TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

TECHNISCHE BERICHTE
TECHNICAL REPORTS

ISSN 1430-211X

TUD-FI14-04-August 2014

Dr. Frank J. Furrer, Jan Reimann (Eds.)
Institut für Software- und Multimediatechnik

Impact and Challenges of Software in 2025

Technische Universität Dresden
Fakultät Informatik
D–01062 Dresden
Germany
URL: http://www.inf.tu–dresden.de/

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Hauptseminar Sommersemester 2014

Dr. Frank J. Furrer

Impact and Challenges of Software in 2025

Collected Papers

Editors: Dr. Frank J. Furrer, Jan Reimann

Technische Universität Dresden
Technical Report TUD-FI14-04-August 2014

"Software is the fuel of our modern economy – Where are we going?"

Table of Contents

Introduction

Dr. Frank J. Furrer

Context

Today (2014), software is the key ingredient of most products and services. Software generates innovation and progress in many modern industries. Software is an indispensable element of evolution, of quality of life, and of our future. Software development is (slowly) evolving from a craft to an industrial discipline. Software – and the ability to efficiently produce and evolve high-quality software – is the single most important success factor for many highly competitive industries.

Software technology, development methods and tools, and applications in more and more areas are rapidly evolving. The impact of software in 2025 in nearly all areas of life, work, relationships, culture, and society is expected to be massive.

The question of the future of software is therefore important. However – like all predictions – quite difficult. Some market forces, industrial developments, social needs, and technology trends are visible today. How will they develop and influence the software we will have in 2025?

Topic

This seminar worked on answers to the central question: *Which are the situation, the challenges, and the impact of software in the year 2025?*

The focus lies on 3 relevant areas:

Q1 How will the software in 2025 be different from today's software?

Q2 How will software be engineered, developed, operated, and evolved in 2025?

Q3 What is the impact of software on people, work and society in 2025?

Each participant had to choose one focus and elaborate on one specific theme related to the focus question. The following choices were made:

Name	Q1	Q2	Q3
Bierzynski, Kay	X		
Gollasch, David		X	
Korger, Christina			X
Peschel, Paul		X	
Rausch, Jonas			X
Schön, Hendrik	X		

Objectives

The participants learned:

(a) to do focused research in a specific area,

(b) to author a scientific paper,

(c) to improve their LaTeX expertise,

(d) to experience the peer-review process,

(e) to learn how to deliver effective and efficient presentations,

(f) to benefit from a considerable broading of their perspective in the field of technology, software, applications, and impact.

The participation in the Hauptseminar will enable the participants to plan their personal education and professional evolution.

Mandatory Reading

All participants were asked to read the following 3 references:

- Edgar G. Daylight, Sebastian Nanz (Editors): *Conversations: The Future of Software Engineering – Panel Discussions.* 22-23 November 2010, ETH Zurich. Lonely Scholar bvba, Heverlee, Belgium, 2011. ISBN 978-94-91386-01-5
- U.S. National Academy of Engineering: *The Engineer of 2020: Visions of Engineering in the New Century.* National Academy Press, Washington D.C., USA, 2004. ISBN 978-0-309-09162-6. Downloadable from: `http://www.nap.edu/download.php?record_id=10999`

- ISTAG – Information Society Technologies Advisory Group (Working Group on Software Technologies), July 2012: *Software Technologies – The missing Key Enabling Technology (Toward a Strategic Agenda for Software Technologies in Europe)*. Downloadable from: `http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf`

Papers

The following papers were authored, peer-reviewed and presented during the Hauptseminar. All papers are reproduced in full on the following pages.

Impact of Heterogeneous Processor Architectures and Adaptation Technologies on the Software of 2025

Kay Bierzynski

Institut für Software- und Multimediatechnik, Technische Universität Dresden
D-01062, Dresden, Germany

Abstract. Many products and services, which we are using today, are powered by software. The software has a big impact on our everyday life and became in the last three decades one of the most important drivers of the global economy. The result of this development is that our present-day society extremely relies on software. This dependency will grow in the future, because of that we need to find and understand the challenges and trends, which will influence software in future, so that research and the economy can focus on them.
A way to do so is to answer the question: What is the difference in software in 2025? We will make a contribution to the answer for this question through discussing the influence of heterogeneous processor units on the software and their deployment environments in 2025 in the first part of the paper. In the discussion we will present some reasons for our conclusion of the first part. The conclusion is that we expect that the software of 2025 will be executed on mobile devices from a heterogeneous processor unit and on stationary devices from a central or graphic processing unit. This very likely development rise the need for more and better dynamic adaptation technologies so that software can efficiently use the different processor unit kinds.
In the second part of the paper we discuss reasons beside the efficiently use of processor units for the importance of the self adaptation approach, the autonomic computing approach and the role approach for the software in 2025. But important as autonomic computing may be we expect that autonomic computing as envisioned by IBM [10] will not completely exist in 2025, because of the standardization process, the difficult development of a query language, legal issues and security issues. More information to these are presented in the paper. To conclude both discussions we will give some recommendations.

Keywords: APU, CPU, GPU, heterogeneous processor unit, autonomic computing, self adaptive software, role

1 Introduction

Many products and services, which we are using today, are powered by software. The software has a big impact on our everyday life and became in the last three

decades one of the most important drivers of the global economy. The result of this development is that our present-day society extremely relies on software. This dependency will grow in the future, because of that it is important to find and understand the challenges and trends, which will influence software in future. One possibility to do so is to answer the question: What is the difference in software in 2025? To formulate a complete answer for that question is hard, because software is used in many different areas. Some of them for example are the following: parallel, cloud and high performance computing; big data; social computing; internet-based applications and real-time services; embedded systems; human-centred computing and multi-media; enterprise applications and the future generation of software-intensive systems [3].

It is not possible to analysing and discussing all challenges and trends in the different areas in a single paper, this is the reason why we will focus on two specific topics. The first topic is the impact of heterogeneous processing units on the software in 2025. We will deal with this topic through analysing the question: Which kind of processing units will execute software in 2025? As a basis for the analysis, we will use the results of an experiment, which was executed in my bachelor thesis [1]. After that we present the importance of the self adaptation approach, the autonomic computing approach and the roles approach for the software in 2025 and the reasons for why we think that autonomic computing as envisioned by IBM [10] will not completely exist in 2025.

Based on the two topics the structure of the paper is as follows: In the next section we will talk about existing works. The third section deals with the first topic and the fourth section with the second. In the last section we present our conclusions and recommendations.

2 Existing Work

The two works, which inspired this paper, are the report "Software Technologies The missing Key Enabling Technology (Toward a Strategic Agenda for Software Technologies in Europe)" from the Information Society Technologies Advisory Group [3] short ISTAG and the book "Conversations: The Future of Software Engineering Panel Discussions" from Edgar G. Daylight and Sebastian Nanz [2].

The latter is about two panels, where some of the top researchers in the area of software engineering discuss how the state of the art of software and software engineering is today and how it should be in the future. As already mentioned our paper is inspired by this book so for example we got the idea to look in the area of software parallelism from a statement of Bertrand Meyer in the second panel [2].

The ISTAG propose in their report the creation of a European Software Agenda. To support their proposal they analyse the current strengths and emerging trends in software technology and a number of critical application areas including: parallel, cloud and high performance computing; big data; social computing; internet-based applications and real-time services; embedded systems;

human-centred computing and multi-media; enterprise applications and the future generation of software-intensive systems [3].

This paper is different from these two works to the amount that we specifically focus on the influence of heterogeneous processing units in the software of 2025 and on the importance of the self adaptive software approach, the autonomic computing approach and the role approach for the software in 2025.

For the field of adaptation technologies the paper "The Vision of Autonomic Computing" from Jeffrey O. Kephart and David M. Chess [10] and the white paper "An Architectural Blueprint for Autonomic Computing" [11] are important to mention. In the first paper the vision of autonomic computing is delivered. This term describes computing environments with the ability to manage itself and dynamically adapt to change in accordance with business policies and objectives. These environments can perform such activities based on situations they observe or sense in the IT environment rather than requiring IT professionals to initiate the task (from [11]). The properties such environments beside self-managing should have are listed in the first paper as well. They are self-configuring, self-healing, self-optimizing, and self-protecting. The terms self-configuring and self-optimizing are self-explaining. The term self-healing means that autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware [10]. Self-protection describes the task of a autonomic computing system to defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures and to anticipate problems based on early reports from sensors and take steps to avoid or mitigate them [10]. Beside the basic definition of autonomic computing Kephart et. al. present in their paper [10] the engineering challenges and scientific challenge, which we need to overcome to realize the vision of autonomic computing, as well as the architectural considerations for autonomic systems. An architectural blueprint for autonomic computing system is the result of a work from the IBM Business Consulting Service and is shown in the already mentioned white paper "An Architectural Blueprint for Autonomic Computing" [11].

In this paper, we will deal with autonomic computing through presenting the reasons for why we think that the vision of autonomic computing will be not completely realized in 2025.

3 Impact of Heterogeneous Processor Architectures on the software in 2025

In the first part of this section we describe the terms central processing unit short CPU, graphic processing unit short GPU and accelerated processing unit short APU. After that the experiment from my bachelor thesis[1] is presented. The results of the experiment are used to answer the question which kind of processing units will execute software in 2025.

3.1 Processing Units

Many different kinds of processing units are used in today's products. In this paper, we only look at three of them. These three are the CPUs, the GPUs and the APUs. We choose the CPUs and the GPUs, because they are the most commonly used processing units. The APUs as representatives for heterogeneous processor units are interesting, because they will play an important role in the future of computing especially in the future of mobile computing.

The CPU is the central component of a computer. It monitors and steers the computer. Beside these two tasks the CPU is responsible for carrying out the instructions of a computer program. The memory system of the CPU is called cache.

The GPU is in the computer responsible for calculating the monitor output and other parallel and mathematically-intensive tasks. To deal with such tasks GPUs have many more cores in comparison to CPUs.

The APUs from AMD fall in the category of the heterogeneous processor architectures. These architectures combine a main processor with a coprocessor. The latter is always at least in one area superior to the main processor so that it can support the main processor in specific situations. A common combination of processors is the combination of a CPU as a main processor with a GPU as a coprocessor, because of the already addressed parallel and mathematically-intensive tasks. The combination of processors offers beside the support in specific situations new possibilities for parallel execution of programs.

3.2 The Experiment

In my bachelor thesis [1] the radix sort algorithm was implemented and optimized for CPU, GPU and APU. The APU, which was used, had a CPU as a main processor and GPU as a coprocessor. After the optimization the three resulting variants were tested on their respective processor unit with different data input sizes. Before we look at the results of the tests a short description to the coarse structure of the radix sort algorithm. It consist of the following three steps: create the local histograms, scan and permutation. In the first step the input data is split into several pieces and for every data piece a local histogram is calculated. The local histograms are combined into one global histogram in the process of the scan step. The last step uses the histogram to calculate the position of the data pieces in the sort sequence of the data and puts the pieces at the correct position in the output.

An execution variant of the radix sort is shown in figure 1. In this variant the CPU deals with the first and the last step of the algorithm and the GPU executes the scan step, because of reasons which will follow shortly in this subsection [1]. Before the development of heterogeneous processor units such an variant would be extremely inefficient performance wise due to the slow connection with a small bandwidth between the CPU and the GPU. In my bachelor thesis this variant was implemented, optimized and tested for the APU to see if such variants could be better as variants, which are executed only by one processor kind, in regard

to performance when the CPU and the GPU are more closely connected.

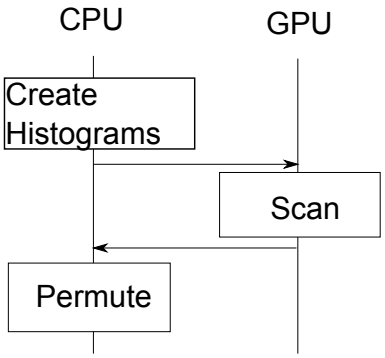

Fig. 1. Radix sort variant, which is partly executed by the CPU and the GPU.

The results of the tests are presented in the figure 2. The basic result of these tests is that starting from a data input size of 64 KB the APU variant is faster as the GPU and CPU variant [1]. This basic result also corresponds to data input sizes ranging from 1MB to 1GB.

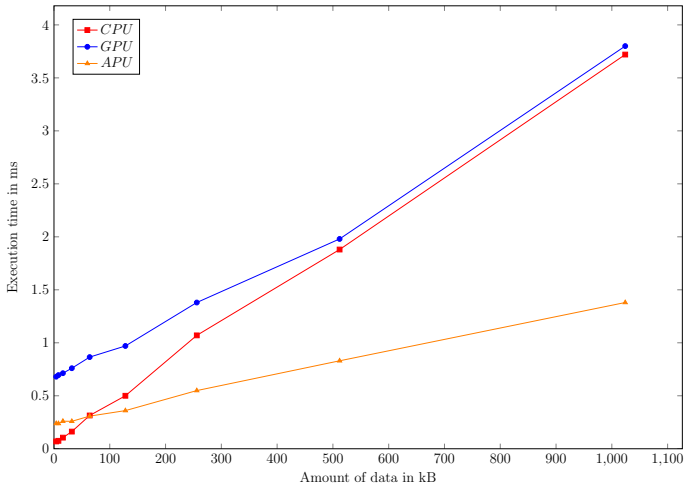

Fig. 2. Diagram with measurements until 1MB

The reason behind this result and the execution, distribution of the steps in the APU variant is the second step of the radix sort variant. This step includes

parallel and mathematically-intensive tasks, which as already described in the subsection 3.1 are faster executed by the GPU as by the CPU. The first and last step in comparison to that are completed faster on the CPU, because through it's cache the CPU is good at loading data pieces and putting the pieces at new positions in the memory. The cache is also the reason why for small input data the CPU variant is the fastest variant, because when the input data fits completely in the cache than the CPU can even execute parallel and mathematically-intensive tasks faster as the GPU [1].

3.3 Which kind of processing units will execute software in 2025?

Which kind of processing units will execute software in 2025? The answer to this question should be APUs when we look at the results presented in the last part, but it is important to mention that the CPU and the GPU used in the experiment didn't belong to the high-end hardware area [1]. The APUs which are on the market today can't compete in the area of performance with high-end graphic cards or CPUs, but they have other benefits.

Some of these benefits are the low production costs and the low energy consumption, which are resulting through the combination of the processors. Another advantage of APUs is the possibility to save space in the products, where an APU is implemented as the processor and the coprocessor of the APU is a GPU. Such products could use the GPU coprocessor as a replacement for a graphic card [6]. These three benefits we listed here are also the reason why APUs are interesting for the mobile computing branch. Since low energy consumption, low costs and low space requirements belong to the most important requirements for mobile computing devices.

Based on these facts we expect for the processor unit usage in 2025 two cases. In the first case the development of APUs and other heterogeneous processors was so successful that CPUs and GPUs are no longer needed and every device has a heterogeneous processor implemented. The other possibility is that processing unit kinds get clear usage domains. So it is very likely that APUs and other heterogeneous processors are primarily implemented in mobile and embedded devices in 2025 and CPUs and GPUs in stationary devices such as servers or desktop computers.

We assume that the second case has a higher plausibility to occur as the first case, because the development of GPUs and CPUs goes on as well. Based on this assumption our answer to the initial question is as follows: In 2025 software on mobile devices will be executed by an APU or another heterogeneous processor unit and software on stationary devices will be processed through a CPU or a GPU.

This distribution of processor unit kinds on specific domains leads to a new challenge for software, because when software applications wants to reach the best possible performance on many different devices than they need to adapt to the implemented processor units and other conditions.

As mentioned in the last subsection 3.2 the APU variant of the radix sort, which was used in the experiment, distributed its components between the main

processor and the coprocessor. This is not the only possibility to implement the execution of software efficient on the APU. Another approach would be to split the input data and let one part processed by the main processor and the other part by the coprocessor [7]. These two approaches can even further refined so for example it is possible to execute components parallel on the APU, when the components not depend on each other. Here we listed just some of the options for executing software, but in 2025 software applications can choose from many more as well.

So the questions which arise is what is the best way to process software on the APU or processors in general? This is hard to answer, because the efficiency of processing depends on many different things and it is clear that humans cannot cope with all of them. The problem of finding the best option gets even bigger, when the software in 2025 is for stationary devices as well. So when we want the best performance, we need software which can automatically and dynamically adapts itself according to the conditions. We can get such software through implementing adaptation technologies. Especially the autonomic computing approach from IBM is interesting for heterogeneous processing units, because in this approach the main processor and the coprocessor would be autonomic elements. Such elements can in a autonomic computing system directly communicate and negotiate with each other. This implies that the main processing unit and the coprocessor can efficiently transfer the data between each other depending on their workload.

The ISTAG addresses this challenge in their report as well. They list some technologies, which could be used to face it. One of these technologies for example, is high-level parallel languages capable of handling heterogeneous many-core processors [3].

4 Adaptation Technologies

In the first part of this section we define the terms, which are role and self adaptive software. After the definitions we discuss the self adaptive approach, the autonomic computing approach and the role approach regarding to the software in 2025 and present the reasons for why we think that the vision of autonomic computing will be not completely realized in 2025.

4.1 Definitions

The definition for self adaptive software was provided in a DARPA Broad Agency Announcement on Self Adaptive Software (BAA-98-12) in December of 1997. The announcement defined self adaptive software as follows:

Definition 1. *Self Adaptive Software evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible. [4]*

Until today, no uniform definition of role exist in software engineering. Based on the works of Reenskaug [8] and Riehle [9] we will define roles for this paper as C. Piechnick et al. [5] described roles in their work:

Definition 2. *A role is a dynamic view or a dynamic service of an object in a specified context, offering the possibility of separation of concerns, interface-structuring, dynamic collaboration description, and access restriction. A role is clearly specified by a role type and can be played and removed from an object at runtime. [5]*

4.2 Self adaptive software, autonomic computing and roles in 2025

At the conclusion of the concluding section we pointed out that it is to await that the software of 2025 needs to adapt to different processor unit kinds in different deployment environment. Beside this challenge many more challenge exists and will arise. Two of these are the problematic of complexity management and dynamic change management. Both of them and the initial challenge of different deployment can be dealt with through adaptation technologies. But today's adaptation technologies are primarily limited to product enhancements, patches within the software development process and other static adaptation methods [5]. This technologies will be not sufficient for these challenges, because they are as already mentioned in the most cases static, lack automation and don't have the ability to adapt themselves.

Hence, when we want software applications, which can for example, adapt to different deployment environments, then we must develop them mostly independent from a hardware platform or operating system and implement technologies through they can automatically and dynamically adapt themselves at deployment or runtime. Such software application would be then belong in a group of self adaptive software. We expect that almost all software applications and systems in 2025 will be self adaptive regarding to at least one of the self-* properties, which were defined as the properties of a autonomic computing system in the paper "The Vision of Autonomic Computing" [10]. But this doesn't mean that we expect that the vision of autonomic computing will be completely realized in 2025, because for the realization of autonomic systems standards are needed. Especially the negotiating process between autonomic elements needs to be standardized, when we want to use autonomic elements from different vendors. The standardization is a problem insofar that the process of developing a standard is slow and we doubt that the standard development processes will be finished until 2025.

Other reasons why we think that complete autonomic systems will not exist in 2025 are the following points: development of a query language to give high level orders to the autonomic system, which the system realizes through deciding and executing low level orders and processes, legal issues and security in general. The foremost reason is difficult in the way that in the best case autonomic systems should be usable by people, which have no IT knowledge and cannot speak English, hence the query language must be intuitive and needs to handle as many

natural languages as possible. The second reason and third reason are problems, because the autonomic system should protect the user against the invasion of his digital privacy and the careless use of private data. To realize the protection of the user laws needs to be defined for what a autonomic system can do without the user's approval, which data should be transferred between the autonomic elements and which data should be send out of the system. We doubt that such laws will be created until 2025 and without such laws the development of the security of autonomic systems is difficult as well, because it is not clear what is legal and what is not. Nevertheless, it is important to focus on the autonomic computing approach, because it will be a great tool to manage complexity and changes of software.

The concept of roles is useful for managing changes of software and dynamical adaptation in general as well, insofar that they can be loaded at runtime. We think roles will be implemented in many software applications, because of their high utility and easy deployment. To get a better understanding of how useful roles are let's see how we can use roles to make the radix sort variant, which was implemented in my bachelor thesis [1], more adaptable.

One option would be to transform the steps of the radix sort in completely independent components. These components than could dynamically switch, depending on the input from the operating system, their configuration and assignment to a specific processor unit kind through the role objects they play. Another possibility is to use roles to transfer dynamically the threads between the main processor and coprocessor depending on their workload, when the radix sort is executed on a heterogeneous processor unit. Threads are the packages in which the processing of the radix sort is divided into by the operating system. Beside these two options it gives many more ways to use roles for adaptation.

5 Conclusion

The initial question for our paper was the following: What is the difference in software in 2025? In the area of adaptation technologies we contributed to the answer of this question insofar that we presented reasons for why we expect that almost all software applications and systems in 2025 will implement automatically and dynamically adaptation technologies through they can adapt themselves to highly diverse requirements and why we think that the vision of autonomic computing [10] will not be realized in 2025. This will be a big step from today's software, because today the implemented adaptation technologies are in most cases static and not automatically. Another contribution to the answer of the initial question we made is that we discussed why it is very likely that in 2025 software on mobile devices will be always executed by a heterogeneous processing unit and on stationary devices from a central or a graphic processing unit. This will give rise to a challenge regarding performance, which can overcome through adaptation technologies, and is different from today insofar that at the moment software is in the most cases executed by central processing units.

Based on the expectations we listed, we recommend the following things. In general, we should focus on adaptation technologies and approaches. To realize the vision of autonomic computing until 2025 we should execute standardization processes in the research and development of autonomic computing and involve lawyers in the process beside focusing on the challenges, which were pointed out in the paper "The Vision of Autonomic Computing" [10]. For the other presented adaptation approach we suggest that the concept of roles should be implemented in all programming languages.

References

1. Kay Bierzynski. Effizientes Sortieren auf heterogenen Prozessoren. 2013. Certification link: http://forschungsinfo.tu-dresden.de/detail/abschlussarbeit/28376.
2. Edgar G. Daylight and Sebastian Nanz. Conversations: The Future of Software Engineering Panel Discussions. 22-23 November 2010.
3. ISTAG Information Society Technologies Advisory Group. Software Technologies The missing Key Enabling Technology (Toward a Strategic Agenda for Software Technologies in Europe). July 2012. Downloadable from: http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf.
4. Robert Laddaga and Paul Robertson. Self Adaptive Software: A Position Paper. June 2004. Downloadable from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.5472.
5. Christian Piechnick, Sebastian Richly, Sebastian Götz, Claas Wilke and Uwe Amann. Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation - Smart Application Grids. May 2012. Downloadable from: http://www.thinkmind.org/index.php?view=article&articleid=adaptive_2012_5_10_50066.
6. Benedict Gaster, Lee Howes, David Kaeli, Perhaad Mistry and Dana Schaa. Heterogeneous Computing with OpenCL. Morgan Kaufmann, August 2011. Downloadable from: http://www.sciencedirect.com/science/book/9780123877666
7. Michael Christopher Delorme. Parallel Sorting on the Heterogeneous AMD Fusion Accelerated Processing Unit. 2013. Downloadable from: https://tspace.library.utoronto.ca/handle/1807/35116.
8. T. Reenskaug, P. Wold, and O. A. Lehne. Working with objects - the OOram software engineering method. Manning, 1996. Downloadable from: http://heim.ifi.uio.no/~trygver/1996/book/WorkingWithObjects.pdf.
9. D. Riehle and T. Gross. Role model based framework design and integration. In Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOPSLA 98, pages 117133, New York, NY, USA, 1998. ACM. Downloadable from: http://dl.acm.org/citation.cfm?id=286951.
10. Jeffrey O. Kephart, David M. Chess. The Vision of Autonomic Computing. IEEE Computer Society, New York, January 2003, pp. 41-50. Downloadable from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1160055.
11. IBM Business Consulting Services. An Architectural Blueprint for Autonomic Computing. IBM Autonomic Computing, 4th edition, June 2006. Downloadable from: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf.

Facing Future Software Engineering Challenges by Means of Software Product Lines

David Gollasch

TU Dresden, Faculty of Computer Science, Institute of Software- and Multimedia-Technology, Chair for Software Technology
david.gollasch@tu-dresden.de

Abstract. As software requirements change very rapidly, software engineering principles have to keep up with these changes. This paper names significant trends in the field of software engineering and discusses ways to cope with these trends in the future. The software product line approach seems to be promising here to have a strong influence in the future on larger software products where it's affordable to apply the product line approach. While it currently allows abstraction of variability in space, it will support variability in time as well in the near future. This paper discusses in how far this approach can cope with the former identified upcoming trends.

Keywords: software engineering, future trends, software product lines, software evolution, hyper feature models, variability

1 Introduction

Software engineering is the field of creating and maintaining software products. As the requirements and expectations on software are subject to change very rapidly over time, this has a strong influence on methods, tools and processes in the software engineering field. Hence, it is meaningful to discuss upcoming trends in this area in order to find feasible development approaches.

One promising approach to cope with the rising challenges is software product line engineering. Software product lines allow efficiently developing complex software products by applying reuse principles and introduce a high level of abstraction at design time. It is the goal of this paper to discuss the product line approach in context of its future-awareness. Therefore, the concept of hyper feature models will be discussed.

To obtain this goal, section 2 of this paper discusses currently observable trends in the area of software engineering according to their influence in the future. This leads to new challenges in the future that new software engineering approaches have to face. As software product lines appear to be promising, section 3 presents the underlying principles and aspects to address the challenges. The concept of hyper feature models as current state-of-the-art findings is presented as well, as this is highly interesting for the future-readiness of the

presented development approach. Section 4 sets the product line approach of section 3 including the hyper feature model approach against the discussed trends of section 2. Finally, section 5 summarizes the findings and gives a short outlook for further investigation.

2 Upcoming Trends in Software Engineering

There are some observable trends in today's field of software engineering. Boehm identified ten of them in his findings [1]. This paper focusses on a selection of five out of them that I assume to be relevant for the further discussion.

A more rapid development and adaptability. As software requirements change increasingly fast, the development has to keep up. Therefor, future development strategies have to enable a more efficient way of composing software and adapt to changing requirements.

More software criticality and need for assurance. Software is becoming more and more ubiquitous as it is more easy to adapt software to new requirements than hardware. This leads to the need of a higher software quality. Reliable software is indispensable in some fields, e.g. logistics and medicine.

Increasing complexity and need for scalability and interoperability. Due to the rising importance of the internet and networks in general, software that interacts with other systems is increasingly required. That leads to a growth in complexity and to software that can be scaled.

A need to accommodate commercial off-the-shelf and legacy software. The interoperability between a product and existing COTS (commercial off-the-shelf) or legacy software that is already in use on the customer's side gains in importance as well.

More emphasis on users and end value. Technical borders that developers have to cope with are becoming less relevant in software engineering. That more and more moves the user and cost-benefits into the focus. Future software development strategies should support this trend by means of enabling the composition of software that fits the user's requirements as exactly as possible.

2.1 Coping With These Trends

The above-mentioned trends are on a less technical level, but rather on a relatively abstract one.

Generally, it is the developer's goal to create software with the right set of features, that can be developed efficiently and that can evolve easily. This

implies that coping with these trends is only possible if software engineering can be done on a more abstract level [6]. Higher abstraction allows developers to focus rather on the conceptual aspects of a software than on the technical ones. This is possible as abstraction means the reduction of the concrete software to a more general conception.

A common way to introduce additional levels of abstraction is the use of models. In consequence, a model based engineering approach can help to cope with the mentioned trends. A low level of abstraction means less generalization. A typical low abstraction level of a software is a complete structural model of the code in form of a UML class diagram. A higher level of abstraction can be a component model or even the representation of features in feature models.

3 Software Product Lines and Feature Modeling

Software product line engineering allows the efficient development of relatively similar software products due to capturing commonalities and variable functionality. This allows an intense level of reusing software components in multiple products that can be derived out of one product line.

To illustrate the principle of software product lines it is feasible to find a relation to building blocks. There are two development life cycles: the *Domain Engineering* life cycle and the *Application Engineering* life cycle [7]. The main purpose of the first life cycle is to build up the actual product line which includes the development of all relevant parts or aspects of the later to be derived products. These parts and aspects can be interpreted as a set of different building blocks. During the second life cycle the actual product will be derived which can be seen as the process of building a specific object out of the former prepared set of building blocks. Blocks that are part of every derived object define *commonalities*, blocks that are not mandatory for each object define *variability*.

The model-based product line approach allows the mentioned level of abstraction to cope with the challenges stated in section 2. That makes it appear to be a very promising approach [2].

Feature models are used to represent product lines through a set of features and dependencies between those features. Features can map to structural and behavioral (functional) and/or qualitative and characteristic (non-functional) aspects of a software system [5].

A concrete product can be derived out of a product line through a configuration process. This process encompasses the selection of features included in the product line while respecting existing constraints.

3.1 Variability in Space

Software product lines allow multiple levels of abstraction as features in feature models can represent functional and non-functional aspects of a software. So the levels of abstraction can be determined as follows (according to [5]):

1. structural level (representation of functional aspects of a software)

2. (user-visible) behavioral level
3. qualitative/non-functional level
4. software-system's characteristic level

These levels of abstraction have the structure of resulting configurations in mind. These variable aspects are also called "variability in space". This can be assumed as state-of-the-art.

3.2 Variability in Time

Current product lines only support variability in space. There is a lack of supporting evolutionary processes. This is important in terms of the support for future-ready software development. Respecting software evolution in product lines is called "variability in time" which introduces the time abstraction level.

Variability in time enhances reuse in product lines which makes it a necessity when it comes to future-ready software development.

Elsner et al. [3] identify three types of variability in time:

1. the linear change over time (evolution and maintenance)
2. supporting multiple versions at one point in time (as part of a configuration)
3. binding over time where different types of variability are present at different stages of development

Referring to the second type of variability in time, the support of multiple versions in configurations, there already is one approach to face this. Seidl at al. [8] propose the use of hyper feature models. They describe an extension of feature models in a way that supports maintaining multiple versions of a feature in a product line.

The field of supporting evolutionary aspects of product lines is the objective of current and ongoing research. In terms of coping with upcoming trends in software engineering, this could be a key aspect in making the software product line approach a feasible solution and having a massive impact in future software development processes as stated by Boehm [1].

3.3 Hyper Feature Models

One approach to include version-awareness into the software product line approach is the *hyper feature model* approach introduced by Seidl et al. [8] as an extension to "normal" feature models introduced by Kang et al. [5].

A feature model represents the *commonalities* and *variabilities* of all software products of a product line. Therefor, it includes a set of features and the dependencies between them.

A feature model is structured as an acyclic graph respectively a hierarchical tree structure. So each feature (except the root feature) has one parent feature and can have multiple sub-features. A concrete product of the product line is called *configuration* and describes a valid subset of the product line's set of

features. One rule that ensures the validity of a configuration is that the selection of a feature automatically leads to the selection of its parent feature.

Next to the hierarchical structure, there are cross-tree-constraints. These constraints define additional rules for the validity of configurations. They represent rules like "If feature A is part of the configuration, feature B has to be selected as well" or "Feature A and B cannot be selected at the same time in one configuration".

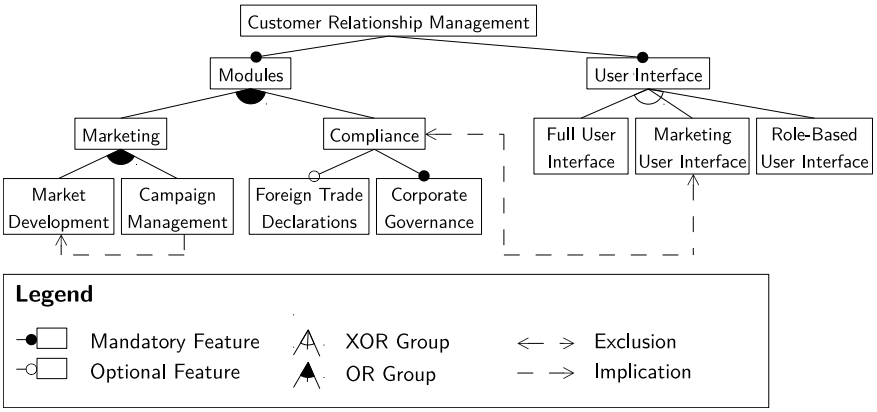

Fig. 1. Example of a feature model based on [4, p. 10]

Figure 1 shows a small example of a simple feature model with all essential elements of such models. The illustration shows the hierarchical structure of features, beginning with the root called *Customer Relationship Management*. During the configuration process, features have to be selected from top to bottom, starting with the root feature and respecting the given dependencies.

Hyper feature models additionally allow maintaining multiple versions of each feature into the model, including the representation of development branches (so each version can have a predecessor and one or multiple successors). The available cross-tree-constraints are extended by version requirements, e.g. "If version 2.0 and above of feature A is part of the configuration, feature B has to be selected in version 1.1 and above as well." Figure 2 extends the example of Figure 1 to an hyper feature model. Each feature has been extended by multiple versions and development branches. Version-aware cross-tree-constraints are not represented in this example.

3.4 Risks, Challenges and Limits of Software Product Lines

The software product line approach is rather complex and needs strong governance. Model inconsistencies directly lead to errors while generating valid configurations and concrete software products. Each change in code and model has to be reflected on each other.

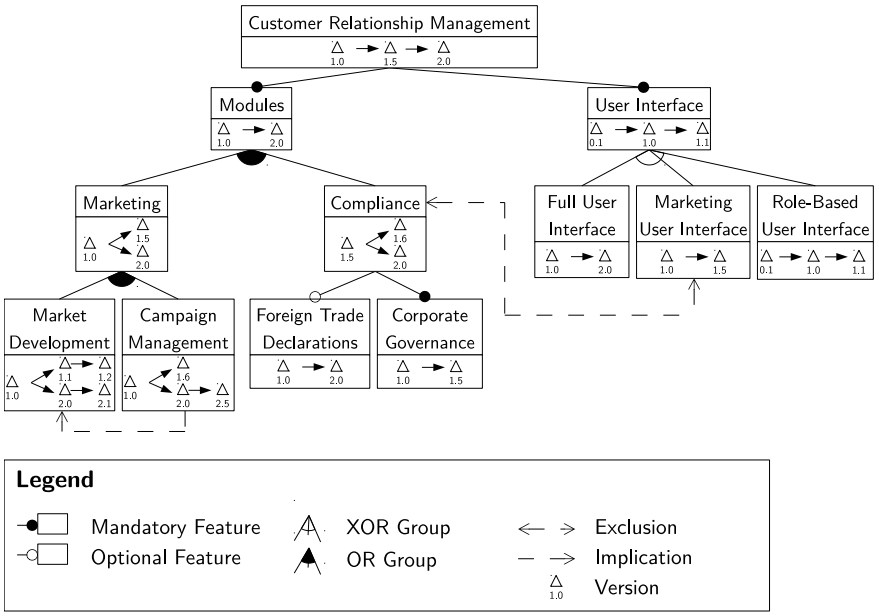

Fig. 2. Example of an hyper feature model based on Figure 1 and the notation proposal of Seidl et al. [8]

That makes is costly and inefficient if only one concrete product should be developed. The software product line approach is only affordable and saves time if multiple products should be generated out of one product line.

Nevertheless, the link between source code and the features in the model is fragile and needs a careful maintenance to ensure the seamless generation of concrete, running software products. The risk of generating redundancy and architecture erosion should be kept in mind.

4 Facing Upcoming Trends Through Software Product Lines

Hyper feature models are only a first attempt to introduce a time abstraction level in software product line engineering. But if we assume that software product lines support variability in space and time, it may cope with the upcoming trends as follows.

According to the trend of a *rapid development*, the strong use of reusing principles helps facing this trend. As a software product line and ecosystem can become very complex due to plenty of dependencies, version-awareness simplifies reusing heavily. However, the product line approach is only appropriate when it is the attempt to generate multiple products out of a product line. That makes it inappropriate for a lot of other software development projects where only one single piece of software is the desired result.

To address the *need for assurance*, composing software out of reliable and proven components can face this trend. As those components are never newly implemented, a version-aware product line will be important as well.

Software product lines cope with *complexity* due to the reuse principle. I do not see a direct advantage in the additional variability in time.

The *need to accommodate COTS* makes it important to keep the own software compatible. As there is no way to influence the life cycle of the third-party software component, version-awareness is fundamental.

The product line approach *copes with user-specific configurations* due to the general variability principle. I, again, do not see a direct advantage in the additional time abstraction.

5 Conclusion

Software in the future tends to get more and more complex and user-oriented while there is a need to get them developed faster without a loss in quality. These trends are a major challenge for software engineers in the future.

Abstraction is one possible key to success. The software product line approach as a model-driven development strategy already allows abstraction on the structural, behavioral, qualitative and characteristic level in form of variability in space.

Variability in time as a time abstraction level will be necessary to cope with the mentioned trends in the future considering evolutionary processes. One approach is the use of hyper feature models as they allow maintaining multiple versions of each feature in one product line.

This makes the software product line approach very promising to be the appropriate development methodology for many software projects in the future. As this approach is very costly to apply, it is only affordable if multiple software products should be generated out of one single product line. Otherwise, the overall development process needs too much time and is rather inefficient.

To get over this drawback, it would be interesting in how far it is possible to combine agile development strategies with the presented software product line approach. If there is a feasible solution, it would be potentially possible to combine the advantages of both worlds and get over the disadvantages mentioned.

References

[1] Barry Boehm. “Some Future Software Engineering Opportunities and Challenges”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 1–32.

[2] Manfred Broy. “Seamless Method- and Model-based Software and Systems Engineering”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 33–47.

[3] Christoph Elsner, Goetz Botterweck, Daniel Lohmann, and Wolfgang Schroeder-Preikschat. “Variability in Time - Product Line Variability and Evolution Revised”. In: *Fourth International Workshop on Variability Modelling of Software-intensive Systems*. VaMoS. Vol. 10. Linz, 2011, pp. 131–137.

[4] David Gollasch. “Qualitaetssicherung mittels Feature-Modellen”. Bachelor Thesis. Dresden: TU Dresden, 2013.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[6] K.RustanM. Leino. “Tools and Behavioral Abstraction: A Direction for Software Engineering”. In: *The Future of Software Engineering*. Ed. by Sebastian Nanz. Springer Berlin Heidelberg, 2011, pp. 115–124.

[7] Frank van der Linden. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. In collab. with Klaus Schmid and Eelco Rommes. Berlin ; New York: Springer, 2007. 333 pp.

[8] Christoph Seidl, Ina Schaefer, and Uwe Assmann. “Capturing variability in space and time with hyper feature models”. In: ACM Press, 2014, pp. 1–8.

Capabilities of Digital Search and Impact on Work and Life in 2025

Christina Korger

Technische Universität Dresden, Fakultät Informatik, Dresden, Germany
christina.korger@tu-dresden.de
http://inf.tu-dresden.de

Abstract. As the amount of digital data is increasing at an accelerating pace, it becomes a central requirement for digital search systems to maintain accessibility. This paper builds on the distinction of different search goals and examines respective search system designs in research and practice to support predictions of impact on work and life in 2025. Major trends such as diverse and more natural search user interfaces, personalization of results or modern forms of human contribution, e.g. human question-answering and social rankings influence the working world and personal life. Beside positive effects in the form of increased support and productivity, several risks are identified. These include increasing dependence on software, loss of credibility of search system providers and information and misuse of personal information. Possible solutions how these challenges can be met are discussed in the remainder. As opposed to previous studies on either search system design or impacts of software, this paper aims to provide a contribution by illustrating the relationship between both.

Keywords: Software in 2025, Digital Search, Cultural Impact, Economic Impact, Information Retrieval, Search Systems, Search Interfaces

1 Introduction

Increasing amounts of digital data assign great responsibility on search interfaces and information retrieval. In order to assess the impact of advances in search on work and life, it is vital to have an overview of state-of-the-art research. This comprises three different areas: the distinction of a user's initial search goal, types of interaction provided in search systems and promising approaches. As a result of the analysis possible and emerging impacts on work and life can be derived from a critical point of view.

A good starting point for research about the impact of software offers "The Software Society" by William Meisel [11]. Brynjolfsson and McAfee provide further insight into impacts on the working world in their work "Race Against The Machine" [3]. For state of the art there should be mentioned Marti Hearst's book on search user interfaces [9], which provides valuable information and references to important research in the specific field of search interfaces.

The quality of a search system is evaluated by the ability to meet a user's search goal. Section 2 distinguishes information needs and categorizes them into three types which require substantially different ways of searching. As a consequence, search systems have to address each type individually. The search interface is essential to the user experience. Section 3 provides an overview of interface design possibilities. Multiple approaches towards better accessibility and relevance of results can be observed from state-of-the-art research. Section 4 covers different search approaches such as vertical, personalized, social and semantic search and gives examples of representative models. As search systems evolve, the relation between humans and computers has to be reconsidered. Section 5 aims to answer the question of what positive and negative impacts on work and life can be derived from upcoming advances in information retrieval and better adaptation to user needs. Section 6 concludes with a summary and a personal reflection on technical evolution in search and impacts in 2025.

2 Search Goals

In a narrow sense the only query correctly answered by a basic web search engine can be a navigational query, signifying someone uses a search engine instead of the address bar to navigate to a specific web site. Another recognized intent is the satisfaction of a transactional need [2]. The user wants to acquire a physical or digital product, e.g. shop for a new camera or download a song, or maybe exchange messages in a chat room. But the original intent behind a search is to satisfy an information need. This implies two things: Firstly, the return of a ranked list of web links is not the shortest way to success. Secondly, the options of query specification are not necessarily sufficient to express an information need. To overcome that distance, users and especially their information needs must be identified and thoroughly understood. On this basis search systems can be adapted accordingly. With some simplifications, information needs can be grouped into three categories: simple fact search, complex problem solving and explorative search.

Many queries are about a specific fact. This can be the size of a building, the name of an actor, today's weather conditions or the price of a product converted to a different currency. The answers are short and mostly unambiguous. Another type of information needs addresses complex problem solving. As well as this may be approached using varying techniques, it requires extensive information and context. Relevant results do not accumulate to a single truth. A correct set of answers would be uninfluenced and complete or at a minimum representative in terms of composition. A third information need has been formulated as information exploration by Waterworth and Chignell [16]. There is no explicit question, thus the underlying goal is not a direct answer but an overview of available information and help in concretizing the question.

3 Search System Interaction

On the one hand, the above performed categorization demonstrates the requirement for search systems of dynamic adaptability to individual needs. On the other hand, it has to provide interaction possibilities guaranteeing simple and expedient use. This section covers different options of query specification and the presentation and preparation of results.

3.1 Query Specification

The standard model of query specification in web search consists of keyword search in text form. Content-specific search systems like digital libraries additionally support structural filtering. Compared to the most natural means of information exchange, oral communication, this may appear inflexible, too restricted in terms of expressiveness or just too much effort to convert to from the original thought.

Advances in speech recognition and natural language interpretation stimulate attempts to offer alternative specification mechanisms. Especially mobile search applications rely on interaction types more suited to a small screen.

As a second approach, [10], [17] investigate the utility of content-specific input, based on the assumption that a query of the same content type as the sought answer can be specified more easily or accurately. For instance, it may be easier to define the rough appearance of an image by sketching it than by describing it with words. This works particularly well if similarities can be better computed on the same content type, as well as when the information seeker has a specific object in mind.

3.2 Presentation of Results

Similar to the different interaction possibilities of query initiation, results can be presented and prepared in alternative ways. As well as web search implies the conventional return of a ranked list of potentially relevant page links (enriched with relevant meta information such as titles and snippets), engines such as Google[1] currently feature a variety of output generation.

With the same query, a web search, an image search, a video search or a map search can be conducted. Specific search for news, books, flights or apps is possible, corresponding to the heterogeneous pool of domain-specific webpages.

Even some queries not covered by one of these search extensions are partly being answered in additional boxes next to the list of web links. For example *"100 pound to euro"*, *"cinema"* or *"weather"* are presented "direct-to-content" [11], whereas the query *"Robin Hood"* leads to a summarized presentation of images, the respective Wikipedia article and a suggested disambiguation (if the search intent was to retrieve information on the film, not the legend).

[1] `https://www.google.com`

In addition to text-dominated presentation there are several approaches of result visualization, for instance thumbnails of each document i.e. page previews in case of web search. Further possible visualizations include display of computed graphics about relations between topical categorization, temporal correlations or tag clouds. Dörk et al. [6] provide a detailed investigation of the concept of visual exploration. While the response time-benefit ratio currently does not always justify the additional effort, this problem could vanish in the future due to increased performance of both hardware and software.

Opposed to visual output, a query can also be answered in oral form. Together with natural language input this offers the particular advantage of a hands- and eyes-free operation.

Leading further, a search system can be just the entry point to an inquiry, automatically forwarding to a respective system which provides tailored output in arbitrary form. Meisel [11] elaborates on a similar vision with the following words: "While it isn't a current feature as this is written, it would be a natural evolution if one personal assistant application could be supported by another personal assistant application through an appropriate transfer of control. [...] If requested from a general PA or another specialized PA, e.g., by saying 'XYZ Company customer service,' the active PA can launch the company customer service PA." An explicit example illustrating this case would be talking to your personal assistant about getting directions to some place and having this inquiry forwarded to your navigation system, immediately starting operation.

4 Search Approaches

While some of the mentioned interaction models promise a more natural usage of search tools, others may prove a minor contribution to usability in future search systems. This section describes different approaches from state-of-the-art research, user needs they address and interaction models implemented.

4.1 Vertical Search

One field of research in information retrieval concentrates on the vertical search approach. Vertical search engines as opposed to serving a general purpose focus on a specific content segment. This can be defined by content type, domain, target audience, location or enterprise.[2] Approaches of this kind are driven by the hypothesis that they can provide higher precision of results and targeted support of user tasks. While the starting point of most searches on the web might be a general search engine, many continue to a vertical site. Looking for a new desktop background may involve a visit to `flickr.com`. Inspiration of what to cook tomorrow may be found on `allrecipes.com`. Shopping may be pursued on `amazon.com`. Places nearby can easily be explored from mobile phones via location-based applications like Yelp[3] and various others. Last, but

[2] cf. SEO Term Glossary: `http://seotermglossary.com/vertical-search-engine/`
[3] `http://www.yelp.com/`

not least, enterprise search solutions combining company-relevant information from multiple internal and external sources are good business. The idea has been successfully implemented in practice but is still gaining importance.

4.2 Personalized Search

A second field of research is represented by personalized search. The aim is to return results tailored to the individual by incorporating context information. The simplest form of context information is daytime and location. For example, a query for *"cinema"* conducted on Google immediately returns a list of movies running at cinemas nearby and a map of where they are located. Additional information can be extracted directly from user profiles or indirectly from logging search behavior. Sohn et al. [14] observe particular benefits for mobile search applications with 72% of the information needs surveyed being classified as contextual.

4.3 Social Search

Another promising field of research in information retrieval is social search. According to Hearst [9], social search can be divided into three manifestations: social ranking, collaborative search and human question-answering. Commencing with the first, many search systems are being supported by constant user feedback. Results are computed based on user ratings, letting subsequent users profit of the so-called collective intelligence. This mechanism can also be applied as an additional sort or filter option as known from e-commerce platforms such as `amazon.com` or `hostelworld.com`.

Collaborative search, on the other hand, originates from findings that in the context of work solving complex search tasks in cooperation is common practice [1]. The standard collaboration model deploying an ordinary search system is one person interacting directly with the computer, the remaining participants influencing the process by making suggestions to the former. Research mainly addresses this problem at the interface level. Models range from display synchronization to the implementation of notification mechanisms as a replacement for physical presence and oral communication or with the intent to increase productivity. Pickens et al. [13] introduce a system with separate roles and interfaces for exploration and evaluation, facilitating synchronized mutual influence while protecting workflows from interruption.

The third manifestation of social search is human question-answering. The term represents the model of already well-established Q&A sites (short for "questions and answers") such as `ask.com` and associates of the Stack Exchange Network[4]. Users can post a question and suggest answers to open questions in reverse, assigning part of the search engine task back to humans. Consequently, users can conduct a search for terms occurring either in the question, i.e. someone else's query, or in the answer. As the questions provide a more detailed

[4] `http://www.stackexchange.com`

description of an information need, it enables better judgement of relevance to the seeker and is often accompanied by a more tailored answer than computation could facilitate.

4.4 Semantic Search

One more field of research to be considered is committed to semantic search. The idea behind the concept is to improve recognition of user intents by applying semantical analysis and interpretation to the query. This can either be achieved similarly to Google's Knowledge Graph[5] by statistic evaluation, or based on ontologies which map knowledge to a hierarchy of concepts enriched with types, properties and inter-entity relationships. Large ontologies exist, especially for specific domains such as Gene Ontology for the biomedical sector, forming the basis for semantic search engines.

For general purpose search engines, semantic search is the foundation of many *direct-to-content* result presentations, including elements like "related search" or discussed integration of Wikipedia references. Further subsets of semantic search identified by Grimes [8] are faceted search, clustered search and natural language search. The former two relate to query classification into facets respectively predefined or extracted from the content of results. Examples for predefined facets are topic, task, spatial and time sensitivity and can be applied either individually or simultaneously (cf. González-Caro and Baeza-Yates [7]). A representative of the latter is the computational knowledge engine WolframAlpha[6] which states the goal "to make all systematic knowledge immediately computable and accessible to everyone"[7]. It accepts free-form input and directly generates answers from a curated knowledge base, combining natural language search with the *direct-to-content* approach. Another type of application offering a complete natural language interface is the mobile personal assistant, among them Apple's Siri[8] which also uses WolframAlpha as a service.

5 Impact of Digital Search

Preceding sections have given an outline of the state of the art and trends in search system design. This section concentrates on the derivation of possible positive and negative impacts in 2025. Recent developments and state-of-the-art research imply a trend towards ubiquitous and fast access to a large base of knowledge, either through general purpose systems with rich interaction features or through specialized systems such as domain-specific search engines and intranets.

In both private and professional life this will lead to increasing use - and thus to dependence on software. One critical aspect in dealing with digital information

[5] http://www.google.com/insidesearch/features/search/knowledge.html
[6] http://www.wolframalpha.com
[7] http://www.wolframalpha.com/about.html
[8] https://www.apple.com/ios/siri/

distributed over the web is trustworthiness. To serve as a reliable source, the origin of any information must be traceable. The retrieval of information should best be transparent to the user. Neither should information be withhold, nor manipulated. If this issue is ignored, it will not be possible to guarantee full reliability. For this reason there has to be some kind of independent control structure ensuring the objectivity of digital search engines and providers.

Another common problem to be managed is possible misuse of personal information. Ubiquitous access to information creates great value, but the individuum always needs to be protected. In this regard, improving personal decision-making powers over what information to share is an important step forward.

5.1 Impact on Work

Beside general aspects, explicit effects on the business world and professional life can be derived. Brynjolfsson and McAfee [3] emphasize the economic impact of technological innovation by calling it the "third industrial revolution".

One incipient impact is the shift of task fields. While long-term prospects may decline in certain fields, jobs may emerge in others. As outlined in Section 3 on interactions, natural language processing and speech recognition do and will experience significant progress. For economy this implies an automation of services. Call center or help desk agents, but also general customer services in automated form are already being established[9] and are likely to replace jobs little by little.

Another impact of more efficient search and accessibility of information may be a decreasing need for human consultants. Library reference staff or booksellers are going to become redundant as digital search, neither tied to time nor location, can offer equal advice or aggregated recommendations.

On the one hand, the replacement of human labour with software offers cost savings and independence for enterprises. Customers may value constant performance and the convenience of unlimited access. On the other hand, there are reasons why some people still prefer human interaction. While digital software agents are no longer built only on mechanisms as simple as yes/no navigation or a predefined topic selection, it still seems a long way to go until software will be able to react almost freely or even creatively to human inquiries.

Where creativity is part of the solution, retrieval of information often is insufficient. Here, human and computer abilities may prove an excellent complement to each other and highly increase productivity. For example a software consultant looks up facts and solutions to single, isolated problems but can much more efficiently combine and adapt everything to an actual more complex problem with various dependencies. Meisel [11] expresses this very well as the computer providing an "extension to human capabilities". Supporting developments in search are models of personal assistants, quoted collaborative search, general relevance

[9] cf. IBM Watson for businesses as an example of software services building on automated search mechanisms: `http://www.ibm.com/smarterplanet/us/en/ibmwatson/`

improvements by better retrieval and preparation, and highly esteemed intranet solutions.

Jobs with an especial requirement of creativity directly emerging are conceivable in the field of digital content evaluation and organization, supervision of knowledge contribution and in the field of regulations and protection mechanisms handling privacy and security.

On the other hand, acceleration of developments particularly driven by advances in information retrieval might encompass a dynamization of the economic sector. This can mean an advantage for start-ups and everyone with entrepreneurial spirit and a good business plan. Brynjolfsson and McAfee [3] lend further support to this idea, claiming "enormous opportunities for individuals to use their unique and dispersed knowledge for the benefit of the whole economy".

5.2 Impact on Life

Just like the working world, also people's everyday life and society as a whole will be significantly influenced by developments and new approaches in search systems design. As knowledge becomes more easily acquirable and interaction more natural, improved user experience and increasing use become apparent also in this area of life.

Increasing use will widely release people from memorizing tasks as access to digital information provides immediate reference to build on. This may have consequences extending as far as to education, laying the foundation of both more logic-centered and creative thinking [11].

Whereas findings of Church et al. [5] suggest that mobile search is often deployed during social activities, interaction with surrounding people may otherwise decrease as information needs are saturated by a personal search assistant. Examples are navigation, personalized recommendation as well as facts, with search systems likely to provide more reliability and objectivity or respectively sensitivity to personal preferences. As permanent reassurance is guaranteed, activities such as 'eat at a restaurant not recommended by your personal assistant and without checking the menu in advance and making a reservation' for some people may become close to an adventure.

Cerf [4] provides personal assessment on the threat of personification of software systems[10] accompanying human-like behavior such as natural language interfaces. Although Cerf classifies this not as an immediate danger, he concludes that "users might assume intelligence beyond the system's capacity" [4] and sounds a note of caution for the designers of stated intelligent systems.

One concern voiced specifically on the trend towards personalization is that the user could perceive a distorted image of the world: "The new generation of Internet filters looks at the things you seem to like the actual things you've done, or the things people like you like [...], constantly creating and refining a theory of who you are and what you'll do and want next. Together, these engines

[10] based on [15]

create a unique universe of information for each of us [...] which fundamentally alters the way we encounter ideas and information." [12]

6 Conclusion

In the course of this work trends in the design of search systems, able to address different user goals, have been analysed. Identified as prevailing has been ubiquitous access, interfaces facilitating more diverse and especially natural interaction, and consideration of personal and semantic context for improved relevance and presentation of results. Further developments have been emphasized relating to explicit human contribution in the form of human question-answering and social rankings. These serve as an additional foundation for analysis and sensemaking of data. On the other hand, it has been shown in which ways work and life may be affected by this forthcoming evolution. Main connections can be seen: firstly, in the personal search assistant model and increasing use and dependence; secondly, in the advanced integration into the working environment and enhanced productivity; and finally, specifically in the advances in natural language processing, speech recognition and semantic analysis as a stimulator of automation of services.

My perception is that evolution of search system design has high potential to complement work and life. I am confident that we will master to keep pace with accelerating changes as we do not compete but collaborate with technology. Although in some fields digital search applications will supersede humans as they offer equal or better performance at lower cost, at the same time this means people are able to engage in other tasks. Fields where human strengths exceed those of search systems are where information needs have yet to be elaborated, i.e. in mediation, or where the desire for technical depth has to be estimated carefully to satisfy the information need but not to overwhelm. Apart from this, even though a personal search assistant may provide objective, reliable answers and simple interaction, human interaction may still be appreciated as more engaging and inspiring more confidence through allowing personal judgement.

Whoever controls access to information has great power. It is therefore vital for the reliability of search systems that information about who is responsible for the data and about the principles underlying result computation is transparent and comprehensible to some extent. From my point of view personalized results are never desirable unless explicitly stated. The latter for example is conceivable in the music domain. Separated from access to unbiased search and results the benefits of finding "what people like who like what I like" can consciously be enjoyed. Caution also is advised in terms of distinction between human intelligence and simulation thereof. Whereas natural interaction is important for better accessibility, it should never lead to the misconception that behind this interaction might be more than hardware and code.

In conclusion, while demanding careful attention, evolution of search systems should be perceived as a vital and valuable contribution to both work and life. The ultimate shape is residing in our hands.

References

1. Amershi, S., Morris, M.R.: Cosearch: a system for co-located collaborative web search. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 1647–1656. ACM, New York (2008)
2. Broder, A.: A taxonomy of web search. SIGIR Forum 36(2), 3–10 (Sep 2002)
3. Brynjolfsson, E., McAfee, A.: Race against the machine: How the digital revolution is accelerating innovation, driving productivity, and irreversibly transforming employment and the economy (2012)
4. Cerf, V.G.: Sherry turkle: Alone together. Internet Computing, IEEE 15(2), 95–96 (March 2011)
5. Church, K., Oliver, N.: Understanding mobile web and mobile search use in today's dynamic mobile landscape. In: Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services. pp. 67–76. MobileHCI '11, ACM, New York (2011)
6. Dörk, M., Williamson, C., Carpendale, S.: Navigating tomorrow's web: From searching and browsing to visual exploration. ACM Trans. Web 6(3), 13:1–13:28 (Oct 2012)
7. González-Caro, C., Baeza-Yates, R.: A multi-faceted approach to query intent classification. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) String Processing and Information Retrieval, LNCS, vol. 7024, pp. 368–379. Springer Berlin Heidelberg (2011)
8. Grimes, S.: Breakthrough analysis: Two + nine types of semantic search (Jan 2010), `http://www.informationweek.com/software/information-management/breakthrough-analysis-two-+-nine-types-of-semantic-search/d/d-id/1086310?`
9. Hearst, M.A.: Search User Interfaces. Cambridge University Press, New York, 1st edn. (2009), `http://www.searchuserinterfaces.com`
10. Kreuzer, R., Springmann, M., Kabary, I., Schuldt, H.: An interactive paper and digital pen interface for query-by-sketch image retrieval. In: Baeza-Yates, R., Vries, A., Zaragoza, H., Cambazoglu, B., Murdock, V., Lempel, R., Silvestri, F. (eds.) Advances in Information Retrieval, LNCS, vol. 7224, pp. 317–328. Springer Berlin Heidelberg (2012)
11. Meisel, W.: The Software Society: Cultural and Economic Impact. Trafford Publishing (2013)
12. Pariser, E.: The filter bubble: What the Internet is hiding from you. Penguin UK (2011)
13. Pickens, J., Golovchinsky, G., Shah, C., Qvarfordt, P., Back, M.: Algorithmic mediation for collaborative exploratory search. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. pp. 315–322. ACM, New York (2008)
14. Sohn, T., Li, K.A., Griswold, W.G., Hollan, J.D.: A diary study of mobile information needs. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 433–442. CHI '08, ACM, New York (2008)
15. Turkle, S.: Alone Together: Why We Expect More from Technology and Less from Each Other. Basic Books (2011)
16. Waterworth, J.A., Chignell, M.H.: A model of information exploration. Hypermedia 3(1), 35–58 (1991)
17. Zhu, S., Zou, L., Fang, B.: Content based image retrieval via a transductive model. Journal of Intelligent Information Systems 42(1), 95–109 (2014)

Transparent Components for Software Systems

The future of software engineering in 2025

Paul Peschel

Technische Universität Dresden
Fakultät Informatik
Institut für Software-und Multimediatechnik
Lehrstuhl Softwaretechnologie

Abstract. Most of the improvements today and in the future will be achieved with software. Since more and more systems in our modern world depend on complex software, the development process of the software has to be adapted to upcoming needs in the future. A way to manage the complexity is to use tools for automated code generation by specifications. Hence, a key technology will be the composition of arbitrary software components to a big software system. For this purpose we will describe the usage of the Double Roof Model already used in the hardware/software codesign. We will also reflect the benefits and possible problems which may occur when using transparent software components. These software components are highly reusable and can be made by the open source community. So firms can almost move the expensive cost factors of code writing, testing and maintaining of their software systems to the open source community and produce with a little effort software with a good quality.

Keywords: automatic software component composition, transparent software components, reliability

1 Introduction

There is a run between the upcoming high performance embedded systems which are called Cyber Physical Systems(CPS) when they interact with other systems and the needed software systems to drive these devices. Because such digital devices are all getting cheaper more people can become users of such systems like the assistance systems in modern cars, intelligent devices at home building the internet of things, and so on.

The economy is pushing the transformation from an almost analogous world to a nearly full digitalized world because of the high profits this digital market will generate. Most innovations today are created within this digital market. With the increasing amount of devices and needs of the economy generated by the innovation and resulting profit pressure, the software systems will become more complex and have to be delivered in shorter time to the market. That is why we have to prepare for the future of software systems to make even very complex

software systems still maintainable in an easy way and the development cheap and fast to gain a high profit. The highest profit will be gained with software platforms in the future(see [3]). So the facility to easy generate platforms will be a key technology in the future of software development.

In particular we need software systems which satisfy the needs of the economy and hence we need approaches that enable the easy connection of all devices, collect and interpret data in real time and we need platforms which combine all these systems together to provide a perfect user experience. The software development process can already not follow the evolution of the new upcoming digital market resulting in many crashes of complex software projects.

The vision will be to create methods, tools and software development processes which enable us to create automatically software that will fit the functional and non-functional properties of the future applications. An achievable vision will be a framework which allows a user with less software engineering knowledge to create new software just by combining already tested software components(e.g. made by the open source community).

So the question is how should we improve the software developing processes to easy enable complex software systems? A nice approach would be to connect arbitrary software components together especially the open source components which are almost well tested and maintained by the open source community. So we have got an automatic evolution process of these components by the work of the open source community. The focus is on how to combine these components to reach a Pareto-optimal state [2] of the whole resulting software system and how to ensure reliability [5]. Because of the heterogeneity of the CPS we also have to ensure independence from the platform where the software will work on.

2 Related work

We want to present the ideas this paper is based on. In particular it was the book "Conversations"[4] and the ISTAG report[6] providing the ideas of experienced software developers for this paper. Furthermore the approaches in the Hardware/Software Codesign[7] and the researches in the reliability of software[5] completed the concepts for this paper.

2.1 Basic ideas

Because software is getting more complex with respect to the upcoming technologies like Internet of Things, Cyber Physical Systems (CPS), high performance computing on multi-core computers we need methods and tools to build more robust and resilient software. A possible way could be a tool which builds whole software systems without programming. But to fulfil the functional and non-functional properties we need such methods like the complete automatic simulation of the software including the interaction with the user. Another approach would be a target independent compiler analysis, static and dynamic(at runtime) code optimization.

The European software industry can become leading in the world when it exploits new ways of software engineering which increases the efficiency and quality of produced software. Hence, there is also a need to reduce the lead times to get more complex software projects finished. A way to get this managed is the reuse of core infrastructures (e.g. maintained by the open source community) of open source frameworks which are easy extensible(e.g. eclipse). Another possibility is the combination of existing software components which we will reflect in this paper.

The quality of software can be improved by introducing transparency into it like it happens in the open source software. Hence, a great community can give feedback about the code and helps to make it better and leads to a better quality of the software. Another approach will be to find errors as early as possible e.g. already in the meta model or in the code itself during compilation with pre-and postconditions and invariants. So we can avoid errors occurring at runtime. In this paper we will extend the idea of transparency in software by introducing transparent software components (see 3).

For this purpose we will also use the ideas presented in the ISTAG report. An advice of this report concerning software engineering in the future is the installation of a library of best practices and exemplary software like for energy efficiency or large scale efficiency management. Exemplary for such libraries are the mentioned websites *berlios.de* or *ow2.org*. The idea of this paper relies on establishing such a central managed library to use it as a source for transparent software components.

Because it is getting too difficult to develop a complex software system only by requirement specifications and descriptions we are pressured to use domain-specific modelling e.g. with ontologies. An approach would be to reuse the created artefacts for the domain in later software projects corresponding to the same domain. The created ontologies and requirements should be used to generate software systems only by the obtained design specifications.

In the future we need higher abstraction levels which helps us to manage the things which are hard to overview for humans like parallel programming and to let the software system work on different hardware architectures. For instance approaches in machine learning could help us to deal with high parallelism in future software systems. So we will present in this paper a possible tool similar to a synthesizer tool which generates out of a set of software components and design specifications the wished software system with respect to these for humans difficult to overview things.

2.2 Hardware/Software Codesign

In the future of codesign we have to deal with a growing amount of software dependent devices. It will become hard to develop rapidly new software for the hardware. One approach to create new software is the *Double Roof Model* of Codesign [7] (see fig. 1). It describes a way to create the software implementation and hardware structure by specifications.

The Double Roof Model just takes the specifications made by the developer out of the wanted functional behaviour of the whole system (left arrow in fig. 1). The synthesis step chooses then the appropriate architecture out of a known set of platform architectures and components called *Design Space* (right arrow in fig. 1) and generates the implementation corresponding to the specifications (vertical arrows from the top roof to the bottom roof in fig. 1). At this point the resulting implementations are used as an additional specification for other components (horizontal dashed lines in fig. 1). The result is a refined model which already contains the necessary non-functional properties and design specifications. As a source of software components and architectures for the Design Space we could use the software library mentioned in 2.1.To find the optimal components and

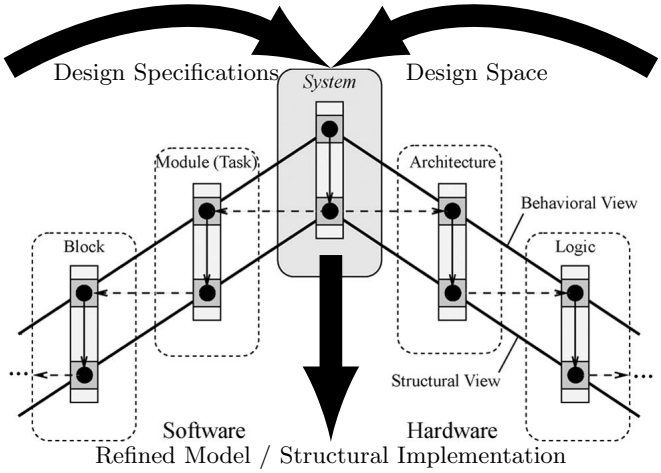

Fig. 1. Double Roof Model - Each result of one synthesis step (vertical arrows) is used as additional specification (horizontal dashed arrows) for the next step

architectures the paper describes a way of *Design Space Exploration*. The aspects the exploration respects are efficiency, optimality and there should not be only one but many components regarding to these aspects. To find an optimal Design Space corresponding to the wanted objectives we have to solve a multi-objective optimization problem. Algorithms like Integer Linear Programming(ILP) are used to solve the problem and hence to generate a Pareto-front of components building the *Input Space*.

In this paper we will use the Double Roof Model to create whole software systems out of design specifications. The Design Space will be a set of Transparent Software Components (see 3).

2.3 Software Reliability Engineering: A Roadmap

The future of reliability is focussing on software architecture and component based software engineering(see [5]). While in the past software reliability relied on specific design models, architectures and measurements with respect to reliability properties in the future this methods have to be improved.

A manually measuring of failures will be replaced by automatic methods which will not add unnecessary complexity to the software and don't disturb the system while monitoring it. The major task in the future will be the research of new failure-resilient software architectures guaranteeing separation of components during runtime. So failures can be clear assigned to components.

The software system should be able to measure reliability properties by itself like the time between failures as well as the number of occurred failures in a specific period. With this information the software system can predict failures of components by using e.g. machine learning techniques.

3 Transparent Software Components

Transparent software components are functional parts of software with an open source code. There are two views on transparent software components. 1) Firms can make parts of their software products public so everyone can take a look at the code. This kind of peer review will increase the quality of this piece of software (see [4]) by relying on the experience of other software developers. Hence, it will improve the code. A side effect of this code review is the saving of money and the production of software with a good quality. 2) Another aspect of transparent software components is the open availability of software made by the open source community. In this case we will introduce the term "transparent software component" as a software component which is free available, developed and maintained by the open source community and hence has an open source code. This software can be used to create a set of components where implementation frameworks can build an appropriate architecture corresponding to given design specifications out of this Design Space. For filling the Design Space it is strongly necessary to have a central managed repository of open source components to avoid redundancy and to enable strict versioning of the transparent software components. For this purpose exemplary software from the web could be used (e.g. *berlios.de* or *ow2.org*).

3.1 Evolution of Transparent Software Components

The Design Space of transparent software is governed by a natural evolution process in the sense that the open source community is doing a continues development. It leads to a continued improvement of the software and keeps it up to date.

The evolution process forces the community also to abort the development of software e.g. if there is no demand for it, the quality is not good enough or

the functional and non-functional properties do not fit the needs of the users and developers that use this software. This mechanism lead in the future to a set of well designed and tested software.

If we use this set as our Design Space the software which was generated out of it will benefit of the quality improvements of the components. This dependence between the software system and the used transparent components enable a low expensive maintenance and an automatic good quality of the software and hence lower tremendously the development costs.

Like in the natural evolution the set of open source software increases after new ideas and approaches leading to new software and maybe completely new branches in the field of software. So we have an automated innovation process.

4 Composition of Transparent Components

One problem of using transparent components in the Design Space is the composition of them to new software systems. Because the open source community almost does not develop components with specific coupling standards (e.g. defined by the industry to use components in a Design Space for generating software systems) we have to find ways how to compose components with an arbitrary interface and data. There are two ways of doing that.

4.1 Static Composition

The static coupling of components happens at the compilation time. That means that already at design time the data structures the components use to exchange information through their interfaces have to be known. In particular the developer have to use them with an interface definition language (IDL) or have to integrate the component manually connecting the interfaces at the right point in the code. Latter will be necessary for open source components which usually don't define their interfaces by an IDL.

A benefit of a static coupling is the opportunity to choose the most suitable components which fit the non-functional design specifications. Though, it is only useful if we already know at design time which components we will use for our desired software system. A future approach would be a tool which recognizes the interfaces of the available components by itself and knows how to compose them to a functional software system with respect to the given specifications.

4.2 Dynamic Composition - Roles

A dynamic coupling is in the future more applicable than a static one. Because future software systems should ensure a very high availability stopping the application for maintenance reasons should be avoided. Hence, a dynamic way of interchanging or adding new components is necessary.

Since with static coupling the software system has to be recompiled to add or replace an arbitrary software component we need an advantage mechanism

to couple components dynamically. However, we need a dynamic view of the components which will be realized with roles (see [1]) in the future.

The concept of roles will be introduced into the design part of the software development process. So we will introduce another model level above the meta model where we can create role models. After the roles and contexts are specified they will be mapped on the objects of the meta model(e.g. a UML model). Hence, the role model defines the interoperability and the interchangeability of the software system components.

The role model enables the software system a dynamic view on a new component corresponding to the role model. It allows a dynamic adding and integrating of new components if they fulfil a specific role in the software system.

The automatic determination of a components role relies on the facility to determine the components semantic. It will be a further scientific challenge to develop such a functionality but it will make a big step towards an automatic synthesizer tool. Hence, a further approach would be to pass the role model in addition to the specifications and Design Space to the synthesizer tool which maps the role model on the components of the Design Space with respect to the design specifications. Of course the semantic of the open source components in the Design Space have to be determined before. This is necessary, since the open source components were not intended to be used in an arbitrary software system and may have not any meta data about their semantics saved.

So for the future would it be very helpful to have multiple meta data like architecture information in the source code included. An additional external documentation will become unnecessary and can't get lost with the time. Design information in the source code of components will also make the automatic semantic determination less complex.

4.3 Pareto-optimality of the *Input Set*

Another scientific research topic will be the ensuring of Pareto-optimality of the composed components. Because a set of components composed to a whole system can lower the efficiency if several components are not independent from each other or do not fit the non-functional properties the input set of components to the synthesizer tool should be Pareto-optimal[7]. It defines a set of components in which each component conflicts the processing of another component as less as possible. Figure 2 illustrates the optimization of the component set.

This means before we can compose the software components to a software system we have to choose a subset (big ellipse in Fig. 2) out of the Design Space and generating the input space by making this subset Pareto-optimal. So the input set defines a Pareto-front created by solving the multi-objective optimization problem. Objectives can be to minimize looming conflicts components can have with each other and to optimize non-functional properties the system should have at the end. In Fig. 2 the length of the arrows above each component indicates the value of a specific property serving an objective. Hence, we have to derive the possible effects of each component to other components while execution and to measure non-functional properties even of the resulting system(e.g.

with simulations mentioned in [6,4]). This is a similar research challenge like the determination of the components semantic and behaviour. Optimizing the con-

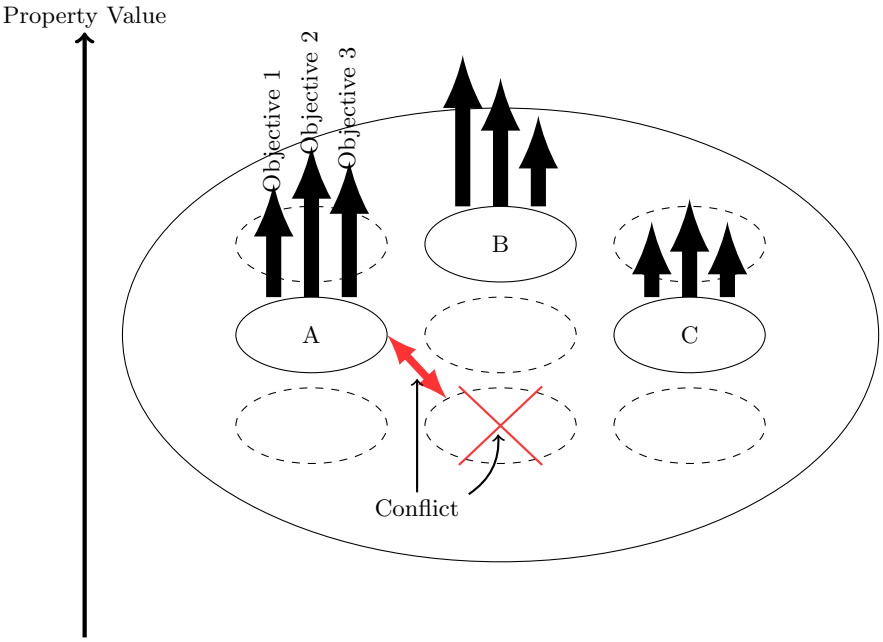

Fig. 2. Multi Objective Optimization - Automatic selection of components to gain optimal non-functional properties. The ellipses notated with a big letter define components which have valued properties. The values are indicated by the length of the arrows each serving a specific optimization objective. The goal is to select these components which improves the optimization objectives best without conflicting other components. Conflicting components with lower optimization benefits will be removed.

flict potential of the set of components will be usually achieved by separation of concerns and a high cohesion of the components. However, a resulting low coupling enables the software system to a better isolation of components which disturb the normal operation of the system (see 4.4). Hence, a Pareto-optimal input set results in a low coupling of the components and less complexity of the resulting software system.

4.4 Ensuring Reliability

Because we want to show in this paper a possible way how to create software systems in the future we reflect in this section reliability. A perfect software system will be completely stable. Though, a perfect composition of perfect working components resulting in a perfect stable software system may even with a

Pareto-optimal input set not possible. So it is necessary to ensure reliability to guarantee a stable software system. The need of reliability can be introduced with the interaction of the components because the components may be not tested in their actual runtime environment and hence we can not guarantee a failure free working in the system.

The separation of components as the intended consequence of generating a Pareto-optimal input set leads to a separation of concerns of the resulting software system. It makes it easier to identify failures occurring in the software system. Since failures in software do not happen independent (see [5]) an error can be easier assigned to a component if there is a high cohesion and low dependence between the components.

The software system should have the ability to reconfigure its components. If an error occurs and the responsible component could be identified the software system has three possibilities: It can reconfigure this component, replace it or put it into isolation to avoid further violation of the rest of the system.

A future advantage will be the automatic enabling of reliability in software systems. The developer should only define in the non-functional properties and several constraints which give the synthesizer tool a hint which components have to be selected for the input space. Again we need to determine automatically the behaviour of open source software components since the developer of the component can not provide measurement values concerning reliability because he usually does not know the runtime environment of the software system.

5 Conclusion

We have seen it will be a long way to establish a software engineering process to automatically generate software systems. There is still much research needed in some specific fields mentioned in this paper. For instance we should develop a tool which fulfils the synthesizer step for selecting software components corresponding to the given design specifications and composing them to an entire software system.

Because of the usage of open source components in the design space we gain an automatic maintenance and improvements made by the open source community. Hence, this will lower the software development costs of firms but also the authority over the components. The evolution of the transparent components will lead to a continues improvement of the software quality and to innovations in the domain of computer science.

Since open source developers creating their software just for a specific system or hopefully in a generic manner we need methods for choosing the right components out of the Design Space. For the selection of the software components out of the Design Space first we need an automatic way to determine the semantic and the runtime behaviour of the components in the appropriate environment of the resulting software system. This will be a further research topic(e.g. machine learning).

The complexity of the semantic and behavioural determination step can become less if we introduce a standard in the open source community which defines to write meta data into the components source code. For this, a programming language supporting architectural information inside the code would be helpful.

The recognized semantics and behaviour information of the components can be used to generate a Pareto-optimal set of input components which will build the software system at the end. The information enable also reliability of the software system and helps it to detect in the lead-up to runtime possible points of failure of the components.

In the year 2025 we expect a central managed repository of open source software components which can be freely integrated into software projects. This is the basis for the described synthesizer tool. Current researches like on roles (see [1]) should finish 2025 and enables a dynamic component composition. Also the standards for the development process with transparent software components will be defined 2025. The creation of a synthesizer tool will be a still open research topic for the time after 2025.

References

1. Prof. Uwe Aßmann. Role-based software infrastructures `https://wwwdb.inf.tu-dresden.de/rosi/lectures/rosi-lecture/`. [Last access: 22.07.2014].
2. Lars Boshold. Multikriterielle optimierung. `http://www.scai.fraunhofer.de/fileadmin/ArbeitsgruppeTrottenberg/WS0809/seminar/Boshold.ppt` [Last access: 30.05.2014].
3. Michael A Cusumano. *Staying Power: Six Enduring Principles for Managing Strategy and Innovation in an Uncertain World (lessons from Microsoft, Apple, Intel, Google, Toyota and More)*. Oxford University Press, 2010.
4. Sebastian Nanz Edgar G. Daylight, editor. *Conversations: The Future of Software Engineering: Panel Discussions*. Lonely Scholar, 2011.
5. Michael R Lyu. Software reliability engineering: A roadmap. In *2007 Future of Software Engineering*, pages 153–170. IEEE Computer Society, 2007.
6. ISTAG – Information Society Technologies Advisory Group (Working Group on Software Technologies). *Software Technologies – The missing Key Enabling Technology (Toward a Strategic Agenda for Software Technologies in Europe)*. July 2012.
7. Jürgen Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1411–1430, 2012.

Functionality, Threats and Influence of Ubiquitous Personal Assistants with Regard to the Society

Jonas Rausch

Dresden University of Technology
Jonas.Rausch@mailbox.tu-dresden.de

Abstract. With the ever increasing power in computation, the capabilities of software to perform very complex processes and tasks are expanded to a hole new level. This leads to more benefits for people due to more reliable and smarter software. With the help of such software, tasks which were seen as not automatable only a few years ago will be accomplishable by machines. This paper outlines how this trend will continue towards the year 2025 with the invention and ongoing development of personal assistants on mobile devices. The communication with these assistants will be mainly based on voice input from the user which will feel much more natural as of today due to the rapid progress in fields like natural language processing. Other key technologies like speech understanding and machine learning are also playing a great role in terms of developing more natural personal assistants. This development will peak eventually in an extended form of the Personal Assistant Model: Ubiquitous Personal Assistants, which will have an impact on every part of our life due to the time- and location-independent possibility of usage through multiple mobile devices. An unpleasant side effect will be security, safety and social issues that have to be examined and solved simultaneously. Finding solutions especially for the security problems will be the key factor for Ubiquitous Personal Assistants getting introduced and accepted by a wide variety of people in the upcoming years. Independently, companies will extend the use of personal assistants due to workflow-related reasons which enables the increasing automation of jobs and thus will harm the society.

Keywords: personal assistants, PAM, Personl Assistant Model, UPA, Ubiquitous Personal Assistants, smart software, job automation, cloud-based software, mobile devices, voice communication, NLP, natural language processing, natural user interface, speech understanding, machine learning, knowledge representation, technology communities

1 Introduction

In the year 2014, technology is pervasive in almost every part of our daily life. Whether we use our smart phones to check the weather forecast for the next days or consult the GPS system of our car to get directions, the connection to

mobile services from mobile devices gets indispensable and self-evident. With the help of the Internet, the software behind these services can be ran in the cloud on multiple powerful servers, allowing the software on the mobile devices to be executed with much more speed and efficiency than the device alone could provide. You experience this power every time when you do a *Google* web search, since the calculation of the resulting web pages is done by servers in the cloud. This cloud-based software also leads to a faster change of software because it is no longer necessary to buy software physically and install it from a CD or DVD on your PC. Instead, purchasing and using the software is done online. The user is not forced to buy new hardware or to worry about maintenance anymore, even the updates are installed in the cloud with the result that the user does not have to worry about them. Thus, little improvements to the software can be added very fast and the company which sells the software deals with only one version which can be maintained easier than various released versions with every customer having a different one.
Cloud-based software leads to the profitable situation that we can use computer intelligence paired with much computational power from almost every location through our mobile devices. Even as of today, almost every information we obtain on our device originates from online services which are ran by cloud-based software, thus a connection of the mobile devices to the Internet is very important.
Now, with the infrastructure of cloud-based software and fast wireless connection for mobile devices set up, the next step would be to tighten the human-software connection in a way that we can take advantage of these technologies not only in the private sector but also in our job and every part of our life. This should additionally be done in a more quickly and natural way. The *ISTAG* (Information Society Technologies Advisory Group) labels such mobile devices as "smart objects" which "will increase the intelligence, control and communication capabilities of a wide range of objects, enabling their interaction and cooperation with people and organizations." [1]
The vision is to connect people and software in a way that an individual is time- and location-independently able to search for and obtain any needed information or services through a natural user interface by using computer intelligence and resources provided by the world wide web. An approach to this requirement is the Personal Assistant Model (PAM) described by William Meisel [2]. This paper discusses required technologies, possible capabilities and the usage of personal assistants which will be available in every part of our life in the near future towards the year 2025. Furthermore, I will take a look at Ubiquitous Personal Assistants [2] which are based on the Personal Assistant Model and embody an extension of the search concept used by these assistants. The occurring risks and negative consequences of personal assistants are also covered in a later section.

2 The Personal Assistant Model

The main approach of the Personal Assistant Model (PAM) is the usage of a personal assistant that is integrated in a mobile device. To fulfill the vision mentioned in the section above, several key technologies are required.

2.1 Required Key Technologies of the PAM

The concept of the Personal Assistant Model (PAM) aims for a communication with a mobile device that is as natural as possible. The user interface of personal assistants should therefore make use of techniques which on one hand imitate the interaction with another human and on the other hand do not require prior knowledge of using advanced mobile devices like smart phones. The avoidance of "too technological" foreknowledge could help to delight technology-reluctant or skeptical people and to prevail them to use personal assistants, while they still would not use a smart phone. Especially older people benefit from this approach since they have generally a lower motivation to learn using new technologies, even though they would profit greatly and in some cases would have to rely on personal assistants, for example instructing the personal assistant by voice to do an emergency call.
Thus, advanced natural language processing (NLP) is one of the key technologies for an effective usage of personal assistants. In earlier times, especially in the 1980s, the focus of research laid on speech-to-text respectively speech recognition where spoken words are translated into text [3]. A technique that is very useful for dictating messages or calendar entries into our mobile device, but not sufficient enough to actually communicate with the device. It does not understand the meaning of the words, it just recognizes them with statistically-based speech recognition algorithms [2]. For advanced tasks, for example asking the device whether there is a meeting scheduled for the next day, the much more challenging technology of speech understanding is indispensable. Today, personal assistant applications perform the computation intensive process of speech understanding in the network or cloud.
Another key technology that supports the NLP of the PAM is machine learning. It enables the personal assistant to consider and learn from your personal interests, search behavior, location, financial situation and many more factors in order to answer an information request. The personal assistant becomes personalized and adjusts its behavior to the user, leading to better search results that are much more suitable for this particular user [2].
A further requirement for the PAM is to provide the answer to an information need as directly as possible, which means that the result is not a list of websites that may contain the needed information but the information it self. To accomplish this task, methods of the last key technology, knowledge representation, are necessary. These methods extract the answer to a query out of knowledge-bases [4]. Today, early forms of personal assistants like Apple's *Siri* can manage this task for queries in specific domains like the weather for a certain day or the result of a football game.

2.2 Applications of Personal Assistants

With the help of the presented technologies, you can communicate very naturally with the personal assistant and advise or consult it to accomplish different tasks for you. The underlying software of the personal assistant can process queries being committed as voice or text input. Even though voice interaction is the more sophisticated and desired option, text interaction remains important due to occasionally ambient noise or privacy concerns which make voice input impracticable. It is of course also possible to issue instructions per gesture and touch. If the subject to be searched for is not comprehensible enough, e.g. poorly described, the personal assistant should request additional information or suggest alternatives by voice in a communication-like manner. The queries are interpreted by the personal assistant's computer intelligence using the mentioned technologies of NLP. Afterwards, a search for the extracted objective is not only executed in the web but also in the personal information which are stored both on the device and in the cloud. Thus, our personal information become more organized and more easily available to us because we could simply run a search on them. Due to the processing of the software being done in the cloud, search results get returned very quickly and accurately. These results should be presented in the most natural and intuitive form, for example by voice with supporting graphics or pictures. Another feature would be providing information proactively in appropriate situations, even though the user did not send any information request. Because of the described possibilities of communication, there is a trend to an ever increasing dependence on personal assistants because they simply expand what we know and what we can do. We will start to rely on them in a way that makes it almost impossible to not having it with us when we leave the house, whether it is for work or to engage in a hobby. Today, examples for the dependence are messaging services, access to the email account or the account balance, calendar entries or simply music and e-books. In the future, even more tasks can be accomplished by personal assistants. Additionally, time- and location-independently use is provided by assistants located on mobile devices.
As the research on NLP proceeds, mobile devices will be fully hands- and eyes-free controllable, making other forms of interacting in most cases dispensable. This leads to the opportunity to use these devices as assistance in one's job where using both hands all the time while having the personal assistant as source of knowledge accessible via voice is a huge advantage.
All in all, the result of the current development will be a mobile device with an integrated personal assistant which can basically help you in every part of your life. Thus, I sympathize with William Meisel's view as he stated in [2] "Increasingly, we will view our mobile devices as an extension of ourselves."

3 Ubiquitous Personal Assistants

Personal assistants are either specialized to a certain field of operation or provide general usage for all kinds of objectives. [2] Generalized personal assistants encounter the problem that they can not answer every question to every topic

because they have a limited scope of knowledge mostly related to common use-cases (for example navigational tasks, the weather or making an appointment). To solve this issue, many different personal assistants have to be connected to collect all the specific knowledge for every context. It is easy to see, that this does fall short of the once formulated vision of a personal assistant that can provide help in every part of a human life. Instead, the user is forced to maintain a more or less uncomfortable number of personal assistants on multiple devices which constantly have to communicate with each other. Another problem is the not consistent user interface because it is not very likely that every single used personal assistant is from the same company.
Again, William Meisel proposed a new idea: Ubiquitous Personal Assistants (UPAs), an extension of personal assistants based on the Personal Assistant Model. [2] The concept of Ubiquitous Personal Assistants is that there should only be one personal assistant across multiple mobile and stationary devices (Figure 1). This single personal assistant manages all the data that was formerly distributed over many assistants in the PAM. Knowledge about the user that was learned through machine learning on one device is now available and used on all the other devices, too, because there is only one single personal assistant that collects and retains all insights. Thus, the UPA is required to have access to all of our personal information in order to facilitate an efficient use. The great advantage of sharing all your information and data with only one personal assistant are the new possibilities related to answering an information request by the user. Due to the central storage of your emails, contacts, files on the PC, media files and the information earned through personalized machine learning, it is possible to search simultaneously in all these sources of information, instead of having to execute a search process on all the different normal personal assistants. A common user interface and therefore only one "interface language" mark yet another advantage. An additional benefit emerges due to the universal usage of all devices that a person owns, which makes it feasible to use the stationary PC at home as one among many devices. This is a clear improvement in comparison with the PAM.

3.1 Impact on Companies - Technology Communities

Due to the comprehensive range of usage of UPAs, company's interest will be drawn. Through the use of only one platform across multiple devices, companies receive two major advantages. Firstly, one platform is easier to develop, upgrade and maintain, increasing the agility of the software running on this platform. The user obtains very similar benefits because he only has to worry about managing a single piece of software, the UPA, offered by only one company. Secondly, the customer gets tied to the company by buying their UPA because it invades every part of the customer's life, even his job where the today occurring "Bring Your Own Device" trend will likely be strongly enhanced as it is stated in [2]. The result is that the user will be increasingly dependent on the company and their technology community, making it hard to switch to another company's community. The best example is *Apple* which is nowadays trying to establish

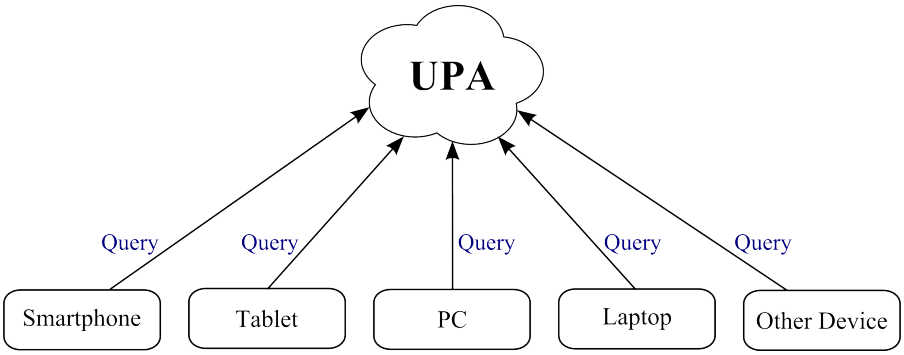

Fig. 1. Multiple devices send information requests (queries) to a single Ubiquitous Personal Assistant (UPA)

their community by providing both software and hardware, a comprehensive cloud-based storage system synchronizing data across multiple devices and their own App Store which enables Apple to have full control over the variety of apps that is being offered to their customers [2]. This trend towards technology communities will increase with other companies concluding the same as Apple and trying to establish their own community, for example Google with its *Android* community or *Facebook* trying to enter this field through Social Media. In the future towards the year 2025, those big companies will try to get more and more control over their users, including buying in smaller companies with innovative ideas [2], for example Facebook buying recently *WhatsApp*, *Instagram* or *Oculus VR*, a company developing virtual reality displays that are attached to the user's head [5]. My prediction is that in 2025, technology communities will be more established as ever, making it almost impossible for smaller firms to come up with new ideas without getting bought in. Even big companies like *IBM* or *Hewlett Packard* that do not develop own communities will be forced to join one of the other communities in order to make the own work more efficient and contemporary.
Due to the fact of one company possessing all of your data, many security issues are raised, which I will outline among other threats in the next chapter.

4 Issues of Ubiquitous Personal Assistants

4.1 Security

The producer of the Ubiquitous Personal Assistant will use cloud-based software to process the data of the customer, which means that also all the personal data is stored in a network, possibly unencrypted. In case of a hacker attack, not only one type of (possibly critical) data could be stolen, but all the data associated with the customer which would cause significantly more damage than for example "only" a stolen email account out of possibly multiple accounts the user owns.

Additionally, advanced methods and algorithms of natural language processing have to be computed in the cloud because of speed and accuracy requirements. Thus, data has to be send almost continuously from a UPA to the network and could be intercepted meanwhile [2]. This issue gets exacerbated due to the fact that the data also gets stored over long periods of time in the cloud which leads to an increased exposure time of the data [6].
Another security relevant feature of personal assistants is providing information proactively in appropriate situations, even though the user did not send any information request. The personal assistant can only recognize such situations under the premise that it analyzes every situation in the first place, which could be a serious security issue. Consider an example in which a business meeting is taking place and your personal assistant automatically records everything that is said, whether it is confidential or not, in order to do some research on the main topics only to provide you with some additional information. The search related to these confidential information could be intercepted by an attacker and the company would suffer significant damage.
A very relevant problem is also the ongoing interweaving of mobile devices in private use and business use. As already mentioned, the "Bring Your Own Device" trend will increase exponentially in the future, not least because they become in fact indispensable for the work because of the assistance they provide to many jobs [2]. The integration of UPAs in the business will cause many security concerns. Business software has to be installed on those devices, confidential data is stored on them and will most certainly get uploaded to a cloud for processing or synchronization tasks, which brings us back to the first issue I have described. Furthermore, if the device of the user gets infected with a virus, the business data is also affected and in acute danger.
The described problems of data theft can potentially be solved by the cloud providers. As it is stated in [6], they have to implement "an information security program and [put] in place effective, reasonable and adequate safeguards that cover physical, administrative and technical aspects of security". Small or mid-sized businesses could even increase their security by providing and controlling the cloud services in-house. In general, there are three big areas to focus on: innovative regulatory frameworks, responsible company governance (which refers to the above described liability of cloud providers) and using appropriate technologies which support privacy, for example encryption and anonymization. Refer to [6] for a much more widespread overview as well as concrete concepts for these three areas.

4.2 Safety

The usage of UPAs will also play a role in safety applications, for example as an on-board computer in an automobile. The enabled possibilities of making a hands free phone call, receiving navigational directions, playing your own music and monitoring all information regarding the vehicle, just to name a few applications, via a single integrated device are very comfortable and desired by the customer. Furthermore, the personalized nature and constant connection to

the Internet of UPAs could facilitate a better assessment and forecast of current situations (e.g. traffic jams) based on the individual road behavior and the present location. An autonomous driving assistant would be the next step and gets developed already [7].
It does however become problematic when a system failure occurs. Maybe it is triggered by a function that is not even required for the actual purpose of the system (which in the case of using UPAs would be various functions). Though it could cause the hole system to shut down which induces a great threat to the user in safety critical applications like the mentioned self-driving car.
A possible solution would be to improve the safety engineering, especially by proving the software in terms of correct functionality in every possible situation. Methods like root cause analysis and stress testing could also be helpful [8].
The main counterargument for the safety and security concerns is that an efficient use of the NLP key technologies of UPAs is not realizable without the processing power of multiple servers in the cloud. It is eventually a trade-off between using the enormous potential and assistance of UPAs (as well as normal personal assistants) and the occurring issues [6].

4.3 Digitalization of Our World

Additionally to the mentioned security and safety issues, the usage of personal assistants in general will exacerbate the already ongoing digitalization of our society. The vast amount of possibilities and the constant usage of UPAs could cause a kind of addiction to the device(s) and an anxiety of losing these capabilities [9], a phenomenon which already can be seen today in terms of smart phones. In such cases, human communication mostly takes place via electronically gadgets. This trend will increase due to the widespread use of personal assistants and thus everyone getting available through a device. There is no need to meet in person anymore, people will rather stay at home. Additionally, the real world will be increasingly easier to simulate and at some point in time, an online "second life" could be established, possibly leading to a loss of sense for reality.
Furthermore, the addiction to a device and the insistence to use them continuously can lead to considerable risks when used in the wrong situations [9], for example while driving a car, riding a bike or even passing the street while staring on a smart phone. The resulting moment of abstraction can lead to tremendous medical consequences.

5 Automation

A big impact on the society emerges out of the extended possibilities of automating once human controlled tasks through not only the use of UPAs but in general more efficient and intelligent software. This trend results in a higher overall productivity [10] which is desirable for companies and could also lead to reduced product prices [2]. Though, full automation of jobs has too many downsides.

The most significant one being the massive loss of jobs due to a decreased need of workers for jobs that require only minor training to be executed because of their repetitive nature. Industrial robots currently are and prospectively will increasingly take over this kind of jobs. Other negative impacts of full automation are the destruction of certain job categories (e.g. assembly line work) and less income for the remaining employees [2]. Thus, we have to find the right balance between utilizing the great possibilities of automation and retaining reasonable working conditions.

5.1 Race With the Machines

A better method to use the capabilities of higher developed software is to exploit the synergy between computer intelligence and human strengths to create a "potentially beautiful partnership" [10] for achieving results which are only possible by using the best properties of both worlds. In the foreseeable future, the combination of human adaptability, experience and reasoning together with the accuracy, computational speed and reliability of machines coupled with software remains indisputable [2]. Additionally, the advantages of personal assistants like voice communication with the machine as well as hands and eyes freedom for the worker can also be used. Furthermore, new job categories get created that would not exist with the help of either side [2].
A good example for the use of this synergy in the workplace is the health care sector [2] where software, possibly running on the mobile device as a personal assistant of a doctor, could automatically collect and organize information about the doctor's patients to be treated. The doctor would not have to waste so much time for deriving information out of the medical patient record due to the intelligent software already having collected and appropriately summarized these information. The real treatment as well as the diagnose is left to the human expert who is far better at considering different medical aspects, additionally to his work experience. Another enabled application are Warehouse Management Systems [2] where workers receive the position of a specific item in the huge warehouse via personal assistant software that keeps track of the inventory.
The big advantage of the synergy for employees is simply that most of them will will not lose their jobs, at least in the near future towards the year 2025. However, at some point in time, software and personal assistants will definitely be that highly developed in the matter of reasoning and NLP that comprehensive automation will become inevitable.

6 Conclusions

This paper described the huge impact of the Personal Assistant Model and its extension, Ubiquitous Personal Assistants, on the society in the near future until the year 2025. It also outlined the numerous benefits as well as the drawbacks and potential endangerments.

The already existing and prospective security problems will be the major challenge during the introduction of UPAs, especially in businesses. Data privacy and protection will play a role in this context. Without proper solutions and security standards, personal assistants are going to experience rejection from a not negligible number of people. Although there is neither entire security nor safety, companies will commence and expand the use of UPAs towards 2025 due to their immense advantages for workflows [2] which will affect the society in a negative way due to the increasing automation of jobs. Only a possible big leak and abuse of sensitive business data and classified technology knowledge could lead to rethinking.
Society has to develop an open-minded awareness of the evident threats of the ongoing digitalization of our world which gets exacerbated by personal assistants. As long as these threats get recognized and controlled by the society, UPAs in my opinion are going to improve the world we live in significantly, especially at developing safety critical applications and providing various aids for older people. Therefore, research should continue the already chosen path on developing and extending Ubiquitous Personal Assistants.

References

1. ISTAG - Information Society Technologies Advisory Group (Working Group on Software Technologies): **Software Technologies** - ***The missing Key Enabling Technology (Toward a Strategic Agenda for Software Technologies in Europe).*** European Commission, Bruessel, Belgium, 2012
2. William Meisel: **The software society - *Cultural and economic impact.*** Trafford Publishing, Bloomington, USA, 2013. ISBN 978-1-4669-7411-1
3. **Natural language processing.** Retrieved 15:18, May 16, 2014, from http://de.wikipedia.org/wiki/Spracherkennung
4. **Knowledge representation and reasoning.** Retrieved 16:37, May 30, 2014, from http://en.wikipedia.org/wiki/Knowledge_representation_and_reasoning
5. **Oculus VR.** Retrieved 10:45, May 29, 2014, from http://www.oculusvr.com/
6. Siani Pearson, George Yee (Editors): **Privacy and Security for Cloud Computing.** Springer-Verlag, London, 2013. ISBN 978-1-447-14188-4
7. **The latest chapter for the self-driving car: mastering city street driving.** Retrieved 14:32, May 28, 2014, from http://googleblog.blogspot.de/2014/04/the-latest-chapter-for-self-driving-car.html
8. **Safety.** Retrieved 15:17, May 28, 2014, from http://en.wikipedia.org/wiki/Safety
9. Sherry Turkle: **Alone Together** - ***Why we expect more from Technology and less from Each Other.*** Basic Books, New York, USA, 2011. ISBN 978-0-465-01021-9.
10. Erik Brynjolfsson, Andrew McAfee: **Race Against the Machine** - ***How the Digital Revolution is Accelarting Innovation, Driving Productivity, and Irreversibly Transforming Employment and the Economy.*** Digital Frontier Press, Lexington, MA, USA, 2011. ISBN 978-0-9844725-11-3

Evolution-driven Changes of Non-Functional Requirements and Their Architecture

Hendrik Schön

Dresden University of Technology
Hendrik.Schoen@tu-dresden.de

Abstract. Software is a product that continuously changes and evolves to adapt to a more and more integrated product in our life. Today, only a few new technologies are developed without integrated software products or embedded software. This progression forces older technologies, best practises, component designs and at least the whole style of the architecture to change side by side with the evolving domain and scope of software products, technology and platforms. Finally, software will evolve to an integrated system that will play a significant role in our life. This paper demonstrates the main drivers of software evolutions of today and predicts how non-functional requirements along with their architecture and design decisions have to adapt to fit to the new changes. It is obvious that not all of the available qualities can be discussed here but the chosen ones give a well suited view to the most important software aspects: performance, availability, modifiability, security, testability and usability. Beside that, it is difficult to compare the importance of the qualities with each other. Research and further development will show the point which will become significant. As a personal view I would predict that security, availability and modifiability are the most critical issues for future software systems and future systems will rise and fall with them. All in all, this paper shows the increasing importance of such qualities in the future rather than a real ranking between them since software systems in 2025 will requires attention to all of them.

Keywords: Software Evolution, Drivers, Non-functional Requirements, Qualities, Architecture, Performance, Availability, Modifiability, Security, Testability, Usability

1 Introduction

While technical progress moves on and new technologies and needs are discovered, software has to evolve and move on accordingly. Thereby, some software principles and styles are maybe adopted or just not relevant in the new environment any longer. New software architectures have to be deployed in the near future. The upcoming environment in 2025 in some cases will have varying requirements for new situations with regard to many non-functional requirements like modifiability, privacy or safety. Innovations in the last twenty years have shown, how variable software should be. For example, how the Internet has

changed the typical single-machine architecture to a wider server-client structure to reach more platforms and devices to interact with. With introduction of smartphones and tablets, the full-installation has given away to the newer cloud-based application structure. Reuse of components and frameworks are now more important than ever. Along with that, the payment methods change and software is constantly moving to the software-in-the-cloud-style. That requires time-dependent fees but provide higher availability and some further functions for micro-payments and others. As mentioned in [14], the major transition is driven by development in other technology sectors. Therefore more types of problems are to be solved through software to improve the social life and scientific researches. "The future is the result of choices made today", as also stated in [14], describes a well defined view of how software has to improve now for its duty to satisfy the needs of the future environment and therefore guarantee the non-functional properties through new architecture solutions.

This paper consists of two major parts. Firstly, it will determine the main software evolution drivers within the context of the current developments. The second part takes a deeper look at some selected non-functional requirements and evaluates their impact and challenges in the future with regard to the previously introduced drivers: performance, availability, modifiability, security, testability and usability.

2 Changes of the Software Product and its Environment

2.1 Software Technology Evolution

It is predictable that established software designs would not always fit the future environment any more. Current technology trends demonstrate some of the future needs of non-functional properties. Cloud computing, as one of the biggest trends, has a direct impact on software architectural styles to provide elasticity and scalability [13]. Today, everybody wants to store, manage, synchronize, deploy and share things by the cloud system or a cloud based architecture respectively. This trend has received a boost in the last years, and it will perspectively increase even more. The main program's business logic layer and data layer will continue moving away from the client side into the cloud system. Even once big monolithic architectures now use increased bandwidth which allows them to use more complexity over network, for instance the *Adobe Creative Cloud* [1] or *Microsoft Office Online* [9]. Beside that trend, parallel computing also plays an important role of future software systems. As you can build, deploy and run software on a web server (or any other cloud based system) you have access to much more computing power through provided cloud-based services like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) or Software-as-a-Service (SaaS). Those are parts of hosting centres where you can simply rent computing power and coupled maintenance. This trend will lead to to more cores, more nodes and more parallelism. In general the complexity and scale of such systems will increase in future. On the one hand the clock frequency will be consistent [15] because of the physical limits (component size, heat, etc.) and

on the other hand the energy power consumption will decrease further in the future through a new thinking in "green energy" (for example, through intelligent resource usage and scheduling). The number of platforms on which software should run, would increase in number while the software has to deal with shared hardware and distributed systems. "The old paradigm of software programs developed for a single hardware platform or operating system will be obsolete." as mentioned in [13] is a important point of how software has to act in future in this field. Newer computing systems will range from smartphones to huge server clusters and each of them will provide parallelism and heterogeneity [13]. Common aspects of software architecture as structure, data flow, communication, memory and others [7] should be adjusted to the requirements of the future scenario.

2.2 Embedded Software Evolution

Software would also be increasingly needed for embedded systems like medical devices for artificial organs or components for connected physical systems (e.g. "smart grid" systems, robots). These systems will be a critical part of our social and scientific life. Thus, it will be essential to evaluate the correctness of software, e.g. through testing routines (unit, assembly, regression and system level tests) or code/logic verification. The awareness of responsibility of the software itself and the programmer of just this software code is significant as the software may have huge critical impact on human life. "In the next years, by further exploiting the Internet and the world-wide-web, embedded systems will increase the intelligence, control and communication capabilities of a wide range of objects, enabling their interaction and cooperation with people and organizations." [13]. The general need for intelligent and smart software will raise to satisfy our needs more and efficiently. Today, we can spot several projects of embedded systems in our social life with a kind of artificial intelligence or at least semi-intelligent software and hardware products, like self-driving cars [6], *Google Glasses* [5] or smart watches with health functionality. The number of those smart objects will grow in the future. Such embedded software products need to adjust its structure to provide all of the requirements in this comparatively new field of use. It is foreseeable that all of those smart objects will be connected via the web and they also would have relatively low power cores but the tasks they will perform become more complex and have to be executed faster. In general, the technical trend seems to follow either the software-hardware-synergy or the independence of hardware and software within complete systems.

2.3 Data Evolution

A further view of future software systems would show an increasing mass of data that has to be stored, processed and analysed. The variety of data from sources will be a wide field and contains structured (relational DBMS, XLS, HTML, PDF) to semi-structured (XML, web services, RDF) to totally unstructured data (sound, image) [13]. But that have to be integrated in one single system in order to analyse these data for other purposes. Also, the data processing follows the

general trend and will be rather real-time analysis than an follow-up summary to provide more service for, e.g. the current customer. Beside that, more and more data will be collected and current database sizes will explode. Less costs and less physical size for more storage capacity will lead to a database system beyond the traditional relational DBMS. Data mining, data management and visual analytics of huge amounts of semantic data then will play an even bigger role in enterprises and science. In order to provide functioning software products, its architecture has to be prepared for this new scenario of data processing to continuously provide acceptable requirements like performance, availability or data integrity. Data transfer, fetching, caching or intelligent locking will then play an even bigger role in such systems, just to mention some of viable methods here.

2.4 Software Principles Evolution

Software standards will be mentioned here as a last bigger part of changing software described here. As long as software itself should evolve, the software standards and best practises would also change side by side. Next generation programming languages allow better describing of behaviour of new systems. Novel protocols serve a likely functionality for incoming needs of social and scientific life and work. Mixed standards will be needed for a proper functionality. "As one would have recommended a full C++ or Java based vertical solution, we could have today a mix of *node.js*, PHP, *iOS*, *Android*, C# [...]. The open protocols (SQL, JSON, HTTP, SSL etc) are solid and rich enough to allow this technological fragmentation." as mentioned well in [13]. Also the code paradigm seems to change since much more open source software is available for free. Free or payable eligible components can be connected together to provide a component based and modular functionality as well as reuse and interchangeability of systems (or parts of a system) instead of a full monolithic and closed architecture to serve more use cases. At least it is worth mentioning a trend for automatically produced software as a factor for cost and performance efficient programming.

3 Non-functional Requirement Trend Assessment

Until now, we have seen the main evolution drivers of software. In this section will try to explain and predict the challenges and impacts of some of the major non-functional properties of software systems, based on these drivers. The selected qualities are chosen from [7] as these represent a good view on the overall software system in my opinion. Some parts of given assumptions here are mentioned in the above section so I do not explain them here furthermore.

3.1 Performance

Even though future systems have to accomplish more work, we still expected at least the same level of reaction at incoming events. It creates the demand of

the future that software is able to respond to real-time information [14]. Since software will handle an even bigger amount of data and will be used in a wider range and amount of embedded systems, it is the task of future systems to provide real-time processing to the user. A technical opponent will be the latency by limited bandwidth since the software logic and data layers move into cloud systems and the increasing amount of system nodes like cores and threads hold other resources (CPU, GPU, dependencies, data, etc.) for diverse objects by which the waiting time expands. The top level approach for lower resource latency (including the bandwidth) are more important than ever before. Some significant design decisions here are the following methods [7]. Firstly, one can simply reduce the resource demand, e.g. by decreasing the incoming events or the plain amount of the resources through better algorithms, more code efficiency or hard limited execution time. Secondly, it can be realised through better resource management like queuing of events, parallelism or simply more available resources). And at least, one can perform resource arbitration (through better scheduling). Thereby enough resources will be available in future IT systems, the management and scheduling are the more important aspects. Also worth mentioning is the maximization of the performance while reducing the power consumption [13] as an important point of future systems because the new trend for "green energy" and environmental awareness. Therefore more nodes are the more predictable trend than more powerful nodes which leads, still in combination with efficient design and programming, to lower power consumption (not to say there are also other methods to decrease power consumption). Through the fast expansion of embedded systems ("smart objects" as semi-intelligent embedded systems) or connected devices, the need of faster processing of more content is a crucial fact software will be faced with. In history, for example *Google*'s *MapReduce* [8] has shown fast indexing of the new web content, new database models like *Hive* [3] are created to handle massive amounts of data content (like *Facebook*'s data). Also, "main memory database concepts have demonstrated significant performance improvements above the classical database approach. Combining this approach with parallel programming could generate a complete new generation of enterprise software systems [...]." [13]. Prospective systems have to implement similar approaches or intentions for the upcoming content. Performance is, like many other qualities, a major important factor for business applications as high latency or slow reactions decrease users happiness. The establishment of alternative devices like smartphones and tablets increase the requirement of proper performance guarantee for all platforms and use cases.

3.2 Availability

It is the intention of the most enterprises to satisfy the user in his needs anytime and anywhere. In a shop, the customer should buy things when and where he wants to, either with his smartphone, tablet, desktop or his notebook. To reach that goal of service, permanent server and system up-time is inescapable. "An increasing reliance on software-driven artefacts will require that the software as infrastructure [...] is always-on." as mentioned in [13]. As a suitable example here

the *Adobe Creative Cloud* can be mentioned. Many creative businesses depend on this concept as a working tool for them. But, if the cloud software is not available (as happened in recent past) all of them are unable to work with their data and software. This has a huge negative impact for both, the customer (and most of the time also the customer's customer) and *Adobe* itself (this has also a direct impact on *reliability*). Also, it is important for the up-time that future systems are checked against failures (cf. Sect. 3.5) as software will be a very critical part of many physical systems, social and political infrastructures and medical services. Therefore a timely fault detection is a major issue. As systems are more distributed in the future, an appropriate management for fault detection systems is needed to handle the greater amount of echo, ping, heartbeats or exceptions. Faults have to be dealt with and solved at runtime through the software itself instead by a programmer afterwards [13] to keep the needed server up-time. That means to predict, detect and solve faults by decision making by the software itself, while preserving the performance and availability. This can happen by reallocating resources, change sub-platforms or components just-in-time in an autonomous way [13]. The most likely outcome will be that simple backups, checkpoints and rollbacks are not suitable anymore, at least not as single fault recovery tool because every minute and hour produce a huge amount of data and enterprise turnover which is lost if we roll-back. And such loss of data is not acceptable for current and future enterprises. Distributed and well-chosen RAID systems will be a standard and a must-have. Other architectural methods like redundancy, state-synchronizing or spare machines (failover computing units) [7] then will become more important than today to serve better "service-level agreements" to the user.

3.3 Modifiability

As mentioned in the previous section of evolving software, these future systems became more component based and therefore more modular, scalable and reusable. Since requirements will change during the lifetime of a software system, modifiability is required to serve such transitions of functionalities. Furthermore, "Software components will be combined in such a way that the result solution will provide new functionality emerging out of existing software components and services." [13]. That is a very important fact as future software will be written completely new only in a few cases. This will have a direct impact on testability (cf. Sect. 3.5). It is the task of software to adapt to newer platforms and environments as their number will be increasing in the future (e.g. a smart-watch or new car entertainment systems). In order to deliver such variety it might be very useful to build component-based software to just modify some parts and linking them intelligently to support a specific system. Big online repositories should give access to large component databases, solutions and protocols and help to characterize the quality and security of these items [13]. "Future systems will often have to operate under conditions that differ significantly from the ones for which they were designed." [13] to get to the heart of it. Several architectural principles are thereby generally relevant for component-based and

modular systems [7]. First, it is important to localize the possibilities of modifications through defined interfaces as this induces lower costs and the amount of changes. Second, to generalise the input options for more functionality for future purposes and to minimise the amount of dependencies. As mentioned, interfaces and their descriptions as well as their generalisation will be very important to the modular software policy for future. The modifiability ("permanent adaptation") of software allows the re-design and re-engineering of used components, which will be finally the basement for the whole software evolution [10] (cf. Sect. 2).

3.4 Security

Due to the spread of IT systems within our society it is obvious that those systems have to be protected against any attacks, malware or other virtual and physical threats. As all our important documents, personal data, cash card information or profiles end up in any software products (probably in the cloud based software) it is potentially attractive to hackers, crackers and other malicious purposes. Due to the increasing complexity and scale of the software systems and the vast amount of stored and processed data, the difficulty of providing guaranteed quality of service for security and privacy [13] accelerate and will be a future hurdle for software engineers and architects. It may be a bit ironic that even malware evolves to a more component-based software product and adapts through drivers, as well as other software products, to the new future environment with its non-functional requirements. Malicious attacks, system overloads and natural disasters can have a huge impact on our economy, our national security, our lifestyles and our sense of personal security [14] pointed the future situation out. "[...] software will be THE critical infrastructure on which all other critical infrastructures will depend, smart grids are just the first indicator for this." [13]. To go along with this development, engineers, designers and architects have to implement more robust and resilient methods for their software systems as well as more generic defence methods to cover as much as possible creative attack vectors. As the availability (cf. Sect. 3.2) methods works fine for physical (natural) disasters (*safety* as a non-functional requirement here), resisting, detecting and recovering from attacks or dysfunctions will be more important than today. To resist attacks, established methods for information hiding, authentication, authorisation, integration, encryption or limitation [7] will work fine in future. But there is no way around to increase their level of trust and therefore the complexity, e.g. more digits for encryption keys, saver implementations, two-factor authentication, distributed possible point-of-failures and the avoiding of single point-of-failures, system decentralising, better reputation- and policy-based rules or layer- and ring-based security architecture. In addition to that, future comfort functions will outline new security problems, for example the relinquishment of passwords inputs. We will also need to identify new components in modular software in order to reduce the risk of security leaks and "[...] verify the the correct function of services, while protecting the privacy [...]." [13]. Responsible intrusion detections systems (especially for networks and cloud systems) prevent systems

from attacks, but with the increasing level and development of technology it is necessary to raise the level and range of attack prevention, detection and recovering in the same way. Full automated checks, analysis and autonomous decision making of logs, traffic, sensors, files and other systems will be more viable due to faster reacting to events for critical IT systems.

3.5 Testability

Testability and integrity are some of the qualities that do not expect huge changes in relation to actual handling. The old paradigm to prove and test software with validation and verification does not change that much. Something emphasized in this quality aspect is to test own software with third party components as software should and would be opened to modularity in the future. That can be done with conventional beta testers, test routines or with monitoring functions within the program [7]. It seems very possible, that monitoring of system states will extend while testing via hand would have a lower impact since new software systems are more complex and scale dynamically what makes it difficult to reach all possible use cases. Because it is the task of the modifiability quality (cf. Sect. 3.3) to provide and require defined ports of used components, it will make these stitched systems actually easier to verify and validate. Especially for critical systems (cf. Sect. 3.4), tested software will be a must-have in future systems (for example *Googles* Self-driving car) [6] as this lowers the risk of failure and thereby have direct impact to most of the other important non-functional properties.

3.6 Usability

It is difficult to say something specific about usability as it will strongly develop with future integration of systems and how they interact with social and scientific life. In 2025 the software systems are not designed only for scientific and computer experts purposes only. A wide range of different kinds of humans would use something which has software in it since the progress of smart objects and the Internet-of-things will continue being developed. You can simply not expect that the average user can deal with terminal, parameter or software configurations in the future systems. Especially in the private life environment there is an incredible increase of computational power, but the user does not need to have any IT related skills to use them correctly. For example, nowadays established smartphones and tablets profit extremely from user friendly software design (*Android* [2] as well as *iOS* [4]) as they allow semi-one-click-installations, automated user security group management, limited software manipulation entry points and other architectural methods as well as new (or better) technologies like natural language processing for more comfortable input. As mentioned in [11] the four following measurable properties define the ambiguous quality "usability" very well: 1) *learnability* as how quickly and easily users can begin to do productive work, 2) *efficiency of use* as the number of tasks per unit that the user can perform, 3) *reliability in use* as an attribute referring to the error rate in using

the system and their recover times, and 4) *satisfaction* as the subjective opinion from the user itself via interviewing, poll or something like that. These references would not change in the future that much (you can still say, they should all increase to make future software successful) but are, like the testability quality, somehow important in questions of the target audience and the acceptance of the new software product in the market.

4 Conclusion

This paper has shown a selection of different non-functional requirements and their architecture aspects of software systems. We have seen the trends and drivers of actual software technologies and how they have an impact on non-functional requirements for future software systems. Because these future systems need generally more attention to the majority of the non-functional requirements, none of the mentioned qualities can be left out of consideration. Software architects have to consider the necessary methods and designs for the new environments. To skip a quality described above (or a quality in general) will always lead to some downsides in the software system, for example loss of stability, functionality and comfortability.

There are much more qualities beside the ones described here, like functionality, reliability, efficiency, maintainability [12] and so on. It is difficult to chose a set of qualities and state them as the most important ones, since it depends on the application scope. But as a conclusion and personal view, I believe that the three following non-functional requirements will have the most impact for all domains in the future and every software system will have to deal with: security, availability and modifiability. These three qualities will build the basement of other requirements, especially for huge qualities like performance, usability, testability, safety, privacy, scalability and so on.

Firstly, security is a major challenge for software in 2025 because software will be known in a context of ubiquitous computing. This fact will increase the need of better security and privacy qualities for the consumer and the enterprise. Critical infrastructures and objects have to be secured against developed malicious purposes to avoid costs and maintain trust. This basically means to deploy up to date adapted security-by-design principles and new generic defence methods to secure all kinds of malicious entry points and provide quite good security quality even in future systems.

Secondly, availability is very important for both, the user and the company since both depend heavily on the provided services which therefore should be up all the time ideally. Because future software will be always connected with the web, the provided services rise and fall with the up-times and down-times. This all-time-connection for nearly every system makes it unacceptable to left availability out of consideration. This requires to implement intelligent error handling and fault prevention as well as autonomous decision making.

Finally, modifiability is an important quality for future adaptable software. This quality lowers the development costs and work time, simultaneously it lets

software be variable for other purposes than first implemented. As stated in chapter Modifiability (cf. Sect. 3.3), the requirements of functionality of future software will change during their lifetime. It will not be effective anymore, to build up unmodifiable software from scratch coupled with a five year lifetime as an example. Software has to be build to even connect future devices and should be able to adapt to future technologies to extend its lifetime and acceptance.

In fact the described scenarios are still predictions, it goes without mentioning that everything can take a different direction in further development from the described paths above. Many things can happen, but in general, these predictions seem to be very feasible.

References

1. Adobe Creative Cloud - Cloud-based creative software and services. http://www.adobe.com/creativecloud.html, [Online; Jul 2014]
2. Android. http://www.android.com, [Online; Feb 2014]
3. Apache Hive TM. http://hive.apache.org, [Online; Jan 2014]
4. Apple - iOS 7. https://www.apple.com/ios/, [Online; Jun 2014]
5. Google Glasses. http://www.google.com/glass/start, [Online; Jun 2014]
6. Just press go: designing a self-driving vehicle. http://googleblog.blogspot.de/2014/05/just-press-go-designing-self-driving.html, [Online; Jul 2014]
7. Lecture Notes from Software Architecture. http://st.inf.tu-dresden.de/content/index.php?node=teaching&leaf=1&subject=235&head=2, [Online; Jun 2014] Lehrstuhl für Softwaretechnologie, TU Dresden. Based on *Bass, L., Clements, P. and Kazman, R.: Software Architecture in Practice. (2003)*
8. MapReduce: Simplified Data Processing on Large Clusters. http://research.google.com/archive/mapreduce.html, [Online; Apr 2014]
9. Office Online - Collaborate on Microsoft Office documents, spreadsheets, presentations and more online. http://office.microsoft.com/en-gb/online, [Online; Jun 2014]
10. Bucchiarone, A., Cappiello, C., Nitto, E.D., Kazhamiakin, R., Mazza, V., Pistore, M.: Design for Adaptation of Service-Based Applications: Main Issues and Requirements. Service-Oriented Computing, ICSOC/ServiceWave 2009 Workshops pp. 467–476
11. Folmer, E., Bosch, J.: Usability Patterns in Software Architecture. Department of Mathematics and Computing Science, University of Groningen (2004)
12. Greefhorst, D., Proper, E.: Architecture Principles. Springer (2011)
13. ISTAG, Working Group on Software Technologies: Software Technologies - The Missing Key Enabling Technology (2012)
14. National Academy of Engineering: The Engineer of 2020 - Visions of Engineering in the new Century (2004)
15. Sutter, H.: The Free Lunch Is Over - A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal 30(3) (2005/2009)