth. My preliminary research reported in Chapter 2 shows that the source of sustainable functionality of a successful system, lies in its ability to respond to its context. This is summarised by Conway's Law:

> *A system reflects the organisational structure that built it.*
>
> [Endres and Rombach 2003, Law L16, pp81-82].

Further research reported in Chapter 2 shows that *context* as a concept, has not received enough theoretical distillation in software engineering. This is exemplified in the different views by software practitioners about the use and meaning of context on different levels of software development—requirements, architecture, and design. After reviewing views and theories from the literature dealing exclusively with the concept of *context*—literature from various fields of knowledge: architecture and urban design, artificial intelligence, anthropology, linguistics, philosophy—the need emerged for a synthesis of *context* as a separate system concern in software analysis.

This preliminary work has resulted in the following research aim:

> *'to present a model of context that shows when to vary and when not to vary a system. Such a model should indicate the opportunity to vary the system when the context reflects soft demands, and indicate when it is not possible to vary the system because the context has strict demands. The model should also indicate when strict demands are based on conjecture, and when soft demands are based on strong evidence. The model should show different degrees of variation that the system may have through context.'*

1.3 Thesis scope

The scope of this thesis is the development, representation, and demonstration of the theoretical framework of *context* within software engineering requirements. While the broader applicability of the contextual framework to other areas of software development is discussed, the evaluation of the approach within these areas is beyond the scope of this thesis.

The research on context presented here, have general application to human-based systems. Some reference is made to the implication of the work on human-based systems, but demonstrating or evaluating such implications are beyond the scope of this research.

1.4 Thesis structure

Figure 1.1 shows the organisation structure of the thesis depicted in a Unified Modelling Language (UML) activity diagram [Mellor and Balcer 2002]. The thesis is organised in three parts, represented in the activity diagram with vertical lines,

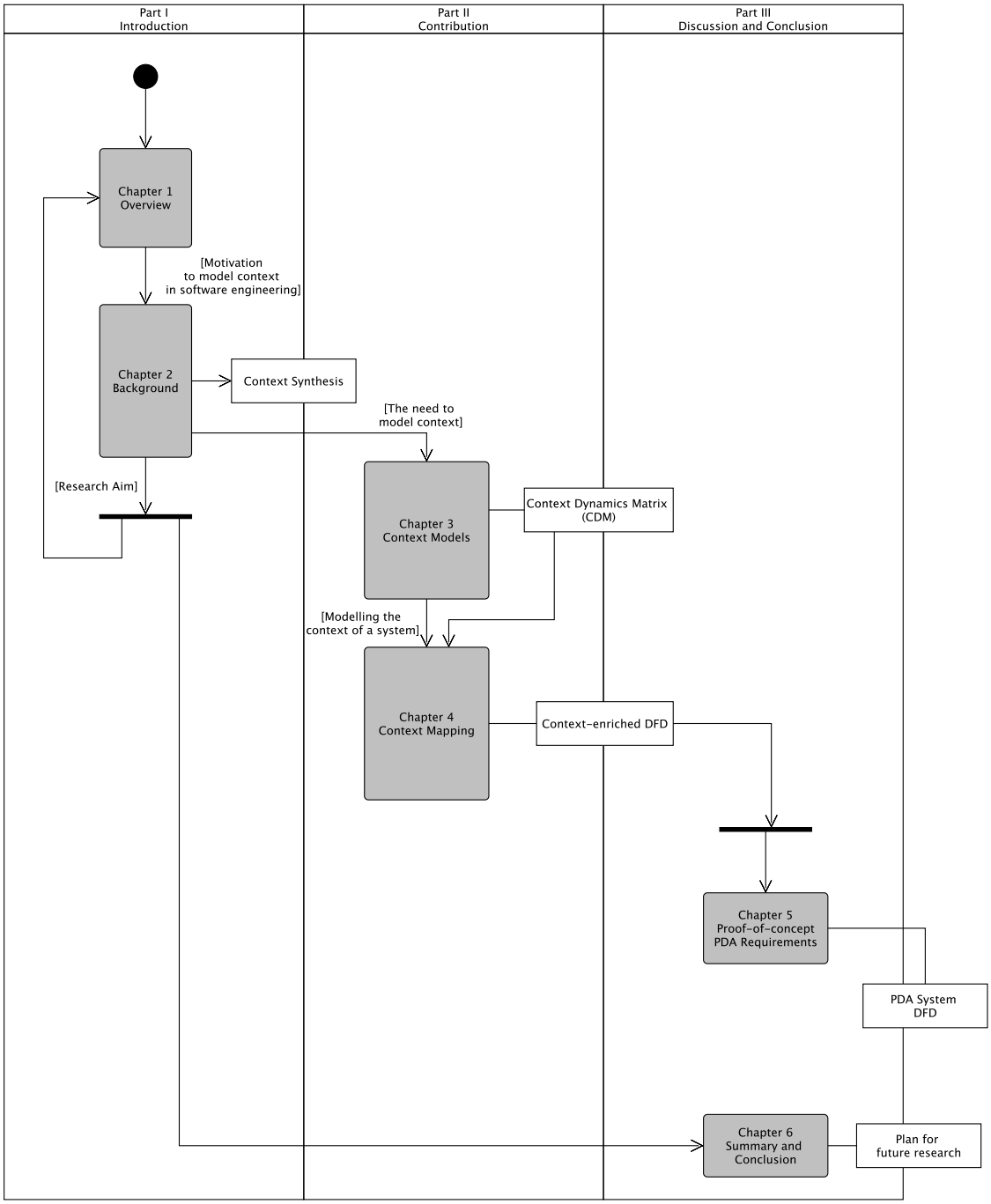


Figure 1.1 – Activity diagram depicting the structure and the flow of ideas and results throughout this thesis.

each part with more than one chapter. Chapters are represented as activities (grey rounded boxes), flow of ideas and conclusions between chapters are represented in arrows and key research contributions are represented by objects (white square boxes).

In the following sections I provide an overview of the thesis with a summary of the findings of each chapter. A more detailed overview of the main research contributions is obtained by reading the introduction of Chapters 3–4 with the conclusion of Chapter 6. A more concise overview of the thesis could also be obtained by reading the preface.

1.4.1 Part I - Introduction

The first part of the thesis (Part I) is formed by two chapters: Chapter 1 and Chapter 2. I present the thesis aim and motivation in this chapter, Chapter 1. Preliminary research is discussed in Chapter 2. In which initial observations leading to realise the need to explain system variety by modelling context is presented. These observations were inspired by the work of Alexander [1964; 1979; 2002] on context and form, Dilley [1999], Scharfstein [1989] on the problem of context, and the discussions of context and patterns in Buschmann et al. [2007]. The collection of ideas thereof, led to the conclusion that a synthesis of *context* is required in software engineering, expressing risk imposed and opportunities offered within the context of each system.

1.4.2 Part II - Contribution

The second part of the thesis (Part II) is formed by two chapters, Chapter 3 and Chapter 4. Chapter 3 discusses context on the level of individual elements, using the *Context Dynamics Matrix* (CDM), a novel approach to model context. Chapter 4 discusses context of individual elements within a system using Data Flow Diagrams (DFD). CDM is used to enrich the context representation of system elements' contexts as represented by DFD.

1.4.3 Part III - Discussion and Conclusion

The third part (Part III) is formed by two chapters. The first chapter, Chapter 5, is the proof-of-concept that describes an industrial case-study to demonstrate the efficacy of the approach. The second chapter, Chapter 6, I summarise the work presented in the thesis and present a survey of related work. I also discuss

limitations and propose future research directions to evaluate and further develop the use of context in software engineering.

1.5 Publications

The contributions made by this thesis are based on the results of preliminary research, some are presented in Chapter 2, and published in the following refereed conference papers.

- Z. Alshaikh and C. Boughton. The Context Dynamics Matrix(CDM): An Approach to Modelling Context. 16th Asia Pacific Software Engineering Conference (APSEC 2009), 2009.
- Z. Alshaikh and C. Boughton. Context centralised method for software architecture: A pattern evolution approach. In 3rd International Conference on Software and Data Technologies (ICSOFT2008), 2008.

1.6 Summary of contributions

The research reported by this thesis makes the following contributions to the existing knowledge of system and software development.

- **A synthesis of context**. A review of context in literature, produces a synthesis of context as two dimensions: influence and perception. The synthesis is based on five themes drawn from the literature that sets the plan for the following chapters to represent the context of systems.
- **Context models**. A novel approach to synthesise and represent the context of elements in terms of context states using CDM. The context synthesis introduced in Chapter 2, is expanded by a force model of influence, and a knowledge model of perception. Both models of context are represented in CDM, which implies sources of system variation when applied to software/system requirements.
- **Context mapping**. To represent the context of systems, the context states of the CDM are mapped to the description of requirements using DFD. The functional view that DFD represents is extended by context states. Thus it is possible for analysts to represent the context of a system as represented by requirement statements, through a context enriched DFD model.

Chapter 2

Background

> In analysis, something that we want to understand is first taken apart. In synthesis, that which we want to understand is first identified as a part of one or more larger systems.
> In the second step of analysis, an effort is made to understand the behavior of each part of a system is taken separately. In the second step of synthesis, an effort is made to understand the function of the larger system(s) of the which the whole is part.
> In analysis, the understanding of the parts of the system to be understood is then aggregated in effort to explain the behavior or properties of the whole. In synthesis, the understanding of the larger containing system is then disaggregated to identify the role or function of the system to be understood.
>
> Ackoff [1999]

Contents

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Bibliography

R. Ackoff. *Re-creating Corporation: A Design of Organization for the* 21^{st} *Century*. Oxford University Press, 1999.

C. Alexander. The determination of components for an indian village. In *Conference on design methods*, pages 83–114, 1963.

C. Alexander. *Notes on the Synthesis of Form*. Harvard University Press, 1964.

C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

C. Alexander. The origins of pattern theory: The future of the theory, and the generation of a living world. *IEEE Software*, pages 71–82, September/October 1999.

C. Alexander. *The Phenomenon of Life*, volume One of *The Nature of Order*. The Center for Environmental Structure, Berkeley, California, 2001.

C. Alexander. *The Process of Creating Life*, volume Two of *The Nature of Order*. The Center for Environmental Structure, Berkeley, California, 2002.

C. Alexander, S. Ishikawa, M. S. with Max Jacobson, I. Fiksdahl-King, and S. Angel. *A pattern language : towns, buildings, construction*. Oxford University Press, 1977.

R. Ali, F. Dalpiaz, and P. Giorgini. *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, chapter A Goal Modeling Framework for Self-contextualizable Software, pages 326–338. Springer-Verlag Berlin Heidelberg, April 2009.

Z. Alshaikh. Space Mission Operations: An Approach to a Unified Satellite Model in xtUML. Technical report, The Australian National University (ANU), July 2006.

Z. Alshaikh and C. Boughton. The Context Dynamics Matrix (CDM): An Approach to Modelling Context. In *16th Asia Pecific Software Engineering Conference (APSEC 2009)*, 2009.

V. Basili, G. Caldiera, and H. Rombach. *Encyclopedia of Software Engineering*, volume 2, chapter Goal Question Metric Paradigm, pages 528–532. Addison Wesley, 1994.

L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley, MA, USA, 2nd edition, 2003.

L. Bass, J. Bergey, P. Clements, P. Merson, I. Ozkaya, and R. Sangwan. A comparison of requirements specification methods from a software architecture perspective. Technical report, Software Engineering Institute , Carnegie Mellon, 2006.

L. Bass, R. Nord, W. Wood, D. Zubrow, and I. Ozkaya. Analysis of Architecture Evaluation Data. *Journal of Systems and Software*, 81:1443–1455, February 2008.

M. Beneceretti, P. Bouquet, and C. Ghidini. On the dimensions of context dependence: Partiality, approximation, and perspective. In *Modeling and Using Context*. Springer-Verlag, Berlin, 2001.

P. Bengtsson and J. Bosch. Scenario-based software architecture reengineering. In *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*, page 308, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8377-5.

D. M. Berry and E. Kamsties. Ambiguity in requirements specification. In J. C. S. do Prado Leite and J. H. Doorn, editors, *Perspectives on Requirements Engineering*, chapter 2, pages 7–44. Kluwer Academic Publishers, 2004.

D. M. Berry and E. Kamsties. The syntactically dangerous all and plural in specifications. *IEEE Software*, 1:55–57, January/February 2005.

B. Boehm and V. R. Basili. Software Defect Reduction Top 10 List. *IEEE Software*, January 2001.

J. Bosch. *Design & use of software architectures*. Addison-Wesley, London, 2000.

H. d. Bruin, J. C. v. Vliet, and Z. Baida. Documenting and analyzing a context-sensitive design space. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 127–141, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V. ISBN 1-4020-7176-0.

F. Bübl. Introducing Context-Based Constraints. In *Lecture Notes In Computer Science*, volume 2306, pages 249 – 263. Springer-Verlag, 2002.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture A System of Patterns*. John Wiley & Sons, 1996.

F. Buschmann, K. Henney, and D. C. Schmidt. *Software-Oriented Software Architecture On Patterns and Pattern Languages*, volume Five. John Wiley & Sons, Ltd, 2007.

H. Cappelen. Semantics and pragmatics: Some central issues. In G. Preyer and G. Peter, editors, *Context-Sensitivity and Semantic Minimalism : New Essays on Semantics and Pragmatics*, pages 3–22. Oxford University Press, 2007.

R. Carnap. *Introduction to semantics*. Harvard University Press, 1942.

S. Chemboli, L. Kane, and L. Johns-Boast. Using moodle to easily develop and deliver high quality courses. In *MoodleMoot*, 2010.

A. Classen, P. Heymans, R. Laney, B. Nuseibeh, and T. T. Tun. On the Structure of Problem Variability: From Feature Diagrams to Problem Frames. In K. Pohl, P. Heymans, K.-C. Kang, and A. Metzger, editors, *First International Workshop on Variability Modelling of Software-intensive Systems*, pages 109–117. The Irish Software Engineering Research Centre, 2007.

BIBLIOGRAPHY

A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a feature: a requirements engineering perspective. In *FASE'08/ETAPS'08: Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering*, pages 16–30, Berlin, Heidelberg, 2008. Springer-Verlag.

P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

J. O. Coplien and N. Harrison. *Organizational Patterns of Agile Software Development*. Pearson Prentice Hall, 2004.

J. Culler. *Literary Theory*. A Brief Insight. Sterling, 2009.

K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE 2005 - Generative Programming and Component Enginering. 4th International Conference*, pages 422–437. Springer, 2005.

A. Dearden and J. Finlay. Pattern Languages in HCI: A Critical Review. *Human-Computer Interaction*, 21(1):49–102, 2006.

S. Deelstra, M. Sinnema, and J. Bosch. *Experiences in Software Product Families: Problems and Issues During Product Derivation*, volume 3154, pages 120–122. Springer Berlin / Heidelberg, 2004.

T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press Upper Saddle River, NJ, USA, 1979.

J. Derrida. *Of Grammatology*. Johns Hopkins University Press, 1998.

J. Dewey. *University of California Publications on Philosophy*, volume 12, chapter Context and Thought, pages 203–224. University of California Press, Berkeley, 1931.

A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

T. Dijk. *Society and discourse: how social contexts influence text and talk*. Cambridge University Press, 2009.

R. Dilley. *The Problem of Context*. Berghan Books, 1999.

A. Endres and D. Rombach. *A Handbook of Software and System Engineering: Empirical Observations, Laws and Theories*. Pearson Education Limited, 2003.

G. H. Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010.

S. Ferber, J. Haag, and J. Savolainen. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In G. Chastek, editor, *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 37–60. Springer Berlin / Heidelberg, 2002.

A. Fetzer. *Recontextualizing context: grammaticality meets appropriateness*. John Benjamins Publishing Company, Amsterdam, 2004.

D. Fey, R. Fajta, and A. Boros. Feature Modeling: A Meta-Model to Enhance Usability and Usefulness. In G. Chastek, editor, *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 198–216. Springer Berlin / Heidelberg, 2002.

P. Feyerabend. *Against Method*. Verso, London; New York, 1988.

M. Foucault. *Archaeology of knowledge*. Routledge classics. Routledge, 2002.

M. Fowler. *Analysis patterns: reusable object models*. Boston : Addison Wesley, 1997.

E. Gamma and K. Beck. JUnit A Cook's Tour, last checked April 2010. URL {`http://junit.sourceforge.net/doc/cookstour/cookstour.htm`}.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch or Why it's hard to build systems out of existing parts. *IEEE Software*, 12(6):17–26, November 1995.

S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):1–45, 2009. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1455229.1455232.

E. Goffman. *Frame analysis: an essay on the organization of experience*. Harmondsworth:Penguin, 1975.

C. Goodwin and A. Duranti. Rethinking context: an introduction. In *Rethinking Context: Language as an Interactive Phenomenon*. Cambridge University Press, 1992.

M. L. Griss, J. Favaro, and M. d. Alessandro. Integrating Feature Modeling with the RSEB. In *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*, page 76, Washington, DC, USA, 1998. IEEE Computer Society.

B. Grne. Conceptual patterns. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBSí06)*, 2006.

R. Guha and J. McCarthy. Varieties of contexts. In *Modeling and Using Context*, page 164:177. Springer-Verlag, Berlin, 2003.

J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. In *Proceedings of IEEE International Requirements Engineering Conference (RE'02*, pages 137–144. IEEE Computer Society Press, 2002.

J. G. Hall, L. Rapanotti, and M. A. Jackson. Problem Oriented Software Engineering: solving the Package Router Control problem. *IEEE Transactions on Software Engineering*, 34(2):226 – 241, March-April 2008.

BIBLIOGRAPHY

M. Halliday. *Explorations in the functions of language*. Explorations in Language Study. Elsevier North-Holland, 1977.

G. Halmans and K. Pohl. Communicating the Variability of a Software-Product Family to Customers. *Software and Systems Modeling*, 2:15–36, 2003.

P. Harvey. Culture and context: The effects of visibilty. In *The problem of context*. Berghan Books, 1999.

D. J. Hatley and I. A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House, 1988.

M. Jackson. The world and the machine. In *ICSE '95: Proceedings of the 17th international conference on Software engineering*, pages 283–292, New York, NY, USA, 1995a. ACM. ISBN 0-89791-708-1. doi: http://doi.acm.org/10.1145/225014.225041.

M. Jackson. *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. ACM Press, 1995b.

M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, 2001.

B. E. John, L. Bass, E. Golden, and P. Stoll. A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns. In *EICS '09: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 3–12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-600-7. doi: http://doi.acm.org/10.1145/1570433.1570437.

K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, November 1990.

K. C. Kang, J. Lee, and P. Donohoe. Feature-Oriented Product Line Engineering. *IEEE Software*, July/August 2002.

D. Kaplan. Demonstratives. In J. Almog, J. Perry, and H. Wettstein, editors, *Themes from Kaplan*. Oxford University Press, 1989.

R. Kazman and L. Bass. Categorizing Business Goals for Software Architectures. Technical Report CMU/SEI-2005-TR-021, Software Engineering Institute, Carnegie Mellon, December 2005.

R. Kazman, L. Bass, M. Webb, and G. Abowd. SAAM: a method for analyzing the properties of software architectures. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 81–90, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-8186-5855-X.

R. Kazman, M. Klein, and P. Clements. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004, The Software Engineering Institute , Carnegie Mellon University, Pittsburgh, PA 15213, August 2000.

R. Kazman, L. Bass, M. Klein, T. Lattanze, and L. Northrop. A basis for analyzing software architecture analysis methods. *Software Quality Journal*, 13:329–355, 2005.

J. Kristeva. Psychoanalysis and the polis. In G. L. Ormiston and A. D. Schrift, editors, *Transforming the hermeneutic context*, chapter 4, pages 89–105. State University of New York Press, 1990.

P. Kruchten. The context of software development, July 2009. URL `http://pkruchten.wordpress.com/2009/07/22/the-context-of-software-development`.

P. Lago and H. van Vliet. Explicit assumptions enrich architectural models. In *Proceedings 27th International Conference on Software Engineering, ICSE 2005*, pages 206–214, May 2005.

R. Laney, L. Barroca, M. Jackson, and B. Nuseibeh. Composing requirements using problem frames. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 122–131, 2004.

K. Lee and K. C. Kang. Feature dependency analysis for product line component design. In J. Bosch and C. Krueger, editors, *Software Reuse: Methods, Techniques and Tools*, volume 3107 of *Lecture Notes in Computer Science*, pages 69–85. Springer Berlin / Heidelberg, 2004.

S. J. Mellor and M. J. Balcer. *Executable UML A foundation for Model-Driven Architecture*. The Addison-Wesley Object Technology Series, 2002.

B. Morris. Context and interpretation : Reflections on nyau rituals in malawi. In *The problem of context*. Berghan Books, 1999.

D. A. Norman. *The psychology of everyday things*. Basic Books, 1988.

D. A. Norman. *Emotional design: why we love (or hate) everyday things*. Basic Books, 2005.

D. A. Norman and S. W. Draper, editors. *User Centered System Design*. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey London, 1986.

W. E. Novak and L. Levine. Success in acquisition: Using archetypes to beat the odds. Technical Report CMU/SEI-2010-TR-016, Software Engineering Institute, September 2010.

I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan. Making practical use of quality attribute information. *IEEE Software*, 25:25–33, 2008.

K. Popper. *The Logic of Scientific Discovery*. Routledge, London and New York, 2006.

C. Potts. Using schematic scenarios to understand user needs. In *DIS '95: Proceedings of the 1st conference on Designing interactive systems*, pages 247–256, New York, NY, USA, 1995. ACM. ISBN 0-89791-673-5. doi: http://doi.acm.org/10.1145/225434.225462.

C. Potts and I. Hsi. Abstraction and context in requirements engineering: Toward a synthesis. *Annals of Software Engineering*, 3:23–61, 1997.

J. Ralyté, C. Rolland, and V. Plihon. Method enhancement with scenario based techniques. In M. Jarke and A. Oberweis, editors, *Advanced Information Systems Engineering*, volume 1626 of *Lecture Notes in Computer Science*, pages 103–118. Springer Berlin / Heidelberg, 2010. URL `http://dx.doi.org/10.1007/3-540-48738-7_9`.

F. Recanati. *Literal Meaning*. Cambridge University Press, 2004.

B. A. Scharfstein. *The Dilemma of Context*. NYU Press, 1989.

H. A. Simon. *The Sciences of the Artificial*. MIT Press, Third edition, 1996.

A. Skjeltorp and A. V. Belushkin. *Forces, Growth and Form in Soft Condensed Matter: At the Interface between Physics and Biology*. Springer, 2004.

M. Strathern. Out of context: The persuasive fictions of anthropology. *Current Anthropology*, 28:251–281, 1987.

R. N. Taylor, N. Medvidović, and E. M. Dashofy. *Software Architecture: foundations, theory, and practice*. John Wiley, 2010.

D. W. Thompson. *On growth and form*. Cambridge University Press, 1966.

T. A. van Dijk. *Discourse and Context: A Sociocognitive Approach*. Cambridge University Press, 2008.

H. C. A. van Tilborg, editor. *Encyclopedia of cryptography and security*. Springer, 2005.

J. Ven, A. Jansen, J. Nijhuis, and J. Bosch. Design decisions: The bridge between rationale and architecture. In A. Dutoit, R. McCall, I. Mistrík, and B. Paech, editors, *Rationale Management in Software Engineering*, pages 329–348. Springer Berlin Heidelberg, 2006.

P. T. Ward and S. J. Mellor. *Structured Development for Real-time Systems*. Yourdon Press, 1986.

K. E. Wiegers. *Software Requirements*. Microsoft Press, 2003.

L. Wittgenstein. *Philosophical investigations*. Oxford, Blackwell, 1974.

R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood. Attribute-driven design (ADD), version 2.0. Technical Report CMU/SEI-2006-TR-023, Software Engineering Institute , Carnegie Mellon University, November 2006.

E. Yourdon. *Modern Structured Analysis*. Prentice-Hall International Editions, 1989.

Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. Leite. From goals to high-variability software design. In A. An, S. Matwin, Z. Ras, and D. Slezak, editors, *Foundations of Intelligent Systems*, volume 4994 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2008.

العنوان:	Notes on the Synthesis of Context A novel approach to model context in software engineering
المؤلف الرئيسي:	Al Shaikh, Ziyad A.
مؤلفين آخرين:	Boughton, Clive(Super)
التاريخ الميلادي:	2011
موقع:	كانبيرا
الصفحات:	1 - 185
رقم MD:	616120
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة دكتوراه
الجامعة:	Australian National University
الكلية:	College of Engineering and Computer Science
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	صناعة البرمجيات ، برامج الحاسبات الالكترونية ، هندسة البرمجيات ، النمذجة ، النظم الخبيرة
رابط:	https://search.mandumah.com/Record/616120

Notes on the Synthesis of Context

A novel approach to model context in software engineering

THE AUSTRALIAN NATIONAL UNIVERSITY

A thesis submitted for the degree of
Doctor of Philosophy
of
The Australian National University

Ziyad A. Alshaikh

Feb 2011