



SOFTWARE ENGINEERING

Key Enabler for Innovation

NESSI White Paper

Networked European Software and Services Initiative

July 2014

Executive Summary

Economy and industry is experiencing a transformation towards software- and services-based businesses. Modern products and services increasingly embed software, or are customized, optimized or managed using software (examples include health, transportation, and utilities). Software engineering is thus playing an increasingly important key role in the responsiveness, quality and security of many industries. Mastering software challenges through advanced software engineering techniques, methods and tools is a must for all software-intensive industry sectors to stay competitive with their products and services. As an example, using cloud environments and applying big data approaches in the software development and software lifecycle is needed to keep pace with the accelerating market demands. As another example, being able to deal with the increasing complexity of software systems as triggered by cyber-physical systems or large scale distributed services requires fundamentally new models and approaches in software engineering.

In this white paper, the European Technology Platform NESSI (Networked European Software and Services Initiative) seeks to raise awareness for the continued and even increased need for **EU software engineering research and innovation programmes** in order for Europe to remain competitive and innovative. To this end, this white paper reflects NESSI's input to the forthcoming Horizon 2020 work programme (ICT/LEIT WP2016-2017). Specifically, it describes NESSI's view on software engineering research and innovation by identifying important research challenges and recommendations.

This white paper identifies relevant **software engineering research challenges** faced in software engineering for future software-intensive systems in three major technology areas:

- **Software engineering in and for the Cloud**
- **Software engineering for Cyber-Physical Systems**
- **Software engineering for and with Big Data**

This white paper also provides **recommendations** on how the specifics software should be addressed during **product and service innovation**. In addition, the paper provides recommendations on **skill and competency building** needed for a well-trained industry workforce. Finally, it delivers recommendations for maximizing the impact of **software engineering research and innovation projects** based on lessons learned from past FP7 projects.

As analysed and demonstrated throughout this white paper, software engineering principles, techniques, methods and tools need to evolve and novel ones need to be devised in order to keep up with fast-paced technology and societal changes and therefore being able to cope with the new challenges. Due to the growing complexity and multi-disciplinarity, software engineering solutions cannot be devised by companies and research organisations in isolation. Similar to other engineering disciplines, software engineering research and innovation requires **concerted efforts of industry and academia** to deliver practically relevant and significant solutions. This requires performing industry-near research exploiting real-world software engineering cases.

For Europe to remain competitive, this means that the opportunity and specific needs for such joint research and innovation efforts should be sustained; ideally, even increased. In addition, Europe should strive to better reuse, exploit and leverage already existing outcomes of past research projects. Together such activities will reduce the risk that Europe might lose the competitive ground on software and software-intensive systems and, as a result, will strongly depend on software technology and skills from non-European countries to a higher degree than advisable. NESSI considers the software engineering funding available in the current work programme (WP2014-2015 – LEIT/ICT-9) a modest starting point at best. **Software engineering research and innovation programmes need to be strengthened** if Europe wants to meet and leverage the opportunities of future ICT trends.

Contents

1.	Introduction	4
1.1.	Relevance of Software	4
1.2.	Importance of Software Engineering	4
1.3.	Major ICT Trends and Impact on Software Engineering	6
1.4.	White paper Contributions	7
2.	Software Engineering in and for the Cloud	8
3.	Software Engineering of Cyber-Physical Systems	11
4.	Software Engineering for and with Big Data	13
5.	Complementary Recommendations	14
5.1.	Software Engineering for Product and Service Innovation	14
5.2.	Software Engineering Skills and Competencies	16
5.3.	Best Practices for Software Engineering Research & Innovation	17
6.	Conclusions	19
	Authors and Contributors	21
	References	22

1. Introduction

1.1. Relevance of Software

Software has become ubiquitous in today's digital world. Software is embedded in almost all kinds of modern products and services of our surroundings [1] [2] [3] [4] [5].

From a **societal** point of view, software provides flexibility, intelligence and security to all the complex systems and equipment that support and control the different key infrastructures of our society: transportation, communication, energy, industry, business, government, healthcare, entertainment, etc. Software also has profound impact on our social life, most visible in the way how it changed the way in which we communicate, interact, interoperate and collaborate both in our professional and private digital lives. Software will also enable the public sector to transform and manage the new and growing challenging societal demands, e.g., in its service provisioning within healthcare, towards meeting an increasingly ageing population, building a 24/7 public sector service provisioning and educating children.

From an **economic** point of view, software is one of the main drivers of the European economy [1]. Software increases productivity and competitiveness in all business activities: industry, commerce, services, finance, etc. Software enables, for instance, competitive industry sectors to innovate and grow, and software fosters disruptive ideas leading to software-intensive products and services to become dominant in the market place. Software plays an instrumental role in the digital economy. Furthermore, software is embedded within the majority of products we use today and a key enabler for innovation, growth and employment in almost all sectors of the economy. Software has become the nerve centre of all modern societies [4]

From a **technologic** point of view, the traditional split into software and hardware and thus their respective business models will disappear. There is a strong shift from hardware to software, as value creation is moving up on the technology stack. This means we will see a *transition of vertical industry sectors* from being very hardware intensive and product-based with respect to development costs to industries where innovations and development costs are much more software-driven, and where the notion of service has become a dominant factor in market offerings. Software will be increasingly provided by services accessed from a wide range of terminals (personal or collective, fixed or mobile) and the computer will be progressively replaced by the network. This means we will see the *availability and usability of software-based solutions and services*, both on single platforms (e.g., smartphones) and through the Internet (e.g., cloud-based services), going from stationary and stand-alone to mobile and interconnected usage patterns – anywhere and anytime.

In summary, software has had a predominant role over the last 15 to 20 years in our digital world and will remain a driving force for its continued transformation. Software is the **key enabler for innovation** [6]. It allows delivering differentiating features and services, and doing so with short turnaround times and high speed to market. As a result, software has become the prime industrial differentiator and basis for innovation [3]. It is not simply there in its own right, but serves as a key enabler that needs to be developed in close cooperation with other R&D units and domains: Software is what makes most of our modern products and services work.

1.2. Importance of Software Engineering

As Commissioner Viviane Reding has already rightly pointed out in her speech at the Truffles 100 meeting in November 2007, "the ability to produce software is a strategic economic capability" [1]. This means that the key role of a "systematic approach to the development,

operation, maintenance, and retirement of software” [7], i.e., **software engineering**, on a European level is well understood. Since then, this view has been emphasized by many different groups (e.g., see [3] [1] [4]). The importance of software engineering will remain and even grow and thus needs to be sustained and strengthened also in forthcoming research and innovation efforts beyond the current EU work programme.

It must be emphasized that software engineering is not just programming: Metaphorically speaking, software engineering is related to programming in a way that building design is related to laying bricks. “Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.” [7]. Software engineering thus constitutes an essential capability for European industry, and high quality of software can turn into a competitive advantage.

Software engineering research and innovation delivers new methods, techniques, mechanisms, languages and tooling which advance software production and engineering in itself. Software engineering research thus delivers principles, techniques, methods and tools that explain how to efficiently and effectively build software systems with reliable quality guarantees (such as security, safety, privacy, performance and trust) and fulfilment of the expectations of the users and business owners. Software engineering research and innovation ensures that European industry will remain skilled, capable and competitive in delivering software and software-based products to their customers and markets.

In contrast to other engineering disciplines, software engineering needs to account for specific properties due to the unique characteristics of software as artefacts. Those characteristics include:

- **Immateriality:** Software is everywhere today. But even though we can find software in most of today’s products and services, it is usually not visible. Because software is so ubiquitous and not directly visible, its significance to Europe is not always easy to identify or appreciate [4]. Due to this immateriality of software it may appear to be easily and arbitrarily changeable, which – in industrial reality – may imply significant quality problems and significant effort for identifying defects in software.
- **Servicification:** Immateriality of software is further amplified by virtualisation of software assets (such as middleware, tools and whole applications), which can in turn be offered as services. A service represents functionality with associated quality characteristics (typically defined in a service-level agreement) offered by a service provider via a service interface [8]. The service itself may change as long as the functionality and the service-level agreement remain the same, thereby providing an increased level of flexibility.
- **No manufacturing:** The main cost drivers for software are personnel costs. Software can be easily copied, distributed and deployed with basically no costs. Hence, software engineering costs are determined by the effort and time invested in terms of human resources. Industrial value creation is progressively shifting upwards the technology stack since the effort invested in software development and engineering is continuously increasing. This leads to an increased need for efficient and effective software engineering methods, techniques and tools.
- **Dedicated skill set:** The success of software engineering projects to a large degree depends on the quality of personnel and their skills and education. Sound software engineering principles, techniques and methods are essential elements of these peoples’ skill set and significantly go beyond what can be expected from an ordinary programmer.

1.3. Major ICT Trends and Impact on Software Engineering

Even though software engineering research has produced impressive results over the past years, new major trends in information technology lead to an amplification of existing challenges as well as the emergence of previously unknown novel challenges (see Figure 1 for an overview).

This white paper looks into the following three mega trends in information technology to cluster the discussion of software engineering challenges:

- **Software Engineering in and for the Cloud:** Cloud computing is an important innovation driver of the current and the next decade, affecting not only the software and services sector, but all software-intensive sectors that benefit from software innovations. Clouds will increasingly become the paradigm for delivering all kinds of services, from IT services to full business services, and they will constitute collaboration hubs for all types of business networks [2].
- **Software Engineering of Cyber-Physical Systems:** Cyber-physical Systems (CPS) enable new kinds of embedded system services by integrating networked embedded systems with services of the information systems infrastructure, thereby being able to interact with and expand the capabilities of the physical world through computation, communication and control. Cyber-physical Systems thus form an important basis for the development of innovative products and services [9].
- **Software Engineering for and with Big Data:** Big Data is about extracting valuable information from data to use it in intelligent ways such as to revolutionize decision-making in businesses, science and society, thereby enhancing the companies' competitiveness and leading to new industries, jobs and services [10] [11].

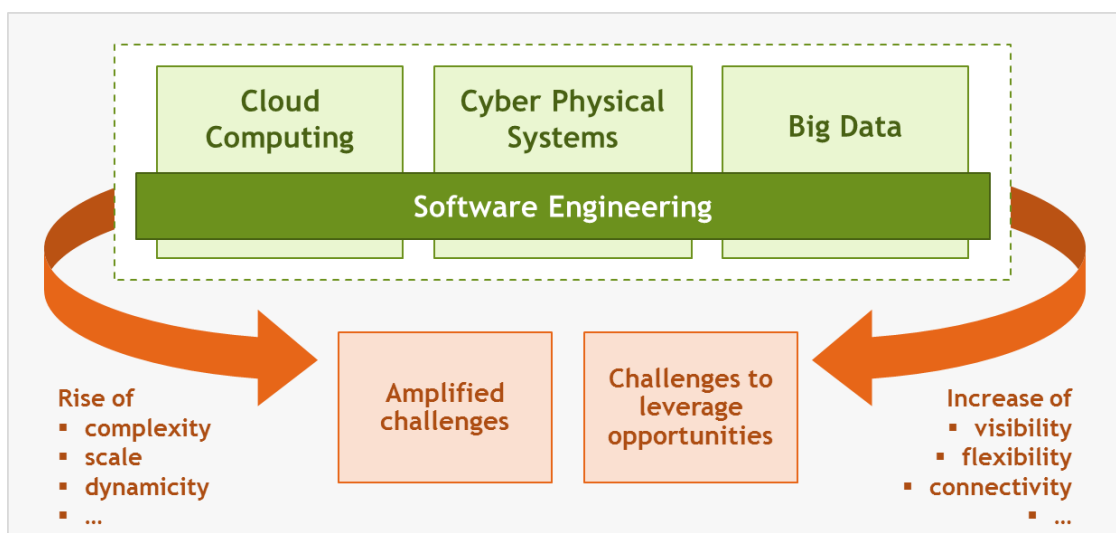


Figure 1: Software engineering challenges arising from trends in information technology

These technology trends and the resulting requirements influence how software systems are built and thus lead to challenges that need to be addressed with novel software engineering principles, methods and practises. As an example, only one or two decades ago, software development was all about stand-alone, monolithic systems, i.e., a closed and controlled world. Today it's about loosely coupled, interconnected, interoperable, adaptive and autonomous components, where we cannot assume to grasp the total as the sum of its parts. In

such an open world, one part has no or only partial control of its surroundings, i.e., of the other parts.

What is important to stress is that two kinds of research challenges emerge from those technology developments as indicated in Figure 1.

- On the one hand, there are many existing software engineering challenges that are now **amplified** due to complexity, scale or dynamicity implied by novel information technology. Examples for such challenges include the efficient and effective migration of legacy code to the cloud, the question of how to cope with increasingly complex and dynamic networks of systems of systems and the effective testing of Big Data applications in order to ensure their quality.
- On the other hand, novel information technology offers new **opportunities** that can be leveraged to address problems from a new angle. As a result, this leads to new challenges on how to leverage those opportunities. Examples include questions of how to use cloud computing to deliver energy efficient software, how to use immediate feedback from cloud applications to better understand customer needs or how to use Big Data analytics to systematically determine trends in open source software communities and thus better manage a company's open source asset base.

1.4. White paper Contributions

NESSI aims to have an impact on the technological future by identifying strategic research directions and proposing corresponding actions. NESSI gathers representatives from industry (large and small), academia and research organisations, and public administration and is a European Technology Platform (ETP) active at an international level (see <http://www.nessi-europe.eu/>). NESSI closely monitors technology and policy developments in the software and services domain.

The importance of software engineering has been emphasized by many different groups as mentioned above (e.g., see [3] [1] [4]). Many of the claims and recommendations of previous papers on software and software engineering remain valid. This paper provides an update to reflect on recent technology developments and trends. In addition, it provides detailed and operational recommendations concerning software *engineering* – and not only for the general topic of software (see the differentiation in Section 1.2). To this end, this white paper makes two major contributions:

As a first major contribution, the white paper identifies relevant **software engineering research challenges**

- in and for the **Cloud** (Section 2),
- for **Cyber-physical Systems** (Section 3) and
- for and with **Big Data** (Section 4).

As a second major contribution, the white paper provides **recommendations** concerning software engineering for **product and service innovation**, software engineering **skills and competencies** and best practices for organising software engineering **research and innovation projects** (Section 5).

In order to identify the software engineering challenges, NESSI partners provided pressing industry cases that show the limitations of current software engineering. These industry cases served both as examples for concrete challenges as well as a basis for discussing and identifying more general challenges. After an initial set of challenges was identified, NESSI members have been consulted and invited to confirm the relevance of the challenges and provide additional aspects and perspectives on software engineering research and innovation.

As a result, the challenges and recommendations presented in this white paper reflect relevant areas for software engineering research and innovation identified by NESSI partners and members. These challenges and recommendations may thus be instrumental in shaping future research and innovation investments and funding actions.

2. Software Engineering in and for the Cloud

From the end users' viewpoint, telematics systems in the 80s were the first occurrence of services "in the cloud" available to a large portion of the population in some countries, introducing them to online services (banking, travel, information databases, text communications, administration, and so forth). From a technology viewpoint, research on distributed systems and their principles paved the way to grid computing and utility computing in the late 90s. Today, the most commonly adopted definition for Cloud states that "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services that can be rapidly provisioned and released with minimal management effort or service provider interaction." [2].

Originally, cloud computing was a way to mitigate costs and replace capital expenditures with operation expenditures by the IT department of big companies, whereas cloud services now increasingly become an agility, productivity and performance driver for a large number of companies (impacting their processes and organisation) and, associated with the broadband networks and smart devices, a digitalisation driver for the mass market users.

Cloud is a powerful accessibility and innovation trigger: it allows for a progressive and streamline use of a given service, lowering the setup cost; a newly available technology, integrated into a cloud infrastructure can be directly exposed to a large community of adopters, even those with limited resources; data analysis on a large scale and agility of service modification allows for a better adaptation to socio-economic and usages trends, platform-based innovation is natively supported by the cloud.

The use of virtualisation to isolate software from hardware specificities together with the availability of high performance networks led to a shift of most of the design of IT and telecommunications systems from an equipment focus to a software focus: from the operating system and databases to the interactions performed by the users, software brings an unparalleled flexibility for the services, to address both functional concerns (adaptation to the users' needs, to the business requirements, to laws, markets or usages) and non-functional concerns (scalability, performance, optimisation, security, etc.). Software engineering is now at the core of the creation and evolution of a huge number of cloud services, spreading in all corners of society (health, security, education, industry, services, utilities and so forth), representing a new step in the digitalisation of the world.

One can differentiate two kinds of areas for software engineering: the ones focussing more on infrastructure concerns (including Infrastructure- and Platform-as-a-Service), and the ones focussing more on application and process concerns (including Software-as-a-Service and Business-Process-as-a-Service). Figure 2 visualizes those two areas. Considering **infrastructure** concerns allows enterprises and administration to mutualise and optimise their investment on IT infrastructures and to create value by improving the quality, security, availability and scalability of their products and services. Considering **application and process** concerns allows industry to be reactive, to stay competitive and to closely follow the needs of their users while allowing them to complement their offers in interconnected ecosystems.

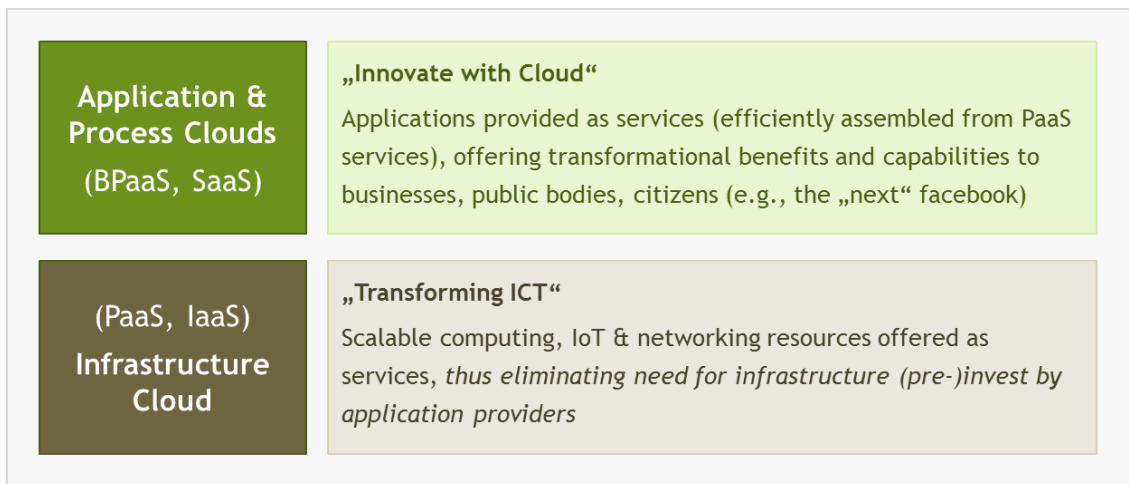


Figure 2: Two main concerns for software engineering in the Cloud (adapted from [2])

Software engineering challenges focussing on infrastructure concerns include:

Challenge Cloud-1: Reliable distributed middleware for decentralised computing and data storage to ensure predictable behaviour and quality. How to ensure predictable behaviour and quality taking into account failure recovery, migration, transactions or dynamic reconfiguration? How to foster the integrated management of cloud and network infrastructures?

Challenge Cloud-2: Languages and APIs for transparency. How to provide software applications with native support for transparency, e.g., about distribution, failures, heterogeneity, adaptation and elasticity of the cloud infrastructure, leading to the notion of Monitoring-as-a-Service? What are suitable, powerful language primitives for cloud applications, e.g., in order to support database queries, event processing and reactive frameworks?

Challenge Cloud-3: Large scale optimisation concepts and heuristics for the deployment of applications and services. How to optimise deployment of applications and services, while taking into account global energy consumption, trade-offs between network/compute/storage, as well as infrastructure management?

Challenge Cloud-4: Model-driven deployment for non-homogenous clouds. How to make use of model-driven approaches for the cloud; e.g., in the form of application “blueprints” from which the deployment of cloud elements is automatically generated? How to ensure that software can be deployed efficiently on heterogeneous computing elements such as CPUs and GPUs? How to adapt software to different devices along the compute continuum, including M2M and IoT devices?

The development of applications and processes as cloud services focusses on the requirements from the users’ perspective and from the software owners’ perspective. This requires agile adaptation of software to the highly dynamic evolution of markets and usages, requiring very fast (within days and minutes and not weeks or months) adaptation of applications (e.g., see [12]), integration and orchestration of services built from various sources, possibly using internal and external cloud services through standard or specific APIs, real-time monitoring of applications and infrastructure to allow for adaptation or optimisation, possibly using knowledge acquired from Big Data systems, easy provisioning of software as a service available to the largest range of platforms and devices, using simple interfaces and APIs and

taking care of business key factors such as security, accounting, billing, management, privacy and so forth. Agile development is one key approach towards shortened development cycles. However, in many relevant industrial settings the tension between upfront investment and planning of a stable software core and the increased agility fostered by instantaneous feedback and continuous deployment must be reconciled [13].

As a result, **Software engineering challenges focussing on application and process concerns** include:

Challenge Cloud-5: Software engineering for cross cutting concerns of cloud applications. How to address quality concerns, such as privacy, security and portability between clouds? How to adequately monitor and control data migration, thereby fostering privacy enforcement and compliance to legal constraints?

Challenge Cloud-6: Software engineering for cloud services on personal and embedded devices. How to design responsive user interfaces? How to handle the distribution of processing and data between the cloud and the local devices? What are the right frameworks and formats reducing the cost of employing multi-device services and applications?

Challenge Cloud-7: Methods and tools for agile life cycle support of cloud applications including the development, testing, deployment and management of cloud applications. What are the right methods and tools for development, testing, deployment and management of cloud applications? How to design tools that drastically reduce the cost of domain- and application-specific software development and maintenance through high-level frameworks, platforms and languages?

Challenge Cloud-8: Global cloud application frameworks including effective languages and reliable architectural patterns. How to adapt application frameworks to the cloud, thereby natively supporting – among others – processes and data migration, as well as reactive programming (triggered by API calls, user actions or infrastructure notifications)? Will this be possible by extensions of existing languages or do we need specific languages? How to capitalise on proven and reliable architectural patterns and how to adapt them for specific types of applications, while ensuring scalability, fault tolerance, and data redundancy? How to link those application frameworks with software platforms supporting SaaS ecosystems, including service provisioning, billing, and operations?

Challenge Cloud-9: Developing on-demand (cloud) and on-premise software applications. How to deliver applications both in the Cloud (on-demand) as well as on premises in order to address customer needs? Can we (jointly) develop software for both models?

Challenge Cloud-10: Cloudification of Legacy. How to support the cloudification of software elements that may not be ready for virtualisation? For instance, how to cloudify data bases? Or, how to make a software component scalable that has not been initially built for elastic infrastructures?

The ultimate goal of software engineering for the cloud could be to make the dream “the cloud is the computer” come true: being able to program a service on a large number of distributed resources as simply as it were a single-processor computer, with nearly infinite power and memory and resilience to failures, a true “virtual machine”.

3. Software Engineering of Cyber-Physical Systems

Embedded Systems provide intelligence to physical objects of everyday life (i.e., products artefacts and systems such as cars, aircraft, trains, personal devices, medical devices, industrial plants, power plants, etc.), making them smart objects. In the coming years, embedded systems will increase the intelligence, control and communication capabilities of a wide range of objects, enabling their interaction and cooperation with people and organisations also in the physical world thanks to their actuation capabilities. Such smart objects will be joined together to create highly distributed systems, called Cyber-Physical Systems (CPS), by bringing a wealth of opportunities and innovations in technology, applications and business models [3].

The emergence of highly distributed and large-scale CPS means that software has to live in an *open* and highly dynamic world. Traditionally, software development was based on the closed world assumption, which means that the boundary between the system and its environment is known during design-time and that the environment does not change while the system is executing. In contrast, open world systems cannot be specified completely during design-time due to incomplete knowledge about, for instance, services and their actual quality at provisioning time, sensors available during system operation to obtain contextual information, the availability of other systems to interact and cooperate with, the amount and quality of data obtained, as well as context changes faced during operation. In addition, this world is at the same time “cyber” (e.g., monitoring/sensing cyber activities and actuating operations in the virtual world, such as adapting Web-based services or generating posts over social networks) and “physical” (e.g., monitoring/sensing environmental indicators and actuating response operations in the real world, which cannot always be cancelled or rolled back without tangible effects, such as commanding Heating, Ventilation and Air Conditioning (HVAC) systems or opening doors in home automation scenarios). The development of CPS thus has to live inherently with uncertainty in the specifications [14]: during operation, such systems must frequently adapt to the executing environment changes faced at run-time and must be able to continue to behave in a controlled and safe way, thus posing novel technical challenges for the software engineering of services and applications for CPS [13].

Given the unique and challenging aspects of Cyber-Physical Systems (CPS) sketched above, the realisation and market-adoption of CPS services and applications calls for addressing the following key **challenges on software engineering for Cyber-Physical Systems**:

Important quality aspects for Cyber-Physical Systems include scalability, e.g., ensuring that CPS applications can scale to urban-wide deployment environments. CPS solutions call for capabilities to monitor, control and manage the quality/performance constraints at provisioning time in open and dynamic executing environments, possibly via prioritised operations in response to the growing distance from expected and allowed behaviour.

Challenge CPS-1: Handling quality and performance requirements in large-scale, open environments. How can we address stringent quality and performance requirements in software engineering methodologies and solutions for CPS? How can we handle situations when dealing with large-scale open environments such as smart cities, where sensing quality may be extremely differentiated and tasks to actuate are subject to many different sources of uncertainty? How can we extend testing and formal verification to deal with uncertainty and variability at the same time?

Given the potential complexity and hard technical challenges associated with the inherent nature of Cyber-Physical Systems, leaving the whole burden of context monitoring, integration and adaptation to application developers will not be sustainable.

Challenge CPS-2: Principles, methods and tools supporting the software life-cycle of Cyber-Physical Systems. What are adequate principles, methods and tools to significantly reduce development costs and time-to-market for Cyber-Physical Systems? How can we leverage continuous integration, testing and certification to this end? How can we provide life-cycle support for millions of decentralised system instances?

Many kinds of Cyber-Physical Systems, especially the ones involving wide-scale deployment environments, call for the integration with a relatively large amount of server-side (Cloud) processing/storage resources. Examples include systems where trend identification analytics or final user applications are run.

Challenge CPS-3: Integration of Cyber-Physical Systems with Cloud and Big Data solutions. How can we integrate Cyber-Physical Systems with Cloud and Big Data infrastructures? How can we bring the processing of data closer to the data sources in order to properly and innovatively manage the flow of information? How can we model application and sensor profiles to reduce inefficient continuous interaction between CPS and server-side resources?

In cockpits and control towers, human operators are involved to interpret data, to judge the criticality of a given situation and to decide on the adaption of an application during run-time as a reaction to foreseeable and unforeseeable changes and exceptions. Also, smart spaces often involve human actors, e.g., users who can perform actuation operations based on CPS application suggestions (persuasive computing), thus dynamically modifying the execution environment and context with the uncertainties connected to human participation and involvement.

Challenge CPS-4: Considering human-in-the-loop aspects and adaptation in Cyber-Physical Systems. How can we provide software system operators with dedicated adaptation mechanisms to leverage human decision making for adapting Cyber-Physical Systems to unforeseeable situations? For instance, can we adopt human-in-the-loop principles, such as cockpits and control towers? May such principles provide a path for developing systems for which the closed world assumption does not hold anymore? From a different angle, How can we consider user characteristics and behaviour during the design of (adaptive) Cyber-Physical Systems? User incentives, “punishments” and user-operated actuation may provide novel ways of adapting the environment of CPS. How can we leverage those for continuous observation-analysis-adaptation loops to significantly change the way CPS services and applications are designed and put into execution?

Challenge CPS-5: Middleware and platforms for dynamic choreography and adaptation of Cyber-Physical Systems. How to deliver novel Cyber-Physical System-oriented support platforms with cross-layer visibility of both application requirements and low-layer context information? How can we engineer for run-time adaptation based on this visibility? To which extent can we separate the implementation details of the adaptation mechanisms from the business logic of cyber-physical applications? How can we support the choreography of autonomous sub-systems and handle the large number of sub-systems in dynamic environments? How to balance the capabilities of the platforms with needs towards resource efficiency; e.g., using big and powerful frameworks (“bloat-ware”) vs. using specific “hand optimised” programs? Can we optimise or prune unused code and components?

Challenge CPS-6: Novel, powerful programming abstractions for implementing Cyber-Physical Systems. What are the right abstractions that are easy to understand and use, but at the same time sufficiently expressive to be mapped efficiently and ideally automatically to executable code? What would be adequate abstractions (models) and in-

formation-hiding principles (interfaces)? How can we use those abstractions to address unexpected conditions and sensing/actuation component faults and to define safe operational areas? How can those abstractions be used to support deployment on diverse devices and hardware configurations?

4. Software Engineering for and with Big Data

The continuous and tremendous growth of data volume, the better accessibility of data, and the availability of powerful IT systems have led to intensified activities around Big Data [11]. The volume, velocity, variety and need for veracity of data has exploded in the past years because of new social behaviours, societal transformations as well as the vast spread of software systems and Cyber-Physical Systems [10].

One important aspect of software engineering when it comes to Big Data is related to the design of software systems. Using Big Data technologies to design and build large scalable data systems creates a significant software architecture challenge for software architects. The challenge is posed primarily by the scale factor where software architects must explicitly deal with issues typically appearing in distributed systems. Problems like data replication, data consistency, temporary failures, communications latencies and concurrent processing need to be explicitly addressed in the system design. Such issues are amplified in a Big Data context, where systems need to dynamically grow to utilize data geographically distributed.

Overall, the following **software engineering challenges for and with Big Data** arise:

Today, NoSQL and MapReduce are predominant for the efficient storage, representation and query of Big Data. However, apart from large, long-standing batch jobs, many Big Data queries involve small, short and increasingly interactive jobs.

Challenge BigData-1: Scalable software architectures for Big Data. How to address the fundamental issues of scalability, performance, and availability that become necessary when dealing with Big Data systems and applications that have to cope with unprecedented size, speed, diversity and noise of data? How to support such kinds of jobs and deliver new architectures that, for instance, combine classical RDBMS techniques for storage and querying on top of NoSQL and MapReduce paradigms? This will lead to a new generation of software designs that optimise Big Data querying and retrieval.

Due to the large quantities of data, possibly very heterogeneous, it becomes challenging to create comprehensive test suites and environments to sufficiently cover and validate the software before it is deployed in production environments.

Challenge BigData-2: Leveraging software engineering techniques for quality assurance of data-intensive software. How can we ensure the quality of Big Data software through adopting and extending proven quality assurance techniques from software engineering? Can we generate (for instance by means of simulation) sufficient and representative test data (e.g., covering extreme cases) in order to ensure resilience and robustness of Big Data applications? Can we complement testing with (formal) verification techniques for Big Data? How can we leverage fast prototyping to test the quality of Big Data applications early on during development; e.g., by using interpreted languages for fast feature deployment and debugging?

Challenge BigData-3: Online diagnosis of data-intensive software. How can we monitor and thus ensure the quality of Big Data systems during their operation? The analysis of monitoring logs may itself be considered a Big Data problem as logs for complex systems

can easily reach a large size in small periods of time. How can we use Big Data techniques to analyse Big Data systems in operation? Can we identify undesired patterns and deviations by analysing the massive amount of “meta-data” being collected?

Despite the abundance of storage at relatively low cost, the storage and query of data at a large scale will continue to remain challenging. Cloud storage services usually are not adequate as they lack range query support, and support for transactional semantics for operations spanning multiple keys.

Challenge BigData-4: New and improved software algorithms for data streams and storage. How can we build novel algorithms that store and cluster data objects – in dynamic data stream mode – and subsequently facilitate searching and retrieving information, as well as presenting it in a useful manner? How can we leverage efficient storage techniques to greatly decrease the processing time of data? Could we optimize storage across many nodes in a cluster in order to leverage more efficient designs?

Challenge BigData-5: Big Data engineering methods and frameworks. How can we support the engineering of Big Data applications through targeted methods and platforms? How can we pave the way from online analytical processing (OLAP) systems to full-fledged Big Data analysis frameworks that bring Big Data technology into a systems perspective? As a foundation for such frameworks, we need new data organisations that better fit the intrinsic data cube model of n -dimensional data. These would include the partitioning and distribution of data in several tables to enable parallelism, whereas indexing, replication and data management hierarchies could improve execution efficiency and throughput.

Even now, data mining of forums, forges, blogs and social networks allows detecting usage trends of application frameworks, open source components, etc. Also, analysing the data collected from Cloud applications (e.g., Software-as-Service offerings) can be used to detect user trends, preferences, as well as the needs to evolve and adapt applications.

Challenge BigData-6: Using Big Data analytics during software engineering. How can we employ Big Data analytics to address current software engineering problems (e.g., to better understand user needs; to identify the points in an application that should or should not be adapted to context changes; to perform root cause analysis of software failures by mining memory dumps of complex software systems)? What other novel uses of Big Data analytics for addressing software engineering problems may become possible?

5. Complementary Recommendations

Where the above sections have elaborated on concrete software engineering challenges, this section provides complementary recommendations on innovation, skills and project organisation.

5.1. Software Engineering for Product and Service Innovation

Software engineering today extends beyond mere development. It is rather an end-to-end process from inspiration and ideation in the early stage to product sunset. Key questions along this end-to-end creation process comprise ensuring the desirability, viability and feasibility of the software product or service, which means being innovative according to Tim Brown’s definition [15].

With current technology trends including Cloud, Big Data, Agile Development and DevOps, software engineering is becoming smoother and faster than ever. There was a time when the development of the information system often was a limiting factor. Today, software systems and services are an accelerator and they have major impact on all processes of companies. This includes processes in the front and the back office, in the “production lines”, and in support. It affects marketing and the technical teams, as well as managers. All of this places software engineering at the focal point of business development and creates new challenges, such as adopting agile development, fostering short development and release cycles, running continuous integration, involving distributed communities, as well as exploiting and managing open APIs and open source (see Sections 2–4).

Leveraging the accelerating capability of software engineering to foster innovation is currently addressed by combining contemporary approaches like agile development, often in the form of Scrum, design-thinking, as well as business model development. Customer centricity and customer co-innovation is at the heart of these approaches as a common denominator and a touch point (cf. [16] [17]).

One key obstacle that needs to be overcome for customer-centred product and service innovation is to make the transition from traditional, standard software development via first co-innovation with customers and initial target markets to a broader scale in terms of market segments and global adoption. In order to shape the next generation of software companies and customer-driven innovations it is thus essential to identify required organisational setups, team compositions, professions and job functions to support this.

To truly leverage software as an accelerator for innovation, the following **recommendations may ensure the transitioning from research outcomes to innovation.**

Software development or software creation comprises the overall process from early inspiration and ideation to prototyping and testing to implementation and early adoption. In terms of customer-centred product innovation, the focus is on desirability and viability and to a lesser extent on feasibility. It is mainly about engaging customers and end users closely into the product creation process.

Recommendation Innov-1: Leveraging software creation as key enabler for innovation creation. It is crucial to understand and practice customer co-innovation and validate assumptions and prototypes with end users while clarifying related questions about intellectual property. The viability discussion usually starts with understanding the customer value in terms of what they are willing to invest and pay for. Therefore, the product managers, developers and designers and their interplay are required and need to be combined with an agile, iterative and feedback-driven development approach. In the context of desirability the role of designers is increasingly important and hence the profession of designers is currently redefined.

With the internationalisation of software development, companies are increasingly facing challenges in various areas that require new approaches. In general, larger software companies have to manage global development setups and global product delivery with major issues regarding dependency and version management, for instance.

Recommendation Innov-2: Optimising software development resource allocation. To optimise resources (e.g., software designers and support personnel, as well as cross functions) in a global environment, the setting up of shared service centres for development projects should be investigated. Novel mechanisms (including organisational structures, as well as novel methods and software tools) are required to cope with the complexity of handling such global resource sharing. Complementary, systematic reuse of software components needs to be considered as a further means to optimise resource usage and thus productivity, ideally based on platforms and standards.

The most critical phase for most new products, services and start-ups is after they acquired their first customers. This is due to the challenge of expanding beyond the initial target market and achieving global adoption. The latter especially requires globalisation in terms of multi-language support for products and required translations. Today's focus on mobile applications ("responsive designs") puts special emphasis on language characteristics during software design and documentation.

Recommendation Innov-3: Globally scaling software product and service successes. To scale initial local innovation and market success globally, new approaches based on machine translation and new business models that, e.g., leverage crowd authoring and editing, are required in order to ensure multi-lingual offerings and thus foster adoption. Complementarily, new concepts for product documentation, e.g., based on video and multi-media material can become a key differentiator in terms of usability and thus adoption. Moreover, in order to cater to the upcoming need for responsive design and cloud-based applications, software companies need to manage continuous delivery and updates of documentation and training content as well.

5.2. Software Engineering Skills and Competencies

Throughout this white paper, the importance of software technology and software engineering as the key driver for European economy and innovation has become apparent. Software enables us to transform business models and enterprises and even whole sectors of our society, and to become more efficient, cost effective and sustainable [4] [3].

Not only is software everywhere, but everyone should have knowledge of its potential and power for change. Software is in many respects the tool of our time as were pencil and paper in former times. Hence, software literacy should become a basic skill that we need to educate our children in.

What impact does software have on the skills and competences of our citizens? The most revealed impact is that software development may be carried out by people of a very broad range of backgrounds: From teenagers or young entrepreneurs who have set up an app business in their parents' home to professionals from a wide range of disciplines, many of which have primarily learned programming, and then software engineering, by doing, meaning by trial and error. Even though they may not have a formal educational background in software engineering, some may have had one or two (undergraduate) courses in programming. Hence, a vast amount of (often safety-critical) software is developed by people with no formal software engineering background. Is this situation for the good or for the bad?

In autumn 1968, the NATO Software Engineering Conference [18] defined software engineering with the long term goal of solving the problem of a "**software crisis**", which 25 years later was coined by W.W. Gibbs as a chronic crisis [19]. Even today, software engineering is far from as mature as we would like it to be. This is of course a function of the rapid ICT development, the massive penetration of ICT, the pressing reliance on software to solve social and business problems and so forth. But software engineering is also much about context such as social, communicative and psychological to mention a few, and even criticism against software engineering as engineering per se (e.g., [20]).

As most European industries are highly software intensive, software engineering became (explicitly or implicitly) an integral part of other engineering disciplines and technology fields.

The Graduate School Directory [21] shows that software engineering is offered in Bachelor, Master and PhD programmes. However, a closer look into these programmes shows that software engineering modules are mostly offered as part of existing, traditional programmes

of computer science or informatics. Dedicated software engineering programs where the required engineering skills and competencies are explicitly addressed exist only to a smaller extent.

Based on what has been described above, the following two **recommendations are made to strengthen software engineering capacity and the number of skilled software engineers** in Europe:

Recommendation Skills-1: Fostering software engineering skill building during research and innovations activities. Research and innovation projects where academia and industry meet should be understood as an excellent, complementary way to deliver and educate well-trained graduates. Projects foster learning and education by providing multiple opportunities for joint research and development work, including trainings and summer school. Hence, research and innovation projects should strengthen their activities in order to leverage these opportunities for education and skill building in software engineering.

Recommendation Skills-2: Modernising software engineering curricula. The European University system has to be modernised to meet the demands for basic and advanced software engineering skills and competencies. Based on a solid foundation of software engineering principles, software engineering curricula need to address emerging technology trends, including cloud-based, data-intensive, CPS-oriented and/or service-oriented systems. In addition, certification programmes for experts in these areas will help to build a well-trained work-force.

To strengthen fostering innovation, those curricula should also be built from the understanding of the potential of software for creativity and thus learn from and collaborate with social sciences and humanities. One important angle of such curricula may also include generalising work on open source projects during education, constituting a win-win situation for the students and the community.

The long term impact of software relies on educating our future generation(s). In many respects, Europe has been foresighted in taking on ICT as part of programmes, e.g., in both primary and secondary schools. However, an understanding of software goes far beyond goals such as a digitally competent population and digital divide (cf. the Digital Agenda for Europe). Unfortunately, Europe lacks skilled and competent teachers on these levels of education, leading to the following recommendation:

Recommendation Skills-3: Building out software literacy. The European education and school system would significantly benefit from strengthening its ICT training, on all levels and in particular in software. Designing and programming software as a basic skill will profit individuals, businesses/industry and society. In a sense, software literacy must become a key literacy skill to make Europe innovative and competitive and thus serve as a primary foundation for the job creation of this century.

5.3. Best Practices for Software Engineering Research & Innovation

Software's specific characteristics (cf. Section 1.2) need to be considered when planning and executing successful software engineering research and innovation projects. To this end, this section provides a set of best practices from the analysis of past software engineering projects.

To identify best practices for successful software engineering research and innovation projects, NESSI analysed the *public* outcomes and reports of past FP7 projects funded under

Objective 1.2¹, i.e., calls ICT-2007-1.2 (“Software & Services Architectures, Infrastructures and Engineering”), ICT-2009-1.2 (“Internet of Services, Software and Virtualisation”), and ICT-2011-1.2 (“Cloud Computing, Internet of Services and Advanced Software Engineering”).

Not all software engineering projects funded under Objective 1.2 provided a *public* explicit self-assessment and a discussion of lessons learned about their overall approach, strategy and methodology. The ones that did included BIGFOOT, CHOReOS, CLOUD-TM, COMPAS, DEPLOY, PERSIST, Q-IMPRESS, REMCIS, ROMULUS, S-Cube, SEQUOIA, SERENOA, ServiceWeb3.0, SLA@SOI. They have been considered when distilling the below recommendations.

Overall, the analysis of past FP7 projects revealed the following **main best practices of relevance for software engineering projects**. These best practices should thus be considered when soliciting future research and innovation projects on software engineering and related topics, including Cloud, CPS and Big Data.

Recommendation BestPr-1: Perform industry-near research exploiting real-world software engineering cases. Addressing concrete (industry-near) problems clearly facilitates the exploitation and uptake of project outcomes. Real-world use cases (of realistic size and complexity) stimulate problem understanding and evaluation. Rigorous and relevant empirical studies in industry are very important for the future of software engineering. Real world cases and data are crucial for such empirical work, which contributes to understanding the applicability of software engineering techniques methods in practice. In addition, addressing industry-near problems may lead to new research challenges to be addressed in order to overcome the “trench” between theory and practice.

Recommendation BestPr-2: Deliver well-documented, working software tools and pilots to make project outcomes more accessible. As far as possible, projects should develop and ensure sustainability of (pre-industrial) tools to make the project outcomes attractive to industry. In order to foster the understanding of these tools by users, existing technology may serve as basis, and novel features should be introduced progressively as they become available in the project. Ideally, these tools are linked to strategy areas of project partners to foster exploitation. In addition, tool development should be complemented by (virtual) training sessions that make it easy to understand how to work with the project tools. To be accessible for practice, outcomes should be well documented and ideally use well-justified standards for such documentation.

Research publications alone might not be the right way to make the results accessible to practitioners [22]. Thus developing the aforementioned tools, even though it requires significant resources and time, is an important, complementary means to stimulate uptake of results. Often it is not necessary that the code itself is industry quality and still these tools may serve as “pilots” for developing them further into commercial offerings (possibly even as part of innovation actions).

As indicated, implementing many of the above recommendations may require complementing short projects with longer term and adequately sized research and innovation projects (a perspective that is also shared in [3]), which leads to one final recommendation:

¹ See http://cordis.europa.eu/fp7/ict/ssai/projects_en.html

Recommendation BestPr-3: Larger-scale, integrated projects – in addition to small ones – important for software engineering research. The current work programme only foresees funding for small research and innovation projects on software engineering. However, large, integrated projects should be funded in addition to small ones due to the fact that software engineering research and innovation requires concerted efforts of industry and research, as well as needs to consider various angles and aspects at the same time in order to deliver practically relevant and significant solutions. Developing and validating novel software engineering techniques, methods and tools to tackle the cognitive complexity, scale and unprecedented dynamicity of future software systems requires larger-scale and longer-term collaborative efforts.

As an open form of knowledge exchange and management applied to software (ideas, design, code, errors, documentation), open source can improve cooperation and sharing and greatly impact quality, roadmap, standardisation, dissemination of software components or solutions. However, the use of OSS also implies the need for a managing of OSS licences and communities. Especially, this requires a structured approach towards managing the evolution and the complexity of OSS libraries and modules.

Recommendation BestPr-4: If pursuing an open source strategy, it needs to be done early on and by leveraging existing ecosystems. To ensure sustainability and adoption of open source project outcomes, projects have to develop an open source community early on in the projects. Ideally, projects would join an existing open source community (and don't start their own) in order to exploit existing ecosystems. To ensure uptake, projects should strive to remain as API compatible as possible with existing open source implementations and integrate their results into the open source reference implementations. An interesting direction inspired by open source is to develop open models and open interfaces. Open models can serve as basis for standards and thus foster easy access to standards. Open interfaces foster easy implementation by multiple vendors.

Introducing open source to a project requires a learning process. It starts from open source awareness, culture and coordination among project participants, thus it may take some time to develop.

Recommendation BestPr-5: Pursue a systems approach to software delivery. Ideally, technology resulting from software engineering projects should be made available in "packaged" form, i.e., in the form of frameworks, toolboxes or integrated models. To this end, projects should bring their individual solution "components" into a systems perspective in order to foster adoption in practice.

Such integration, of course, requires significant resources, time (i.e., longer duration projects) and strong governance. In turn, this means that these aspects have to be duly planned and considered during project preparation and execution in order to be realistic and successful.

6. Conclusions

As analysed and demonstrated throughout this white paper, software engineering principles, techniques, methods and tools need to evolve and novel ones need to be devised in order to keep up with fast-paced technology and societal changes, thus being able to cope with the new challenges. Due to the growing complexity and multi-disciplinarity, software engineering solutions cannot be devised by companies and research organisations in isolation. Similar to other engineering disciplines, software engineering research and innovation requires **concerted efforts of industry and research** to deliver practically relevant and significant solu-



tions. Novel software engineering techniques, methods and tools must tackle the cognitive complexity, scale and unprecedented dynamicity of future software systems.

In order for Europe to remain competitive, this means that the opportunity and specific needs for such joint research and innovation efforts should be sustained – ideally, even increased. Otherwise, Europe might run the risk of losing the competitive ground on software and software-intensive systems and, as a result, will strongly depend on software technology and skills from non-European countries to a higher degree than advisable.

NESSI considers the software engineering funding available in the current work programme (WP2014-2015 – LEIT/ICT-9) a modest starting point at best. **Software engineering research and innovation programmes need to be strengthened** if Europe wants to meet and leverage the opportunities of future ICT trends.

Authors and Contributors

Paolo Bellavista, CINI

Joe Butler, Intel

Marquart Franz, SIEMENS

Juan Garbajosa, UPM

Yosu Gorroñoigoitia, ATOS

Thomas Hahn, SIEMENS

Tobias Hildenbrand, SAP

Nicole Ignaciuk, paluno – University of Duisburg-Essen

Norbert Kraft, NOKIA

Julie Marguerite, Thales

Andreas Metzger, paluno – University of Duisburg-Essen (editor)

Klaus Pohl, paluno – University of Duisburg-Essen

Klaus-Dieter Platte, SAP / Platte Consult

Valère Robin, Orange

Dumitru (Titi) Roman, SINTEF

Nikos Sarris, ATC

Bjorn Skjellaug, SINTEF

Jesús Angel García Sánchez, INDRA

Arnor Solberg, SINTEF

Hans-Jörg Stotz, SAP

Josef Urban, NOKIA

Jan Patrzalek, SAP

Jennifer Perez, UPM

References

- [1] NESSI, „European Software Strategy,“ NESSI Position Paper, June 2008. [Online]. Available: <http://www.nessi-europe.eu/files/PositionPapers/NESSI%20Position%20Paper%20on%20European%20Software%20Strategy%20.pdf>.
- [2] NESSI, “A Software & Service Perspective on the Future of Cloud in Europe,” NESSI Whitepaper, July 2012. [Online]. Available: http://www.nessi-europe.eu/Files/Private/NESSI_Cloud_WhitePaper.pdf.
- [3] ISTAG, „Software Technologies: The Missing Key Enabling Technology - Toward a Strategic Agenda for Software Technologies in Europe,“ July 2012. [Online]. Available: <http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf>.
- [4] "Playing to Win in the New Software Market: Software 2.0 – Winning for Europe," Report of an Industry Expert Group on a European Software Strategy, June 2009, Version 3.5. [Online]. Available: ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/ssai/European_Software_Strategy.pdf.
- [5] ITEA, ARTEMIS-IA, „High Level Vision 2030,“ 2013. [Online]. Available: <http://www.artemis-ia.eu/publications>.
- [6] A. Arora, L. G. Branstetter and M. Drev, *Going Soft: How the Rise of Software-Based Innovation Led to the Decline of Japan's IT Industry and the Resurgence of Silicon Valley*, MIT Press Journals, July 2013.
- [7] IEEE, *Standard Glossary of Software Engineering Terminology*, 1990.
- [8] E. Di Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou and K. Pohl, “A Journey to Highly Dynamic, Self-adaptive Service-based Applications,” *Automated Software Engineering*, vol. 15, no. 3-4, pp. 313-341, 15(3-4) 2008.
- [9] ECSEL Joint Undertaking, „MultiAnnual Strategic Research and Innovation Agenda (MASRIA 2014),“ 2014. [Online]. Available: http://www.smart-systems-integration.org/public/documents/publications/2014_ecsel_masria_partc.pdf.
- [10] NESSI, “Big Data: A New World of Opportunities,” NESSI Whitepaper, December 2012. [Online]. Available: http://www.nessi-europe.com/Files/Private/NESSI_WhitePaper_BigData.pdf.
- [11] B. NESSI, “DRAFT European Big Data Value Strategic Research & Innovation Agenda, Version 0.7,” Version 0.7, April 2014. [Online]. Available: <http://www.bigdatavalue.eu/index.php/downloads/finish/3-big-data-value/14-big-data-value-strategic-research-and-innovation-agenda/0>.
- [12] TiViT, „Strategic Research Agenda: Need for Speed,“ April 2013. [Online]. Available: http://www.digile.fi/file_attachment/get/SRA_Need4Speed_4.pdf?attachment_id=438.
- [13] A. Metzger and K. Pohl, “Software Product Line Engineering and Variability Management: Research Achievements and Challenges,” in *Future of Software Engineering Track, International Conference on Software Engineering*, Hyderabad, India, June 2014.

- [14] P. Bellavista, G. Cardone, L. Foschini, A. Corradi, C. Borcea, M. Talasila and R. Curtmola, "Fostering Participaction in Smart Cities: a Geo-social Crowdsensing Platform," *IEEE Communications Magazine*, no. Vol. 51, No. 6, pp. 112-119, pp. 112-119, June June 2013.
- [15] T. Brown, *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*, New York: HarperCollins, 2009.
- [16] H. Plattner, A. Back, W. Brenner, R. Jung, H. Österle and R. Winter, *Jumpstarting Scrum with Design Thinking*, St. Gallen: University of St. Gallen - HSG, 2013.
- [17] T. Hildenbrand and J. Meyer, *Intertwining Lean and Design Thinking: Software Product Development - From Empathy to Shipment*, Berlin, Heidelberg: Springer , 2009.
- [18] P. Naur and B. Randell, *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany: Brussels, Scientific Affairs Division, NATO, 1968.*
- [19] W. W. Gibbs, "Software's Chronic Crisis," *Scientific American*, no. pp.72—81, September 1994.
- [20] A. Cockburn, "The End of Software Engineering and the Start of Economic-Cooperative Gaming," *Computer Science and Information Systems*, pp. 1-32, February 2004.
- [21] *Graduate School Directory*, <http://www.gradschools.com/>.
- [22] S. Beecham, P. O'Leary, S. Baker, I. Richardson and J. Noll, "Making Software Engineering Research Relevant," *IEEE Computer*, vol. 47, no. 4, pp. 80-83, 2014.