DevOpsUse: Community-Driven Continuous Innovation of Web Information Infrastructures

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Medieninformatiker
István Koren

aus

Budapest, Ungarn

Berichter: Universitätsprofessor Dr. rer. pol. Matthias Jarke
Universitätsprofessor Dr.-Ing. Günther Schuh
Privatdozent Dr. rer. nat. Ralf Klamma

Tag der mündlichen Prüfung: 8. Mai 2020

Abstract

Since its invention in 1989, the only reliable factor on the Web has been its continuous change and diffusion into more and more application areas. The evolution was shaped by an interplay of new technologies on the one hand, and innovative application ideas from communities on the other. At a technological scale, alternation between vastly distributed and centralized architectures can be observed. The current challenges caused by the ongoing digital transformation are changing workplace settings and the adoption of the Internet of Things in industrial use cases, as for example in the context of Industry 4.0. On the Web, new technologies and device types sprawl together with new communication protocols and revised application programming interfaces (APIs). This inhibits the demanded rapid innovation cycles and creates a disruptive and unstable environment in which the requirements of endless communities must be met.

Information systems infrastructure, while only partially visible and thus hard to grasp, has a strong influence on user practices. Therefore, the aim of this thesis is to stabilize the dichotomies apparent in the Web by means of an agile information systems development methodology. It supports the evolution of infrastructure through community-driven and model-based technologies to guide it on a sustainable path of continuous innovation. Our DevOpsUse methodology includes users in the process of infrastructuring, i.e. the appropriation of infrastructure during its usage. Agile development practices in software engineering, in particular DevOps, promote stronger cooperation between development and operating teams. DevOpsUse additionally fosters a stronger involvement of end users in software development processes. It intends to empower communities of practice to create and run their own software on their specific infrastructure, with the help of various newly developed software artifacts. The instantiation of our DevOpsUse life cycle model starts with Requirements Bazaar, a Web-based tool involving end users in the idea generation and evolution phases. Direwolf is a model-based framework bridging the gap between technocratic API descriptions created by developers, and user interfaces understood by end users. Faster development times require a streamlined deployment, which we achieve with the software container-based Layers Box. Ultimately, distributed development and operation go hand in hand with our evolutionary analytics platform SWEVA.

The newly developed DevOpsUse methodology with its four areas, all involving end users, has been successfully validated by the transitions between three generations of technologies: near real-time peer-to-peer Web architectures, edge computing, and the Internet of Things. All technological leaps could be adequately mastered and supported by significantly end-user-oriented measures. In particular, we were able to demonstrate our methodology's capabilities through longitudinal studies in several large-scale international digitalization projects. DevOpsUse scalability and involvement aspects were confirmed in entrepreneurial and medical teaching courses. Beyond Web information systems, the framework and its open source tools are applicable in further innovative areas like mixed reality and Industry 4.0. Its broad adaptability testifies that DevOpsUse has the potential to unlock sustainable innovation capabilities.

Kurzfassung

Seit der Erfindung im Jahr 1989 ist die einzig verlässliche Konstante des Web die stetige Veränderung und Ausbreitung in immer mehr Anwendungsgebiete. Diese Entwicklung war geprägt durch das Zusammenspiel von neuen Technologien auf der einen Seite, und innovativen Anwendungsideen aus Praxisgemeinschaften auf der anderen. Aus technologischer Sicht war dabei ein Wechsel von stark verteilten und zentralisierten Architekturen zu beobachten. Die gegenwärtigen Herausforderungen der fortlaufenden Digitalisierung sind veränderte Bedingungen am Arbeitsplatz sowie die Einführung von Technologien für das Internet der Dinge in industriellen Anwendungsfällen, wie beispielsweise im Kontext der Industrie 4.0. Parallel breiten sich im Web neue Technologien und Gerätetypen zusammen mit neuen Kommunikationsprotokollen und überarbeiteten Programmierschnittstellen aus. Erforderliche Innovationszyklen hemmend entsteht so ein instabiles Umfeld, in dem dennoch Anforderungen diverser Praxisgemeinschaften erfüllt werden müssen.

Obwohl Informationssystem-Infrastruktur nur begrenzt sichtbar und damit schwer zu fassen ist, hat sie dennoch einen wesentlichen Einfluss auf Nutzerpraxen. Daher ist das Hauptziel dieser Dissertation, die im Web vorhandenen gegenseitigen Wechselbeziehungen durch eine agile Informationssystem-Entwicklungsmethodik zu stabilisieren. Diese unterstützt die Entwicklung der Infrastruktur durch gemeinschaftsorientierte und modellbasierte Technologien, um einen nachhaltigen Kurs der kontinuierlichen Innovation zu ermöglichen. Die DevOpsUse-Methodik basiert auf einer fortlaufenden Einbeziehung von Endanwendern im Sinne des *Infrastructuring*, d.h. der Veränderung der Infrastruktur während ihrer Nutzung. Agile Entwicklungspraxen aus der Softwaretechnik, im Besonderen DevOps, propagieren eine stärkere Kooperation zwischen den Abteilungen Entwicklung (*Development*) und IT-Betrieb (*Operations*). DevOpsUse fördert darüber hinaus eine höhere Beteiligung der Endanwender an Softwareentwicklungsprozessen. Insbesondere soll es Praxisgemeinschaften in die Lage versetzen, ihre eigene Software auf ihrer spezifischen Infrastruktur aufzusetzen und zu betreiben, mit Hilfe diverser neu enwickelter Softwareartifakte. Das DevOpsUse Lebenszyklus-Modell beginnt mit Requirements Bazaar, einem webbasierten Tool, das Endanwender in die Ideenfindungs- und Entwicklungsphase einbezieht. Direwolf ist ein modellbasiertes Rahmenwerk, das die Lücke zwischen technologieorientierten Schnittstellenbeschreibungen und Benutzeroberflächen schließt. Kürzere Entwicklungszeiten erfordern eine straffer organisierte Auslieferung von neuen Softwareversionen, die wir mit der containerbasierten Layers Box erreichen. Im letzten Schritt ergänzen sich die verteilte Entwicklung und der Betrieb mit der evolutionären Analyseplattform SWEVA.

Die DevOpsUse-Methodik mit ihren vier Bereichen der Endanwenderintegration wurde erfolgreich über die Übergänge zwischen drei Technologiegenerationen validiert: der Nahezu-Echtzeit-Kommunikation in Peer-to-Peer Architekturen, Edge Computing, sowie dem Internet der Dinge. Diese technologischen Entwicklungssprünge der letzten Jahre konnten adäquat gemeistert und durch endanwenderorientierte Maßnahmen unterstützt werden. Insbesondere konnten wir die Leistungsfähigkeit unserer Methodik in Langzeitstudien in mehreren internationalen Digitalisierungsprojekten unter Beweis stellen. DevOpsUse Aspekte wie Skalierbarkeit und Nachhaltigkeit wurden in Lehrveranstaltungen zu Themen der Unternehmensgründung und Medizin bestätigt. Über Web-Informationssysteme hinaus sind die entwickelten quelloffenen Anwendungen in weiteren innovativen Bereichen wie der erweiterten Realität und Industrie 4.0 einsetzbar. Die breite Anwendbarkeit verdeutlicht das Potenzial von DevOpsUse, nachhaltig Innovationskraft zu entfalten.

Acknowledgments

This dissertation is the result of my longest journey so far. Countless people helped me in one way or another along the path, and my thanks go to all of them. Nevertheless, I would like to pay special tribute to a few of them.

First of all, I would like to thank my dissertation committee. Professor Matthias Jarke created an environment at Informatik 5 where we can work on ideas in our groups; he pushed me forward at the right times, and continues to trust me and my research goals in an interdisciplinary context. I thank Professor Günther Schuh for giving me an outside perspective as second supervisor. Ralf Klamma had the confidence to hire me as a doctoral candidate. He gave me the opportunity to strengthen my profile at numerous project meetings, including important reviews, as well as conference travels. The management principles of research and teaching within the Advanced Community Information Systems group helped me to organize myself in the long run. I am especially grateful, that despite the global COVID-19 pandemic, my whole dissertation committee was physically present at the exam, obeying social distancing rules. I thank my examiner Professor Wil van der Aalst and the head of exam Professor Hermann Ney for being part of the committee, and for the relaxed atmosphere they contributed to.

I am glad to have had such open and supportive office mates. Zinayida Kensche eased my start at RWTH and, with her family, became a friend I can still count on. The same is true for Sandra Geisler, who over the last years witnessed and defused many frustrating situations, which of course were not too rare. Yet, where would I be without my colleagues from ACIS. Dominik Renzel and his legacy of research diligence (and scripts!) was a role model in integrity. Furthermore, the collaboration and fun with the rest of the ACIS crew, Manh Cuong Pham, Khaled Rashed, Dejan Kovachev, Anna Hannemann, Katya Neulinger, Petru Nicolaescu, Mohsen Shahriari, and our postdocs Milos Kravcik and Michael Derntl made life more enjoyable. I thank Georgios "Yogi" Toubekis and Stefan Schiffer for advising me on several matters. My colleagues Reinhard Linde and Tatjana Liberzon made all technical demands possible in no time. Our team assistants Daniele Glöckner and Claudia Puhl supported me not only with administrative issues, but much more than that, with pleasant conversations. On the last mile, Leany Maaßen and Romina Reddig backed me just as sincerely.

My ambitious goal of an overarching assessment of state-of-the-art technologies would not have been possible without the numerous bachelor and master students I had the honor to advise. My thanks therefore go in chronological order to Jens Bavendiek, Emmanuel Biver, Andreas Guth, Denis Golovin, Barna Zajzon, Ankit Ramani, Tauqeer Ahmad, Alexander Ruppert, Melvin Bender, Philipp Bartels, Yu-Wen Huang, Manuel Gottschlich, Benedikt Hensen, Justin Krause, Rizwan Ali, Delcy Bonilla Oliva, and Dominik Kus. Their relevant contributions are cited at the respective places throughout this thesis. The student assistants Gordon Lawrenz, Kristjan Liiva, Ádám Gavronek, Yordan Manolov, Angel Astorga, Anna Piunova, Bujar Bakiu, Albi Sema, Aldo Myrtaj and in particular Martin Hug and Kevin Jahns helped me to expand prototypes and keep them running continuously. Acknowledgments go to all my funding projects: Learning Layers (FP7), WEKIT (H2020), AR-FOR-EU (Erasmus+), and recently, Internet of Production (DFG). They enabled me to cross many borders and boundaries, and at the same time, proved to me how great it is to work and research in Europe, especially as a true European by background and soul. I have made many new contacts in the projects and during my travels, many of which carried on not only

on a professional but also on a personal level. The same applies to my scholarships from DAAD and JSPS. On my internships and research stays I was tremendously supported by Markus Endler, Yann Jouanique and Shohei Yokoyama. I would like to thank my bachelor and diploma supervisors from TU Dresden Thomas Springer and Daniel Schuster for supporting me early on; it meant a lot to me that both of you joined my presentation.

The thesis is also the result of hard work with the worst work–life balance I could have ever imagined. Many people do not understand that quality of life often decreases with increasing distance from family and lifelong friends. We were all heartbroken that the defense talk had to be done remotely due to the pandemic. I was overwhelmed and very proud that such a large crowd of my companions made their way to the remote presentation and the subsequent Zoom party.

Many thanks to Uta Puster for introducing me to Aachen at the first place, Anja Hohmann for supporting me as virtual paranymph, and Andreas Pursche for giving me valuable advice from industry and experiences from exotic locations. I want to thank my family for accepting my limited availability afar and giving me space to pursue my goals. My parents laid the foundation for my interests early on, and showed me the world. They supported me from Budapest, my sisters Zsófi and Anna kept everyone together in our geographic center Leipzig, and my brother Dani in Cologne gave me insights into a completely different world. Finally, I would like to thank Eszter for everything you have done and went through to support me and covering my back.

Aachen, July 1, 2020
István Koren

Contents

Chapter 1

Introduction

In history, no innovation has ever impacted both private and professional environments as disruptively and rapidly as the progressive spread of computer technology. The role and societal dimension of the invention of the Web was thereby perhaps only matched by the introduction of moveable type by Johannes Gutenberg to Europe, starting the era of mass communication. Today, the digital transformation presents our society with many challenges on various levels, from political and regulatory questions to educational principles. Common roles and responsibilities no longer count; companies and their products emerge from nowhere at an incredible pace, and many disappear just as quickly. Within industry, an economic sector particularly strong in Germany, digitalization is often called the fourth industrial revolution, or *Industry 4.0*. This term refers to a paradigm shift currently taking place in industrial production, towards the use of a combination of Internet and future-oriented technologies [LFK*14b]. On the one hand, triggers are social, economic and political changes; on the other hand, a number of technologies like mobile apps, 3D printers and the Internet of Things (IoT) pushes innovation in industry. This is accompanied by a variety of changes, e.g. new materials, tools, as well as hardware and software systems.

The societal impact is obvious. In an ever faster spinning world, new skills are required on an almost daily basis, with profound consequences on the established educational system of primary, secondary and tertiary education.This also has manifold effects on information systems (IS) development and provisioning. Faster innovation cycles in software development rendered the usual "Microsoft Office" model of software provisioning obsolete. In that model, software was released in cycles of 1-3 years with small security updates in-between, if the powerful big players considered them necessary. The radical shift in speed becomes particularly evident when considering the update rates in the newly established app market economies of mobile operating systems: 14% of apps are updated at least bi-weekly, with apps in the "Social" category being the most revised [MAHa15].

One of the main adjustment parameters to control the velocity of releases, as it has been found over the last years, is the software development methodology, that is, how software engineering is carried out, following the planning, actual implementation and roll-out phases. A great deal has already been achieved in this area in recent decades: The shift from inflexible waterfall models to agile environments has made a significant contribution towards dealing with change. However, the current, further massive spread of information technologies into our living environments will further intensify the problem in the future, and it already does.

In addition to the variety of new hardware, the increasing number of users also plays a key role. This number harbors both opportunities and risks, of which the overload of network infrastructures is only one. The substantial integration of end users in the decision and even development processes has not been fully tackled yet. For instance, while the main driver of agile technologies is the urgent need to meet end user demands faster, the methods of end user participation are not closely integrated with implementation, since in prevalent project management models like Scrum, the user often only appears at the beginning and at the end. This leads us to the question of how existing software development processes can gain a holistic view of the entire community involved, explicitly integrating groups such as end users, designers, developers and operators. What does a supportive information infrastructure look like that can withstand and survive these fast development cycles?

The approach described in this dissertation considers all stakeholders as a Community of Practice (CoP) [Weng98]. The inclusive development process focuses on the individual strengths and potential weaknesses of the people involved. In particular, we analyze how to address these issues with the Web as socio-technical system. There are manifold reasons to choose the Web as the main object of observation. The Web, seen as a *graph of linked resources*, provides a unique opportunity as a testbed for various interaction styles and communication patterns [PaZi17]. In what follows, we refer as "Web technology" to all artifacts that are accessible from a standard browser of the latest generation[1]. The Web's influence and outreach are undisputed. It is now available everywhere, beyond hardware boundaries. This includes for example entire industrial machines, desktop computers, mobile phones, small computers and wearables such as smartwatches and activity trackers. A constantly evolving body of *standards* thereby ensures interoperability between manufacturers and devices. The first 30 years since 1989 were only the beginning; we are convinced, that the existing procedures will further power the integration of manifold user interaction means and information system manifestations.

The research background, societal context and evaluation testbeds of this dissertation are mainly located in the area of technology-enhanced learning (TEL), based on its funding in the context of several large-scale international research projects in TEL. The resulting DevOpsUse methodology emerged from and was employed in several collaborative integrated digitalization projects that manage societal processes, thereby stressing sustainable development. The targets of these collaborations in turn moved within the above-mentioned framework of Industry 4.0.

1.1 Problem Description

One of the Web's key strengths is also among its weaknesses: the continuous context changes do not only increase the Web's applicability and adoption, but also require constant retraining of users, developers and operators in order to handle the new realms. What is therefore required is a specified yet flexible infrastructure at its core, on top of which possible innovative applications and use cases can be rolled out. This infrastructure needs to be inherently open, in the sense of not being ruled by a central institution, be it a government or an overpowering corporation. Instead, all development

[1]The class of *latest generation* browsers is often named "evergreen browsers". The term refers to the fact that the browser is constantly updating itself at regular intervals. As we will see in later chapters, this update frequency is in itself only possible through a sound information system infrastructure.

should be community-driven and reach even end users with domain knowledge. Ideally this should happen to an extent that users can develop software themselves, or at least improve their agency by being able to install it, in order to try out new service and application configurations; figuratively assembling their own tool belt.

The interoperability is however opposed by a number of dichotomies within information system infrastructures; the main ones are represented in Figure 1.1. On the left side, there is the frequent transition between centralized versus distributed service infrastructures. Often imagined as a pendulum, history has gone from large mainframes to personal computers, and back to concentrated cloud data centers, with recent interest in edge-oriented service computing. On the right side of the figure, end users stand their ground against developers. While the end users, as mentioned above, possess domain knowledge of *what* they want to be developed, the realization and the *how* is currently often only possible with extensive knowledge of development environments and specialized understanding of programming languages, databases and tooling. Other dichotomies with a social dimension found for instance in research projects are researchers and application partners: While researchers strive for publications in prestigious journals and conferences for fulfilling the ambitious key figures of their research proposals, companies acting as application partners are mainly interested in economic success of the innovative solutions tried out in combination with research. All these dichotomies are embedded into a changing world where device innovations open up and magnify the solution space. New types of devices in the areas of Internet of Things and wearable computing pose new chances but also challenges. On a societal dimension, digital transformation changes practices of employers, shifting formal learning in classrooms to informal learning settings at the workplace. Further strains are coming for instance from ethical and legal aspects that also need to be taken care of in a holistic framework.

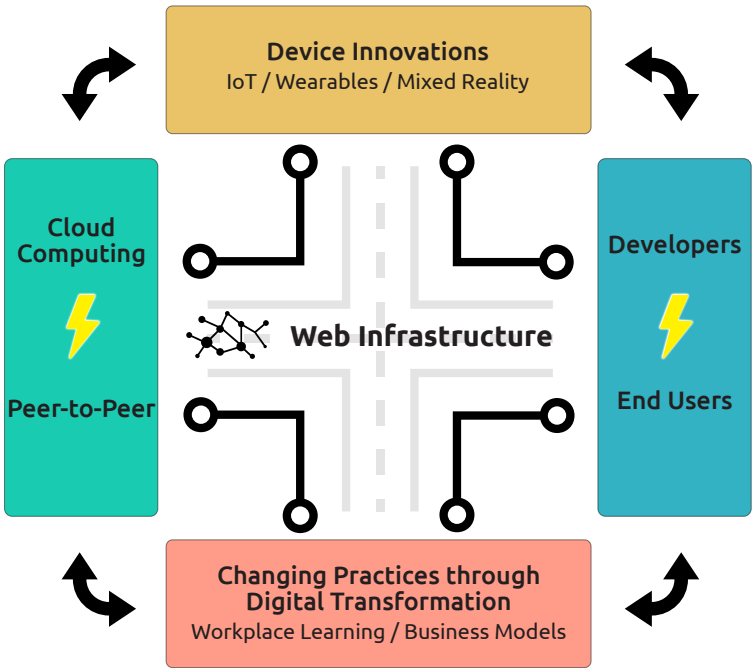


Figure 1.1: Dichotomies in Information Systems Engineering

Altogether, the challenge is to find the constant that overcomes these dichotomies. Our main hypothesis is that this constant is provided by a suitable information infrastructure. Its precise design is the subject of extensive research. This leads us to the research questions discussed in this dissertation that are introduced in the next section.

1.2 Research Questions

Concerning the continuous technological shifts and their impacts on professional communities of practice, this dissertation addresses the following main three research questions:

Research Question RQ1: What are the building blocks of information infrastructures in information systems engineering? What are the fundamental technical elements within the core and on the edge of the infrastructure? How to stabilize the continuous dichotomy between centralized and distributed (peer-to-peer) architectures?

Research Question RQ2: How to enable CoPs to develop information systems on community-specific infrastructure? What are the boundary objects between the community and its infrastructure? Which conventions and formalisms map the worlds of end users and developers as a common denominator? Which are the tools end users need? How can model-driven engineering improve the collaboration of end users and developers?

Research Question RQ3: How to establish a continuous innovation life cycle that enables sustainability of infrastructuring and the created information systems? What are methodological means to operationalize the goals of communities? How to scale the innovation capability and inherent creativity of end user communities?

1.3 Research Methods

This dissertation was carried out within the information systems and databases institute at the computer science department at RWTH Aachen University. It was embedded into the Advanced Community Information Systems group and was written in the context of several European funded, large-scale research projects in the context of technology-enhanced learning. As such, we followed the research tradition of the community-based ATLAS methodology [Klam10c]. It is correlating to the design science methodology in terms of several feedback loops running through development and summative and formative evaluation cycles using the methods and tools developed in earlier doctoral theses [Kova14, Hann14b, Renz16]. Our central hypothesis is that the Web as socio-technical system establishes a common ground as infrastructure to solve many of the challenges encountered within professional *Communities of Practice* (CoP). As CoP according to Wenger, we understand a group of people, who mutually engage in a joint enterprise, with a shared repertoire [Weng98]. We advance the ATLAS methodology in terms of scaling it from a single community under observation to a multitude of communities with the Web as cross-cutting concern. To achieve this goal, we borrow concepts from global-scale agile development practices, in particular

DevOps as a mindset that in the last decade helped to stabilize sped-up development cycles by better communication between development and operations departments. In Section 2.4 we therefore introduce *DevOpsUse* as central methodology and Community of Practice consisting of developers, operators, and users. We analyze the main stages of the DevOpsUse development process with the special premise of end user integration. Along the life cycle, we employ various means like model-driven engineering, end user development and visual analytics.

Any infrastructure commonly works best when it is hidden, when its structure becomes invisible as a black box. Our research therefore faces the challenge of making individual parts visible in order to test adjustments and explore new areas. As a consequence, we do not provide one single design artifact answering all our research questions with a holistic analysis and general evaluation. Instead, we dive deeply into the particular aspects and provide boundary objects taken on by the next step in the life cycle. Conceptually, we perform *infrastructuring* by providing tools that empower their users to shape their own information system design [StBo02, PiSy06]. We consider this to be a fundamental principle for enabling a sustainable information system design and development process.

Our research follows the conceptual framework and guidelines of design science in information systems research by Hevner et al. [HMPR04]. The authors propose seven guidelines that revolve around creating an artifact "in the form of a construct, a model, a method, or an instantiation". In the core Chapters 3–6 we each present particular artifacts and run them through the design science research process. Hereby, the procedure is twofold. On the one hand, the artifacts are advanced on their own, on the other, certain technological aspects are cross-cutting concerns whose elements we revisit in multiple chapters. For instance, Section 3.2 presents *Requirements Bazaar* as continuous innovation tool that we used for integrating end users in the requirements engineering of information systems. It itself was developed iteratively. The first version served as proof-of-concept, the second was then employed in a research project. Finally, the third and current version answered usability issues, but also advanced developmental challenges by relying on las2peer as service framework for easy extensibility. The service's reliance on open REST-based interfaces, however, also enable its usage in innovative new technological environments like mixed reality, as we will see in Section 3.4. Therefore, we consider these artifacts as boundary objects of both our research approach and the overarching DevOpsUse methodology. The individual artifacts jointly builds our provided infrastructure, that ensures that information systems development remains innovative despite frequent changes in the underlying technologies.

1.4 Thesis Contributions

In this dissertation, we develop a methodology and tool support that stabilizes information systems evolution in a changing environment. It is driven by an agile community-driven development life cycle called DevOpsUse which in turn is inspired by state-of-the-art software development coming from industry. In particular, we show how the Web inherently answers questions by its standardization processes and technological foundations.

We contribute several artifacts for each aspect of the life cycle. They are implemented using open interfaces and available as open source resources on the Web. We regard their interconnection and

orientation towards end users as core contribution of this research. While each chapter identifies many individual related work and available information systems, which deal with a particular aspect in great detail, our research revealed a lack of interoperability across aspects. In the presented approach we have therefore simplified and linked all individual aspects in a general and interoperable manner, driven by open Web standards and practices. The entirety of all tools and their interoperability through the Web make up the main contribution of this thesis. Figure 1.2 depicts the DevOpsUse life cycle and links its phases to the core chapters. The main design artifacts are listed in the individual boxes surrounding the methodology model.

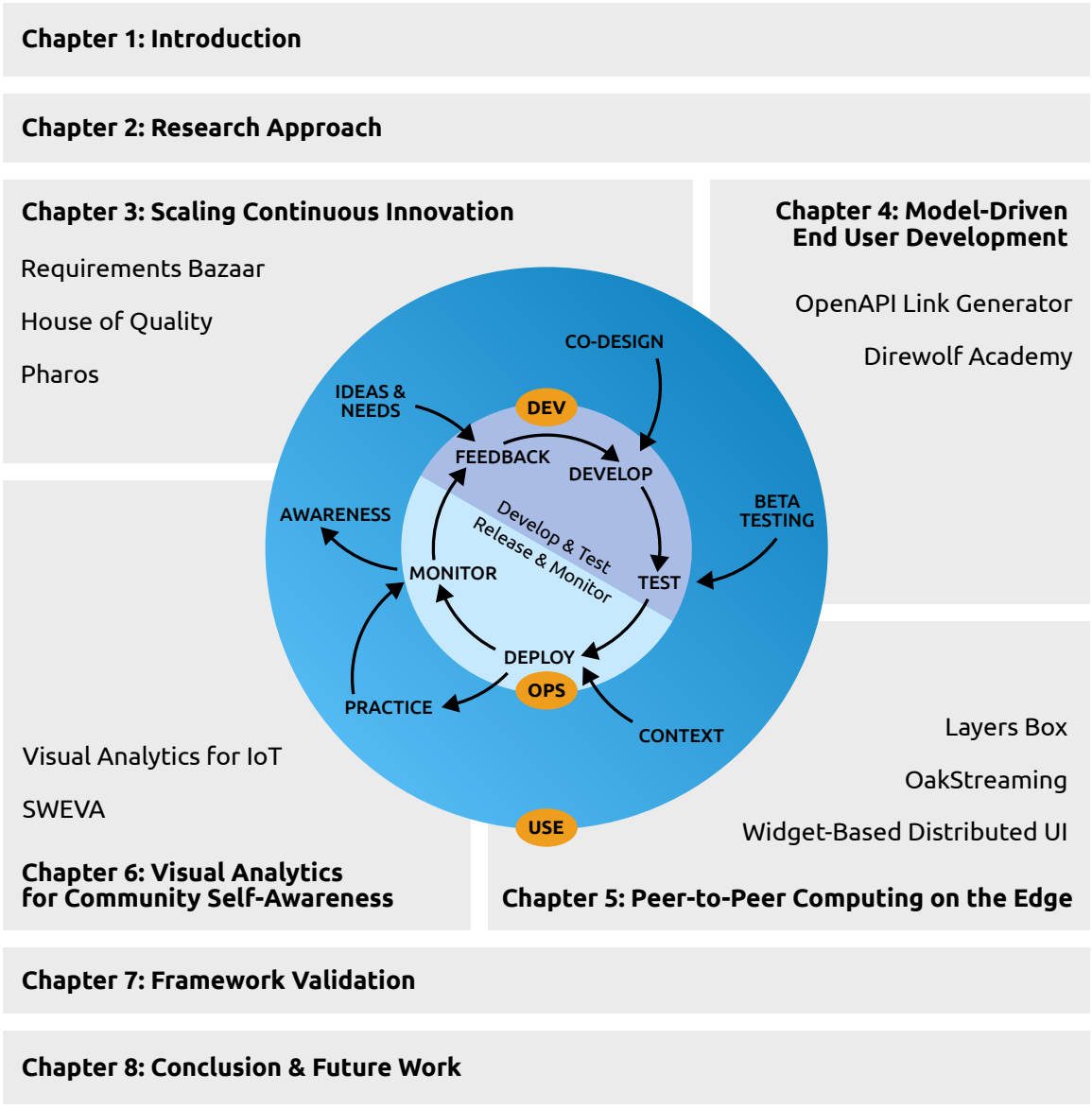


Figure 1.2: Thesis Organization and Main Design Artifacts Following the DevOpsUse Life Cycle

1.5 Thesis Overview

The remainder of this dissertation is organized as follows.

In Chapter 2 we introduce the research approach. First, we clarify our understanding of *infrastructure* and *end user*. We then formulate the *DevOpsUse* methodology based on an extension of *ATLAS* concerning the aspect of scalability.

In Chapter 3 we start the DevOpsUse life cycle by introducing continuous innovation as an agile method of integrating end users into software engineering.

In Chapter 4 we extend the end user integration to a model-based development process. Thereby, we present formalisms that ensure cross-compatibility between various service interface standards. We present the Direwolf framework for generating cross-device Web user interfaces.

In Chapter 5 we present component-based software engineering for deploying service containers onto an edge–computing-oriented cloud system. Subsequently, we show how Web frontends can leverage peer-to-peer computing for diverse innovative use cases.

In Chapter 6 we discuss how to empower communities to analyze their information systems usage themselves. We propose SWEVA as universal tool for creating highly interactive visual analytics, universally accessible on the Web.

In Chapter 7 we validate the DevOpsUse framework by presenting how it mastered three technological leaps over the last years. We then make practical recommendations on how the methodology can help overcoming issues in large-scale research projects, taken from experiences therein.

In Chapter 8 we summarize the dissertation by presenting the most important contributions regarding the research questions. Finally, we look at how DevOpsUse shapes use cases beyond agile community-driven software development, before we light the way towards promising future work.

Chapter 2

Research Approach

Information systems have an undisputed impact on society. Any attempt of supporting the development and operation therefore requires a sound methodological approach, in particular with the radical pace of change mentioned in the previous chapter. In this chapter, we give an overview of the theoretical background and research methodology of this dissertation. The research is embedded into the work at the Advanced Community Information Systems (ACIS) group at the Information Systems and Databases department of RWTH Aachen University. Following the research tradition of our group, this entails a number of implications that we consider in the following. We first describe the overall concept of *infrastructuring* as *in-situ* design work, or *design-in-use* as opposed to *design-before-use* [PiWu09]. It revolves around the notion of an infrastructure as an underlying platform. Any activity on top of it uses it but also shapes its direction. We perform infrastructuring using the ATLAS methodology [Klam10c], which is looking through a perspective centered around professional Communities of Practice (CoP). Starting from it, we look at means of scaling the approach from one community to a large number of communities. When we examine industrial practice, we see that automation is a popular and effective means of scaling. One example is the common DevOps methodology, which describes how developers and operations teams work together to get software rolled out faster. Compared to ATLAS, that revolves around users as community members, it quickly becomes evident that the DevOps model largely neglects the involvement of end users. Although the model is based on user-oriented agile principles, the user is only involved in the model at the beginning and end. Therefore, tools that are conceptually aimed at supporting DevOps do not take into account even shorter feedback loops during development, for example caused by user tests executed in parallel. To this end, in this chapter we look at which other aspects are already common practice in software engineering, in order to develop tool support in later chapters. Based on ATLAS, we present *DevOpsUse* as central methodological core of this dissertation. Our methodology is inspired by agile software development methods observed in industrial practice, and evolved in large-scale digitalization settings. We extended DevOps by this particular notion and its many individual facets. In the chapters hereafter, we show how integrating end users helps in increasing their agency and thereby sustaining development. Within each of these chapters, we included dedicated related work sections that go in detail about the respective topic. Herein, we review the most important literature for the overarching aspects.

In Section 2.1 we start with an overview of the main theme of this dissertation, which is infras-

tructuring. We also define our understanding of *end users* based on literature on *end user development* [LPKW06] in Section 2.2. Section 2.3 then presents the community-driven information systems development methodology, pioneered within the ATLAS methodology developed at the chair. Taking DevOps into consideration, we combine both theories of end-user development and agile practices in the life cycle model DevOpsUse in Section 2.4. We present technology-enhanced learning as testbed for the various application prototypes developed within this dissertation in Section 2.5. The technological context of the Web is given in Section 2.6. Finally, Section 2.7 discusses the implications of DevOpsUse on production engineering.

2.1 Infrastructuring

According to the American Heritage Dictionary of the English Language, an infrastructure is "an underlying base or foundation especially for an organization or system" [Amer18]. In this respect, it can be regarded as a building block on top of which a system can be used, and extended. A common context of the term infrastructure can be found in road traffic, and in the water supply system of a city. For the former, streets, highways and bike lanes represent specific (sub-)parts of a greater whole. Cars can be driven on top of streets and highways, but the law forbids driving cars on bike lanes. To distinguish between intended and unintended use of infrastructure, drivers take implicit and explicit properties into consideration. For instance, on the one hand the form factor of a paved road, i.e. the width and the color, may implicitly indicate that it is not safe for bikes. Explicitly, road signs indicate the purpose of the specific road. This example shows that such signs need to be standardized in order to be uniformly recognizable as such. The road system also explains another aspect of our understanding of infrastructure. While a highway under construction is the work item of road workers, civil engineers and politicians, it quickly becomes an accepted (and taken for granted) infrastructure once the road has been opened to general traffic.

In information systems engineering, the transition from build to use is intrinsically much faster than on construction sites, as bricks-and-stones artifacts entail inherently more difficult logistics. What is at one point in time still the subject of discourse, for example a software updated in development, can be rolled out to billions of users in just a few seconds. Additionally, this process may be executed several times a day. Thus, information systems are a particularly interesting and worthwhile object of investigation concerning infrastructure research. Star & Ruhleder "(...) ask, when - not what - is an infrastructure" [StRu96]. In this context, Star and Bowker coined the verb "to infrastructure" in order to emphasize the conditional, flexible and open character of the infrastructure design process, blurring boundaries between design, use, tailoring, maintenance, and reuse [StBo02]. Infrastructuring is the set of activities performed within the design process [PiSy06] and thus should always be seen in relation to everyday practices [StBo02]. Other authors see *design* in general as a process of infrastructuring, as designers and users are working together to change a project, and as technology and people are brought together to be changed [BDIv17]. The social environment around a project is generally included in the common notion of infrastructuring. "Infrastructures shape and are shaped by the conventions of practice" [StBo02]. Therefore users should be given more agency to be more engaged and invest a significant amount of time in the development process instead of just giving minimal input by answering questionnaires. The term infrastructuring interweaves with the concept of design. Professional designers actively involve users to collect ideas

in order to enable them to express themselves [SMKo11]. Because user satisfaction is one of the key quality metrics of a product, users are a valuable source of ideas [KMMa02], thus giving them influence on the development process is worthwhile. *Participatory design* and *collaborative design* (co-design) refer to the inclusion of users within a development team. The goal is to let users help setting design goals and planning prototypes [CCRN00]. Co-design aims to meet the needs and preferences of users for a specific service or project [BrMa08]. It not only allows users to express their opinions on predefined questions, but also helps to identify problems that require a solution.

There are many minor and major aspects that decide whether a design is considered "good" or bad. It has to answer the needs of users and fulfill certain usability aspects. Thereby, even small decisions like the positioning of a button within a context menu can have an impact. When designers are isolated from users, their experiences and needs, a communication gap can occur. This gap between users and designers is one of the major challenges in design [SaSt08]. What is needed to overcome this gap is to give a voice to users so that they can express their needs. Infrastructuring thereby is not only a phase at the start, but an ongoing work and process. In the sense of the actor-network theory we can therefore describe it as a continuous translation, i.e. punctualization as black box [LaHa99, Tuom01].

In participatory design, the participation of users creates relationships between users and their experiences, but also towards new technologies across different domains [Bodk15b]. As social relations, these relationships are constantly evolving, thus the focus needs to be readjusted. Likewise, communities are under constant change, so their designs are expected to evolve with them [BDIv17].

Three infrastructure dimensions are recognized by Pipek and Wulf [PiWu09]: size/globalness/localness, longetivity/sustainability, and people. People are described as own dimension, as complex systems like infrastructures usually require different perspectives. Moreover, the users are needed to understand the appropriation of systems to either reflect new practices or features, or to eliminate the cause of breakdowns [YoLu17]. With the above-mentioned dimensions in mind, Pipek and Wulf highlight, how the separation of design and use can affect the results as the surroundings of the product have been neglected [PiWu09].

Due to their inherent characteristics, information systems provide the possibility of performing both *design-before-use* and *design-in-use*. Hereby, change is an aspect of everyday practice and not a privilege of professional designers when planning or building the system. Instead, Karasti and Syrjanen [KaSy04] emphasize the need to redesign and express that "if technologies are to be made useful, practitioners must effectively take up the work of design. Technological infrastructures should always be seen in relation to organized human practices, as parts of social systems". Moreover, we need to distinguish between *design-for-use* and *design-for-future-use*. The first one is concerned with giving a solution for existing, known problems. The second one anticipates functionalities that are not yet foreseen. This is the essence of infrastructuring, as designs need to be created with a broader view, adaptable to future scenarios. For this, individuals are brought together to discover yet unknown issues, through experience sharing and social relations, creating a foundation to sustain a community and the relations within [DaDi13].

Pipek and Wulf [PiWu09] identify the goals of infrastructuring as (1) advising users to perform frequent tasks to produce improvement, (2) provide the adequate tools to perform such tasks and (3) prepare and engage in interactions with the system owners. According to Ehn [Ehn08], infrastructuring can also be understood as *future design*. Infrastructuring can be different in nature. For

instance, workshops where designers work together with users are a popular form of participation. It can be considered a form of infrastructuring because it connects existing knowledge of a structure or project and different experiences with physical and digital infrastructures [BDIv17]. Pipek and Wulf furthermore list different kinds of support needed for infrastructuring: basic technological support, articulation support, historicity support, decision support, demonstration support, observation support, simulation support, exploration support, explanation support, delegation support, and (re-)design support. They define the point of infrastructure as "the moment when an infrastructure becomes visible to its users", i.e. when the user crosses the border from using to reflecting or even modifying the technology. This is the point where design changes from before-use to continuous re-conceptualizations or in-situ.

Communities of practice work upon infrastructures, as such they need them to exist [Enge05]. In particular, for remote collaboration, there is an increasing need to create platforms supporting the distribution, exchange, production, consumption and accessibility of ideas within and between communities. Communication is therefore considered part of infrastructuring as bridges between actors and resources in different contexts and practices [MaBo17]. In the literature, *continuity* is considered important within communities. According to Karasti et al. [KaSy04], continuity creates the trust within the community "to interact regularly, maintain reciprocity and collaborate in developmental undertakings". Infrastructuring therefore encompasses to provide the means for discovering and expressing issues and its consequences, to enroll the whole community into the cause [DaDi13]. Marttila and Botero emphasize the need to link active communities as part of the infrastructuring process, resulting in various technical and social interdependencies [MaBo17]. These interdependencies make it possible to define and specific features and functions required by the new system. Infrastructuring supports the formation of communities and reifies relations within the community or between communities through the understanding of their technologies [LTT*17]. The collaborative aspect of infrastructuring within communities is described by Hanseth and Lundberg [HaLu01]. They focus on the following four aspects:

- Infrastructures act as shared resources for a community.
- Different components of an infrastructure are integrated through standardized interfaces.
- There is no strict limit between what is included in the infrastructure and what is not, and who may use it and for which purpose.
- They are heterogeneous, consisting of different kinds of components, human as well as technological.

For completeness, *speculative design* should also be mentioned in this context as critical design practice that considers the implications of applying technology in everyday life by envisioning possible scenarios [DuRa13]. On the one hand, these speculative design concepts can contribute to a better understanding of embedding innovative technological concepts such as wearable computing and mixed reality (cf. Section 3.4). On the other hand, within a community context, these can also be driven by special requirements from the long tail (cf. Section 3.2).

2.2 End User Development

In the analogy of car traffic mentioned in the previous section, we described how drivers use a highway infrastructure by driving around on it to reach their destination. Roads such as country roads and highways are thereby regarded as given by the driver. Personal preferences such as to avoid toll roads, may have been or may have been not entered into assistive systems like the navigation computer. Thus the driver guides the car along the most suitable route possible. In this context, it is evident, that the driver of the car is the *end user* of the road infrastructure. All services and restrictions revolve around the car user and are adjusted to him or her. Clearly, the motorist is not a singular stereotype and possibly carries around further people or goods. That is why highway service areas operate shops for truck drivers in parallel to playgrounds suitable for families. Transferred to information systems, there is a much more diverse landscape of connected services. However, the transition between developers and end users of an information system infrastructure is much more fluid and can be performed substantially faster. For instance, thanks to the various editing options built into modern Web browsers, a pure viewing object, such as a static news website, can be made editable within a few seconds. At the same time, information system usage is much more transitive compared to highway usage. For example, developers constructing an information system can also be considered users: of their operating system, their integrated development environment or their continuous integration software. As such, we see the user as the instance, that is perceiving a given infrastructure as transparent. To clarify our understanding of the term *end user*, we first define it. Since we move in the research field of information infrastructure, it is obvious to use a classification from this domain. The related research area of *end user development* (EUD) thereby provides one of the most cited definitions by Liberman et al.:

> End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [LPKW06].

Here, end users are described as individuals that *act* as non-professional software developers. Thus, they fulfill the particular role of developers at the point where they *create, modify or extend* software. EUD is needed to counterbalance those who produce and those who consume in our modern information society. The authors see end user development at the crossing of human-computer interaction, software engineering, computer-supported cooperative work, and artificial intelligence. Wulf and Jarke argue on the economical side, that while the initial cost of software development with EUD is more expensive, it has the potential to subsequently lower the costs of adaptation, as it encourages the exploitation of opportunities for organizational appropriation [WuJa04]. Newer literature relates the field of end user development to the areas of end user programming and end user software engineering. Therefore, when we refer to end user development, we refer to these related areas as well, and draw from the literature of these three fields, published by the research communities in the aforementioned areas. Therefore, the results of this dissertation were published in all of these communities, with the exception of artificial intelligence.

Liberman et al. see two main possible realizations of end user development. The first is parametrization of existing software, i.e. the adaptation of software by their end users to their specific needs. The second is the creation from scratch, where not only existing software is changed and extended,

but new software is developed from the ground up. They see EUD as being concerned more about the second option. Examples of concrete activities fulfilling the paradigm of end user development are programming by example and model-based development. Within software engineering research, models are particularly interesting, as they provide an abstraction to the concrete software. Applied on EUD, we see models as the smallest common denominator between the worlds of developers and end users. Conceptual descriptions of end users can be used by a system to generate code that can be later on extended by developers [Pate00]. In Chapter 4 we present such a system. The technological advances in the last years have made it possible to implement various aspects of end user development. As mentioned above, the Web browser itself can be used as editing platform. While for decades, executable applications running on the Windows or Mac operating systems had to be first developed, then built into a binary format and finally run, Web applications can now be edited during their runtime, without the need to restart the browser. It just needs to re-evaluate the HTML markup, CSS style code and JavaScript language. As another leap towards exchangeable components on the Web frontend, we will introduce *Web Components* as plugin mechanism in Chapter 4.

While technological advancements made it less cumbersome for end users to participate in the development of Web applications, the task requires investments of users in terms of time and expertise, and last but not least, willingness. The current mismatch between the number of developers versus the amount of end users will however become much more evident in future, with the spread of computing systems into all areas of our lives. The affinity for technology is increasing among younger generations. In literature, there are various approaches that explain the willingness of end users to participate in these processes. The term *Lead User* was introduced by Hippel [Hipp86]. These users expect a particularly high benefit from a change, and therefore appear as innovators who take their destiny into their own hands. However, they still act within a community, and their goal remains to push their community forward, as it is the inherent interest of the whole community to learn from each other and advance thereby. Of particular interest are domain experts, who are experts in their specific domain, not necessarily in computer science; however, they are using computer environments to perform their daily tasks [CFF*03]. End user development together with infrastructuring means gives these users more agency. In design practice, the reverse of tailoring products to large masses, i.e. *scaling up*, was recently described by Myerson as *scaling down*, i.e. the explicit inspection of particular use cases to learn about innovative edge cases that may benefit the whole later on [Myer16]. The author highlights certain principles, like the need to design for specific people instead of artificial personas that are constructed with the goal to correspond to as many users as possible. Lastly, Hannemann discusses the importance of end user development in the context of open source projects [Hann14b].

In the approach presented in this thesis, we integrate end user development with infrastructuring by provisioning an infrastructure that adapts to the needs of a community including end users who are using it. We link these levels in our overarching methodology that we present in the remainder of this chapter and that is derived from the ATLAS methodology discussed in the next section. In Chapter 3 we present social requirements engineering as the first step of our DevOpsUse methodology for a community-oriented means of requirements engineering. There are various techniques for requirements prioritization in a social community context, like user rankings, social network analysis and user statistics.

2.3 Community-Driven Information Systems Development Methodology

Each information system used by professional communities needs a strong methodology that guides its formation, development and on-going sustainability. We start this section by explaining the ATLAS theory, developed at our chair. We then look into state-of-the-art software development methods from industrial practice, before we combine both ATLAS and DevOps in the DevOpsUse life cycle model.

2.3.1 Large-Scale Community Support with ATLAS

The Community of Practice theory by Etienne Wenger is a social learning theory that explains learning processes within professional communities [Weng98]. CoP are groups of people who share a concern or a passion for something they do and who interact regularly to learn how to do it better. At its core are communities, that advance themselves through their shared repertoire, mutual engagement and joint enterprise. It is also the underlying description of the community term in the ATLAS methodology pictured in Figure 2.1. The community evolution at its top occurs through accessing community needs. Hereby, an essential role is played by social requirements engineering as a direct means of gathering ideas and needs. The self-managed principle within ATLAS becomes evident through the self-monitoring tools which empowers communities to analyze their evolution themselves. The MobSOS framework developed by Renzel [Renz16] contains various tools for self-monitoring, such as interface access logging as quantitative, and surveys as qualitative tool. Community-specific information systems development also happens within the community. As a consequence, both researchers and developers are regarded as belonging to the community.

Community-intrinsic repositories and middleware support the CoP evolution and are, as support tools, the main target of our research. With the advent of social software, the development process has become much more complex and engineering methods have to consider informal community structures much more than before. Understanding and exploiting the differences between organizational information systems as planned and community information systems as in use contribute to better utilization of organizational and societal resources.

Empowering communities with self-monitoring tools is rather straightforward, in particular, as tools such as Excel have reached widespread adoption in business domains. Even data science and business intelligence applications such as Tableau and Power BI are now widely used. The introduction can be driven by domain experts who as lead users guide the adoption of a tool within their community. The introduction of means to enable communities to participate in their own information systems development is much more cumbersome, on the other hand. That collaboration within development communities is possible even across remote sites, is shown by numerous real-world practices of open source development. How automation plays a key role in this is shown in the next section using the example of DevOps. We then expand the DevOps methodology to include end users and introduce DevOpsUse, presenting a holistic view of the community, by integrating DevOps and ATLAS.

Our research is driven by the social learning theory of Communities of Practice (CoP) [Weng98]. In a CoP, community members collaborate to mutually improve their practices. Our communities

consist of end users who work in a digitized environment, developers who create software, and researchers analyzing the community-specific interactions. The roles are overlapping, as we enable end users to participate in the development process. However, integrating a large amount of real end users into development requires new approaches for reaching out to them. While traditional methods such as co-design sessions are driven face-to-face with co-located researchers and end users, in our scenarios people may live scattered around the world in different time zones. These make online tools for enabling co-design necessary. However, most of the tools available are limited by the number of users that can use it. Often, if a tool allows for many people to participate, the number of tasks that can be performed using the tool is limited. To coordinate end users and to guide the collaboration, awareness of each others' actions is necessary. Therey, designers seldom recognize the influence of side communication channels [Parm17]. These channels, like e-mails and forums, can be used to discuss designs, and to share design iterations. However, they become difficult to manage over time leading to a restrictive number of collaborators and resulting in sparse and superficial feedback [LTW*15].

Renzel compares the ATLAS and the design science research methodologies [Renz16, RKJa15]. According to Renzel, ATLAS misses a specific focus on information system evaluation, including "(...) success modeling, measurement, analysis, and validation" [Renz16]. The MobSOS framework thus provides methodological and tool support for evaluating community information systems within the ATLAS framework. We further extend ATLAS by means of scaling community information systems development and research efforts from one to many communities. Thus DevOpsUse encompasses multiple communities of practice, developers and the underlying IT infrastructure.

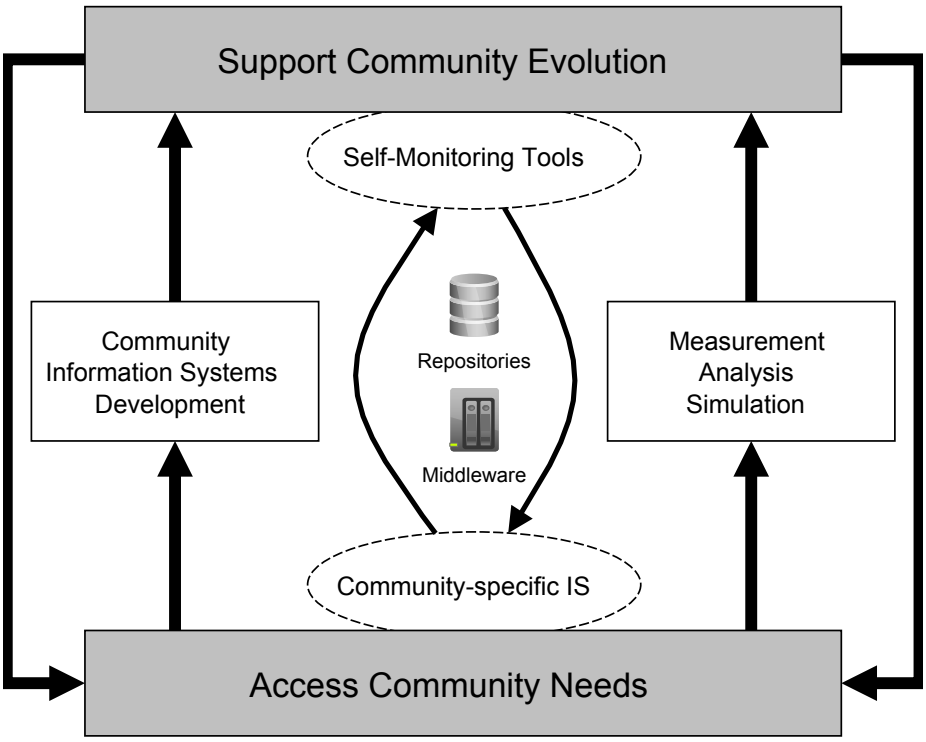


Figure 2.1: The ATLAS Methodology [Klam10c]

Before we present the final conceptual view of DevOpsUse, we first look at current agile development practices in the next section.

2.3.2 DevOps

Agile software engineering methodologies are driven by self-organizing and cross-functional teams and an incremental approach. There are various manifestations of agile principles, like *Extreme Programming*, *Scrum* and *Kanban*. Extreme Programming was one of the first agile software methodologies practices in software development [Beck03]. Among its characteristics, there is pair programming, where two developers sit side-by-side in front of the source code, and one of them looks over the other's shoulder while developing. Scrum is another way of developing in an agile manner. Besides being very similar to Extreme Programming, it is stricter in certain aspects. For instance, it requires specific iteration lengths and more logging. It uses its own vocabulary that is well-known among its applicants. The main feature in Scrum is that project leaders become part of the development team. They turn into team members, and their role is that of a moderator rather than a manager. It helps them to get to the bottom of current problems sooner and better. However, concerning the development tasks, in Scrum there are no fixed roles. Any team member should be capable of taking over work on the tasks regarding various parts of the software life cycle. If team members see themselves as specialists in certain parts, like architecture or deployment, it can be hard for them to accept their role. Kanban is very similar to Scrum, however, it again loosens up some of the time restrictions on sprints. Cross-functional teams, on the opposite, are optional. On the contrary, Kanban's principle of "leadership on all levels" requires that as many employees as possible show leadership at all levels and make suggestions for improvement. Additionally, the division of tasks can be structured in a way that gives experts tasks related to their core competences. In practice, one can see various hybrid forms of agile development practices, often mixing Scrum with Kanban. This latter combination that takes on the features of both methods is often called *Scrumban* [Lada08]. Central artifact in both Scrum and Kanban are project task boards that collect the requirements of a project, broken down into epics and issues. Epics describe larger stories that typically include multiple functionalities that build upon each other. Issues, on the other hand, are smaller-level units that can be worked upon in about eight hours at most, as recommended by Scrum.

Agile software development methodologies have been much-discussed in the last years. Especially in the area of innovative Web businesses, faster release cycles and stronger orientation towards customer needs due to the agile practices, and thus faster deployment of produced software has received increased focus [HuFa11]. It became evident that the developers' goal of releasing software at an increased pace is orthogonal to the operators' aim to provide stable and long-running systems. Traditionally, companies separate the two departments "development" and "operations". The former produces code, while the latter manages the often complex information technology landscape. Scarce communication between these departments and the strict separation of concerns have led to conflicts due to unclear responsibilities and mutual accusation on system failures. The methodology called DevOps (a clipped compound of "development" and "operations") therefore tries to mediate between these departments by promoting cooperation through establishing a new culture [EGHS16]. This is achieved by a tighter integration of software development and deployment; one of the ways how this is achieved is by increased automation. The term DevOps not only

comprises the methodology, but a mindset of working towards the same goals, and also a collection of software tools that support the collaborative and automated culture.

Automation as central core of DevOps is employed at several layers from development over delivery to deployment of software. At the development stage, refactoring stands for changes of the code with the goal to preserve functionality while improving code readability, structure, and reusability. Today, many integrated development environments (IDEs) automate refactoring operations like renaming of variables and externalizing code blocks into own methods or modules. Organization-wide agreed code style guidelines can be checked automatically before uploading code from a development workstation to a shared source code repository. In addition to local tests, software can also be automatically tested for bugs on integration servers. From there, it can be automatically released on download servers, or pushed to app stores for seamless distribution to devices of end users. Initially on mobile operating systems, now also Desktop operating systems like Windows allow automatic updates of installed software managed by the operating system.

There are several threats to DevOps that are opposed to its propagated working culture. For instance, bureaucracy and strict hierarchies do not cope well with cross-departmental cooperation across teams. Even a tendency towards perfectionism is not conducive to a successful implementation of DevOps. Often, the demanded principle of "high quality" is incompatible with the "fail fast" principle of DevOps. However, our community-based paradigm approaches development from the end user perspective, i.e. there is an inherent desire to improve within the community under consideration. How this may work can be seen in the global collaboration of numerous large open source projects. In openness, whether through open communication or open escalation chains, we see a solution to most of the challenges ahead. Even an open analysis platform, as we offer it with SWEVA in chapter 6, can lead to the avoidance of the issues.

Due to the widespread adoption of the methodology behind DevOps, recently, further terms have been coined to denote the collaboration of interdisciplinary teams. For instance, *DevSecOps* refers to the introduction of automated security and IT compliance measures on top of a DevOps infrastructure. In line with this, in the next section we present *DevOpsUse* as the integration of community practices in the development process.

2.4 DevOpsUse

The DevOps methodology as described in the last section is driven by agile development principles. Agile practices in turn promote an increased attention to end users and their specific needs. This is reified by shorter development cycles and an accompanying frequent comparison of the actual and desired state. However, although the end user needs are well taken care of, end user communities themselves are not well represented in the model. The DevOps life cycle presented in the core of Figure 2.2 shows the collaboration of developers and operators. When read clockwise starting from the top left, the cycle starts with *feedback* that is possibly collected from end users; it continues with the development and test phases and is then handed over to operators who deploy the software and then monitor its usage. With the monitoring we are now back at the user. We argue, that there are many more possible points of contact with users. Apart from the aforementioned requirements gathering phase through collecting ideas and needs as feedback, end users are also considered during development and testing in co-design practices and through beta testing. The context of end

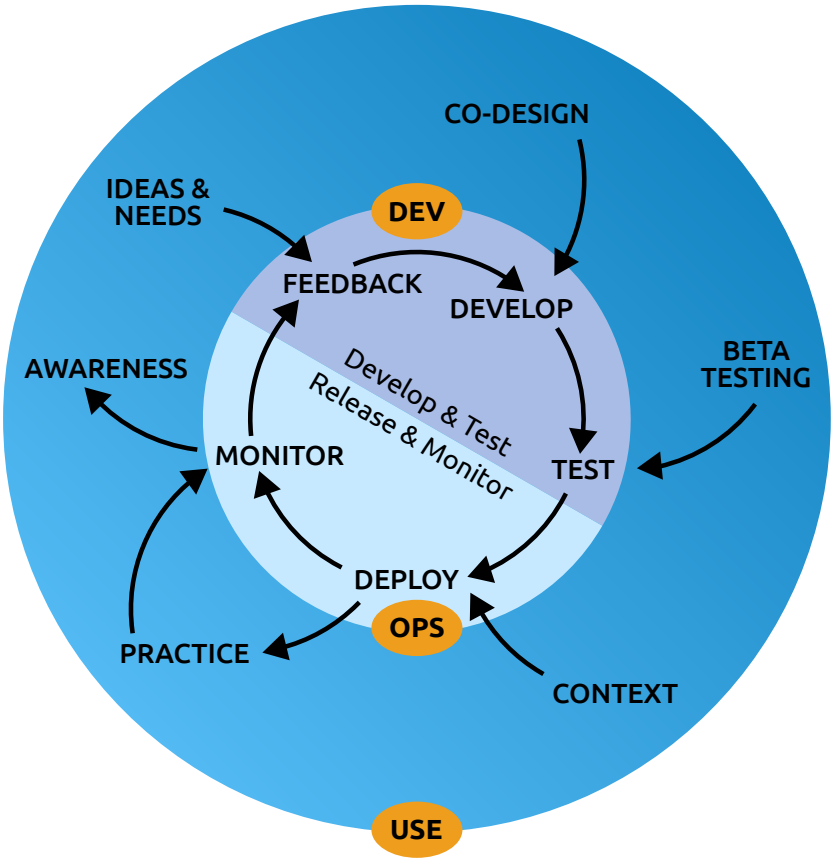


Figure 2.2: The DevOpsUse Methodology

users plays a major role, as it often dictates legislative policies among others, e.g. concerning data privacy. Practices may influence where and how software is executed. Last but not least, monitoring is also effective as self-awareness tool. The main drawback of DevOps as we see it, is therefore the lack of end user involvement as well as the missing awareness for community practices. With the community-driven ATLAS theory in mind, we thus extend the DevOps life cycle with end user communities by appending these practices into the life cycle. This extension is called DevOpsUse. While ATLAS only focuses on one community at a time, DevOpsUse transfers the life cycle to other communities, generalizing it to a certain extent. These specific end user community practices are shown in the outer ring of Figure 2.2.

How to support and automate the collaboration between end users, developers and operators is thus one of the main questions. In particular, it is of interest, how information systems should be designed that support these processes in a continuous innovation life cycle. As both hardware and software platforms on the Web continue to improve, we see many contextual forces and changes affecting community information systems. Thus, any framework supporting communities needs to cope with the changes and its challenges. In the following, we describe the general structure of the end user aspects of DevOpsUse as our proposed societal software development methodology. We start on the top left of Figure 2.2.

Our communities consist of end users working in digitized environments, developers creating software, operators instantiating applications, and researchers analyzing the community-specific interactions. The roles are overlapping, as we enable end users to participate in the development process. First, *social requirements engineering* [ReKl14b] opens up the requirements gathering process from low-scaling focus groups to the crowd of the Web 2.0. A related area of research is therefore called CrowdRE [GSA*17, GDAd15, SOC*17]. We want to enable a platform, where new ideas, feature requests or bug reports can be entered, liked, shared and commented. This enables developers to get in touch with end users at a very early stage, to enable a feedback loop. From here, requirements can be exported to more developer-related issue trackers. At any time, the current state of the implementation of the requirements is communicated clearly, leading to a transparent workflow. If enabled, contributors get automatic notifications by emails upon any change of the state of a chosen requirement. Developers then take over requirements and create an initial prototype. In a co-design sessions, end users participate in the further shape of the software. The next step after requirements are turned into code is putting the developed software into practice. Here, both integration and beta tests contribute to the successful instantiation. While for this part, we do not provide a dedicated tool support in the course of this thesis, we adhered to beta tests on a meta level. For instance, the continuous delivery process of Requirements Bazaar features a beta testing phase[1] and a fully automated roll-out of new versions.

After the software is put into practice, we are interested in the runtime behavior of the system including its usage within the community. For this reason, we perform formative evaluation during the use of the deployed information system. At this point, the diverse nature of today's systems becomes evident. For instance, learning contexts in Industry 4.0 settings are particularly challenging because of the various interdisciplinary data sources. Interacting with industrial and wearable devices naturally produces a lot of data. Some of the sensor data is of environmental origin, like temperature, humidity or brightness. Other data originates from the operating process, such as pushing a button, turning a rotary switch or moving a slide switch. Yet other input is coming from the human machine operator itself and is captured by wearable devices. Such data include the heart rate, body temperature or general body motion sequences. Finally, there are context-dependent data like the time of operation, the age of the machine operator or the years of work experience. It is easy to follow that the amount of data becomes more complex with the number of inputs. Moreover, it makes it hard to grasp interrelations, such as connecting an increased heart rate with a particular machine operation. To this end, we need an overarching analytics strategy that is serving both software usage as well as learning analytics. In communities of practice, the analytics can serve the goal to strengthen the learning goals, and to make the information system more successful at various aspects like failure behavior and usability. Visual analytics is set to combine the power of computer-generated analytics and human interpretation (cf. Chapter 6). Computers are capable of evaluating enormous amounts of data in a very short time. The human brain, conversely, is optimized to quickly identify correlations. Visual analytics is an approach to combine the best of both worlds [KAF*08]. It involves human judgment in the analytics process, utilizing characteristics such as flexibility, creativity and background knowledge. It is possible to directly interact with the represented data, gain new knowledge and therefore make better decisions in the future. The goal of visual analytics is to gain knowledge through model building, visualize it dynamically and then feed back the acquired evidence into the data gathering and filtering process. It aims at

[1] The beta environment of Requirements Bazaar is available on https://beta.requirements-bazaar.org.

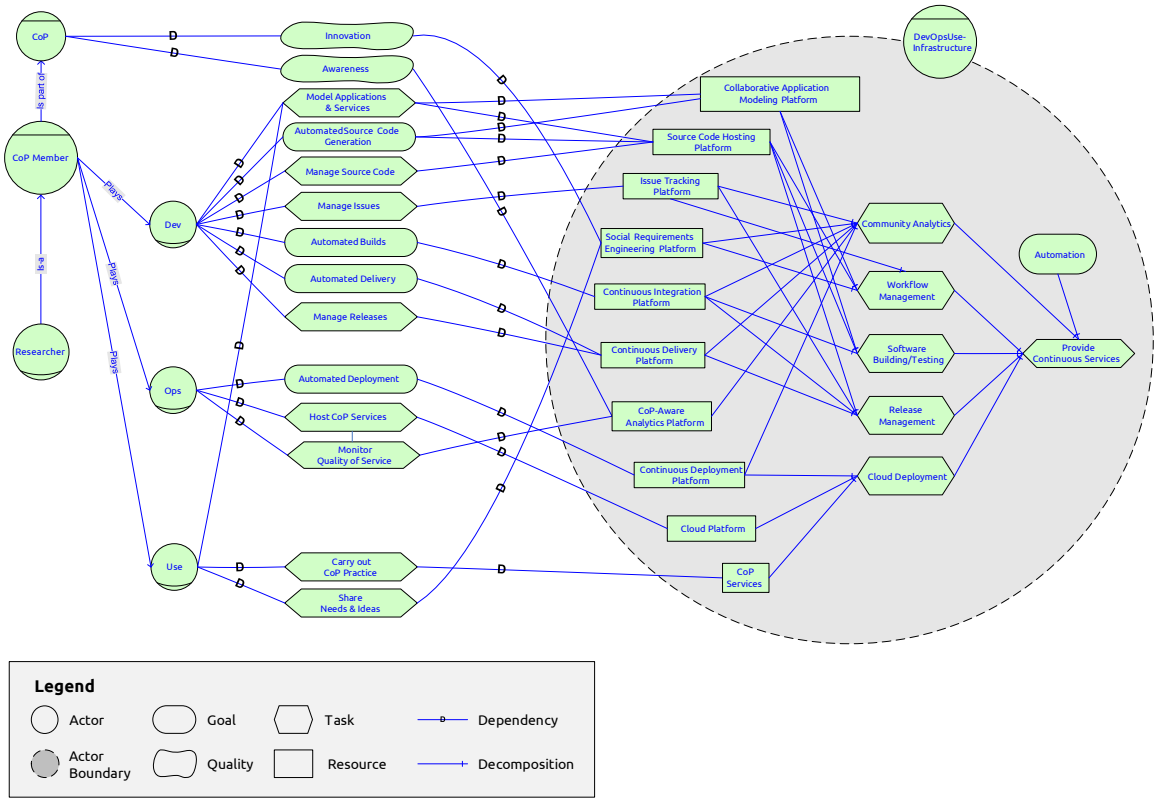


Figure 2.3: Conceptual Model of DevOpsUse in iStar 2.0 Notation

integrating human capabilities into the data analysis process by using visual representations and interaction techniques [KKEM10]. It is a multidisciplinary field with many focus areas [ThCo06]. Examples include weather forecasts, disaster and emergency management, software analytics and physical simulation in engineering. In the DevOpsUse model, users get involved in the analysis process. Data interpretation and reasoning are supported by visualizing the important aspects of the data. When this process is distributed and enlarged to a bigger group of people, additional social processes become active. Thus, visual analytics is even more important if many stakeholders view possibly different views and interests on the data are involved. Therefore, the model needs to be extended by a social dimension. It leads to multi-faceted visual analytics with possibly conflicting debates about interpretation of results and far leading analytic activities. The context of a community using an information system gives a common social structure for the stakeholder in case of missing institutional context. Hereby, information system use and development are inseparably interlocked with each other by balancing agency and structure [OrRo91].

In the area of *technology-enhanced learning* (TEL), which we will depict in more detail in Section 2.5, the particular application of self-monitoring is *community learning analytics* (CLA) which includes the processes of identification, analysis, visualization and support of informal community-regulated learning processes [Klam13]. As professional communities of practice are learning informally, the significance of self-monitoring becomes apparent.

Figure 2.3 breaks down the roles of a DevOpsUse community of practice and depicts their de-

pendendies, relations and tasks in an i* diagram. The i* modeling language is high-level and agent-oriented, with a focus on modeling strategic dependencies between arbitrary, not necessarily human actors [Yu97]. We follow its current notation and spelling suggestion described in the iStar 2.0 language guide by Dapliaz et al. [DFHo16]. We have chosen iStar 2.0 as metamodel, as its origins are in Requirements Engineering [DFHo16] and the diagram can thus be seen as input for the whole societal information systems development process entailed by DevOpsUse. The main conceptual elements of iStar 2.0 are actors and relations between them. An actor is considered an active entity that carries out actions to achieve goals. In iStar 2.0, there are two types of actors. *Roles* describe an abstract characterization within some specialized context or domain. *Agents* are concrete instantations of actors. Actor links express relationships between actors. To accomplish a goal, an actor might depend on another actor; in iStar 2.0 such a dependency is modeled as two actors connected by an edge, that connects a depender and the dependee. The edge is annotated by a goal, a task or a resource as dependum. Internal rationales can be modeled by an actor boundary that contains the internal rationals to external dependencies. An iStar 2.0 model can be visualized via multiple perspectives or views. These are *Strategic Rationale* (SR) with all the details including actor boundaries; *Strategic Dependency* (SD) showing each actor and their dependencies; and a *Hybrid SD/SR* view that combines the aforementioned two. The DevOpsUse conceptual model in Figure 2.3 displays such a hybrid view.

Our model visualizes two main actors on the top: the community of practice and the DevOpsUse infrastructure supporting it. Any community of practice member plays one or more of the roles developer (Dev), operations (Ops), and end user (Use), each of them supporting the CoP in its pursuit of getting better. Each role is focused on reaching a set of goals by executing tasks with the help of resources provided by the infrastructure. Note, that the researcher as a special role is a community of practice member itself, thus he or she may play the sames aforementioned roles. Developers depend on the infrastructure for managing source code, issues and releases with the goal of automation. Therefore, the infrastructure provides a source code hosting platform, an issue tracker, and platforms for automated continuous integration and delivery. Operations depend on the infrastructure for hosting services. Their goal is to automatically instantiate builds previously produced by the build resource and to further support them. They need to constantly measure quality of service with the help of analytics services. End users depend on various services provided by the infrastructure to carry out their particular practices. DevOpsUse considers end users as innovative co-designers of the infrastructure and its services, thus actively contributing to the CoP's overall goal of continuous innovation.

Innovation as quality characteristic heavily depends on feedback from multiple sources and in multiple forms. The social requirements engineering platform as resource enables DevOps and end users to share, discuss, negotiate and prioritize ideas for new services or improvements of current ones. The internal rationale of the infrastructure is to support the CoP in its development process by providing a coherent and integrated set of resources. Community analytics contribute to the overall CoP awareness goals which takes different role-specific forms in DevOpsUse. Communities focus on CoP-specific metrics for community success [Renz16], while DevOps rather focus on metrics provided by the development-targeted infrastructure, as well as to CoP-specific metrics related to development [Hann14b]. A particular example here is the *Social Web Environment for Visual Analytics* (SWEVA) tool that we present in Section 6.4 for community self-awareness; its technological foundation originates from an earlier Internet of Things protocol monitoring system that we were

able to repurpose.

While a far-reaching automation is the ultimate goal, a fully automated life cycle of community-specific development is currently not achievable. Instead, we aim for a culture that supports and accepts automation, where humans are not only required on possible failures, but are also seen as societal links between the subcommunities of end users, developers and operators. For instance, the automatic builds within a continuous integration server make bugs apparent, however the bots are not able to resolve the problems automatically.

Above, we have followed the DevOpsUse life cycle around and presented exemplary tools for supporting end users in the active development of their community information systems. In the following, we look at the scaling aspect that distinguishes the DevOpsUse from the ATLAS methodology. As mentioned before, Renzel [Renz16] discusses the design science research for information systems process by Hevner et al. [HMPR04] that emphasizes a rigorous cycle from problem identification to evaluation and communication. In our research, we followed this cycle for each of the aspects above, by developing artifacts and evaluating them. Moreover, we identified mutual interdependencies as crosscutting concerns between artifacts. For instance, Requirements Bazaar as social requirements engineering tool features a rich API that allowed us to integrate its functionality into the WordPress content management system (cf. Section 3.2.2) as well as into a mixed reality system (cf. Section **??**). In community-oriented design science research, no predefined success criteria can exist [RKJa15]. Thus, with scaling the number of users, also the research method must keep track. An example is crowdsourcing co-design, that we describe in Section 3.3.

DevOpsUse equally stresses the importance of developers. We argue, that the open structures of the Web, and the collaboration it enables, help it sustain itself. All developed artifacts both feature interfaces and are available as open source solutions. The inclusion of the open source community enables and pushes innovation further by allowing the reuse through the permissive licenses. In this respect, by explicitly opening up our artifacts, we follow paths like open innovation, CrowdRE and citizen science. Here the societal dimension becomes apparent. An area where the societal impact is particularly evident is life-long learning. Over the past few years, we have identified this need in various project contexts in the context of technology-enhanced learning. In many places, the tools deemed essential were missing and hat to be developed from scratch. This also shows empirically that research is crucial at this point. The concrete projects are presented in the next section. An overall validation of our research methodology via a long-term study of these projects is discussed in Chapter 7.

2.5 Technology-Enhanced Learning as Testbed

The research performed and described in this dissertation emerged from and was applied in the context of several European funded large-scale and local research projects in the area of technology-enhanced learning (TEL). As learning inherently deals with societal context changes, for instance in the ongoing digital transformation of workplaces, we consider it a highly rewarding area for the type of information systems engineering research we perform. Learning can be supported by technology in various ways. The international Organisation for Economic Co-operation and Development (OECD) distinguishes between three forms of learning [Werq10] *Formal learning*

deals with all education normally carried out by teachers or professors in school or university contexts. *Informal learning* is just-in-time, social, multi-episodic and problem-oriented and about situations [KLHo07]. It is best characterized as learning, that happens as part of other activities. *Non-formal learning* deals with everything else, including sports instructions and programs such as scouting.

A particular challenge became visible for professional Communities of Practice over the last years: the ongoing digital transformation of workplaces. While it enables new kinds of businesses and services, there are many threats to existing workplaces. In this sense, digitization is often perceived as a threat to industrial workplaces, saying that on the long run, technologies like robots will replace human workers. Undeniably, the constant stream of innovation puts pressure on workers to acquire new skills to deal with new tasks, new workflows and new machines. Life-long learning is nowadays considered as a possible solution to help workers to update their knowledge about specific work processes. It can be categorized as informal learning, as by the definition above. Continuous training with information system support plays a special role. Constant evaluation and self-assessment of the training helps to render it more effective and useful. Therefore, it is advantageous that in these high-tech Industry 4.0 settings, both industrial devices and wearable devices of workers collect huge amounts of information. Combined, the gathered data is a valuable asset for further analysis in order to provide insights into human-machine interactions; additionally it may be used for replaying training data to future workforces. A challenge of these workplace settings is that the educational context is not necessarily tied to a particular location and time, but instead may happen at different places over various periods of times.

Figure 2.4: Screenshot From the “Ach so!” Video Annotation App [BPLe14]

Another exemplary use case of software support in informal learning are video annotations. In the context of workplace-oriented informal learning our partners at Aalto University in Finland developed "Ach so!" in collaboration with us. Figure 2.4 shows a screenshot of the application. While the app runs on the Android mobile operating system, it sits on top of our infrastructure that we describe in detail in Section 5.2. Ach so! was developed for use by construction workers. An idea collection revealed the following requirements beforehand [NRK*14]:

- Minimal influence on workflow: The application should not interfere with any of the actual labor that needs to be performed.
- Clear about purpose: The intended goal of the application must be obvious.
- No fine or detailed touch interaction: As construction workers are often required to wear gloves, it is not possible to design apps with a lot of touch interaction.
- Benefit should not depend on video quality: The primary focus of the workers needs to be their workflow, thus any quality issues like missing stabilization of the video feed should not deteriorate the purpose.
- Annotation should be performable later: Enriching the video with contextual and training information should be doable at a more suitable time.

This collection exemplarily shows the special nature of requirements for an information system for informal learning at the workplace. Beyond the app design, we also need specialized evaluation strategies for informal learning as opposed to the researcher-centric, static evaluation methods of formal learning in a classroom. For instance, input and output devices like body-worn sensors and augmented reality head-up displays enable new immersive analytics capabilities. Also, we see new possibilities for formative and summative evaluation of learning systems. Altogether with information system analytics, we see a new quality of learning analytics.

In the following, we characterize the two central research projects that this thesis contributed to. These are *Learning Layers* and *Wearable Experience for Knowledge Intensive Training* (WEKIT).

Learning Layers: Learning Layers was an EU Framework Program 7 large-scale integrating project running from 2012 to 2016, involving 18 partners with a budget of 13 million Euros. Core work packages included R&D activities towards innovative software for scaling informal workplace learning and their evaluation. The objective of the project was to develop a set of modular and flexible technological layers for supporting workplace practices in companies that unlock peer production and scaffold networked learning. This objective was addressed with various mobile apps and social software services on top of a scalable, light-weight infrastructure that allows for swift federated deployment in highly distributed and dynamic settings. Since the start of the project, end users from vocational education and training backgrounds in healthcare and construction were deeply involved in system co-design, and continue so even beyond project lifetime. This project adopted and extended several instruments previously used in the *Responsive Open Learning Environments* (ROLE) project at our chair. These included the *developer task force*, a decision making body, an open source software strategy, and a tool-supported social requirements engineering process. The software results are still in use, commercially and scientifically. In particular, the established single sign-on solution (cf. Section 5.2.2) continually attracts new users.

WEKIT: The WEKIT project was a consortium of twelve partners from six countries and ran under the Horizon 2020 funding of the European Commission from 2015 to 2019. The objectives were to develop an open technology platform for augmented reality experiences, to augment training in situ with live expert guidance and to create a roadmap for augmented reality learning together in a community of stakeholders. The application cases were in aeronautics, medical engineering and space. The key output is an augmented reality software platform that allows to capture experts in real-life situations with a newly developed vest and various body-worn sensors [LVJ*19]. Trainees can then replay the previously recorded actions of the expert in their augmented reality head-up display, where a three dimensional mannequin is instantiated to perform the actions. During the project lifetime, a startup was founded out of the project context that is now serving customers with the soft- and hardware developed within the project, while constantly extending it.

Figure 2.5: WEKIT Application Contexts: Space, Aeronautics, and Medical

2.6 The Web as Testbed for Technological Evolution

This section gives an overview of the technological context in which we operate. We close this chapter with a review on the technological context we are moving in. As highlighted earlier, no technology has ever impacted society the way the Web has over the last decades. It is still under constant flux, driven by various forces like advancements in hardware, changing requirements, and last but not least by legislative influences. The standardization efforts by affiliated companies, research communities and individuals are explicitly designed to adapt and extend the technology stack underneath it. In this regard, over the last years, multiple technological generations have been passed through. It is therefore of paramount importance, that any community-driven software development methodology deals with these fast-paced changes and incorporates it into its underlying life cycle model, in the same way and speed the Web embraces new technological developments. Thus, DevOpsUse can and will be measured on how well it copes with the inherent changes. When designing our methodology, we implemented prototypes for the four areas of innovation, development, deployment as well as analytics that are reflected in the chapter structure of this dissertation (cf. Figure 1.2). We took particular care of making these tools available across platform and hardware silos. In an approach that is cross-cutting these four areas, we validated DevOpsUse with three development stages of the Web that happened over the last years. These are near real-time peer-to-peer computing, edge computing with microservices, and the Internet of Things. At the end of this thesis, in Section 7.1 we will revisit these stages and evaluate the applicability of DevOpsUse to them.

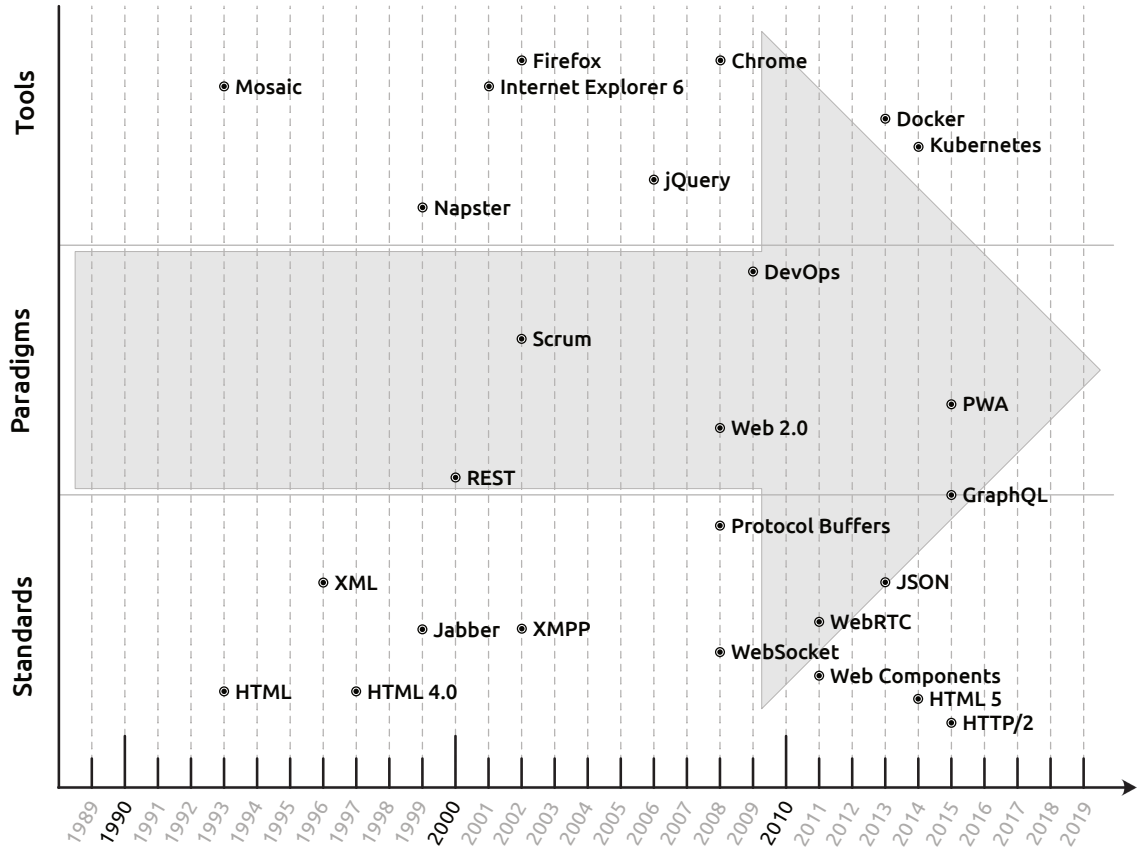


Figure 2.6: Timeline of Standards, Paradigms and Tools as Forces on the Web

Figure 2.6 presents a timeline of contextual forces on the Web, grouped into standards, paradigms and tools. It deliberately starts with 1989 as the birth year of the World Wide Web. We end with the year of submitting this thesis, which is thirty years later. in the first decade in the 1990s, the technological ground-breaking introduction of the HTML and XML standards established the proliferation of the Web. The former, HTML, as a markup language to structure textual information into a visual format, and the latter, XML, as a way to structure data. Later, XML became the foundation for service-oriented architectures via the SOAP protocol. In 2000, Fielding first described the REST protocol in his dissertation [Fiel00], but it took another decade before it became the underlying principle of today's microservice architectures. Around the same year, we see the introduction of Jabber, which was later open sourced as the *Extensible Messaging and Presence Protocol* (XMPP). This went hand in hand with the first implementations of real-time systems on the Web, with the first application of the protocol being instant messaging.

In the year 2006, the JavaScript library jQuery was introduced. Though the influence of a single library seems minor compared to milestones like the introduction of HTML and REST, we are convinced that it helped elevating the Web from a Hypertext-oriented information resource to an interactive platform for highly sophisticated Web applications. Another large contributor was the Ajax concept of asynchronous message flow within the Web browser. Originally introduced as XMLHttpRequest object in the Microsoft Internet Explorer, it first allowed to retrieve content from

a server independently of the synchronous HTTP message exchange that downloaded the initial HTML, CSS and JavaScript resources. Another game changer was the Web 2.0, first perceived through Google Maps and mashups built with it. Particularly groundbreaking here was the innovative design pattern of the map tiles, in which the world map is divided into square objects, whose disappearance from the screen by dragging causes a repositioning in the coordinate space to the other side. This improves the performance of the instantiation, initialization and subsequent destruction of their data structure. It lastingly changed the way large data structures are displayed on resource-constrained end devices. For instance, the common list user interface toolbox on Android is still based on this pattern.

The upcoming of Web 2.0 user interaction patterns then fostered the creation of multiple new standards in the realm of near real-time communication, that is still ongoing. These are mainly *WebSocket* for asynchronous communication with a server, and *WebRTC* with its primary use case being peer-to-peer video transmission from browser to browser. While video conferencing within the browser is still in its infancy, near real-time shared editing applications are now mainstream in the area of office software, through Google Docs and Office 365, though its adoption in other areas is still falling behind. The underlying Google Wave protocol was in a leading position here, even if it was not able to prevail in the end. The 2013 standardized JSON is now the de-facto standard for exchanging data on the Web, and has replaced XML in many applications. It is also the data structure the GraphQL query language has been built upon. While with REST, a whole resource was transferred upon a request, as a query language on the Web, GraphQL allows for a more flexible specification of what the client specifically needs.

Overall, we can perceive an acceleration of innovations in recent years. This is due to accelerated computing operations according to Moore's law, as well as new portable device form factors made possible by the miniaturization of electronics and wireless transmission protocols. On the other hand, a clear social transformation is taking place, through new working models that lead to workforces being constantly on the move, both literally and also figuratively through new technologies that have to be learned. As a result, the Web finds itself in a constant state of tension between these forces. One of the greatest challenges of today is the integration of the Internet of Things. The term stands for the interconnection of numerous everyday devices, allowing new use cases.

In the following we close with an exemplary description of a software architecture for informal learning applications based on Web technologies. We use it to demonstrate the universal applicability of the Web as testbed, or "blackboard" [PaZi17] of architectural styles. In this testbed, it is explicitly possible to emulate various architectures within the given infrastructure. In the example information system, services and frontends are deployed and provided by backends on a server. Services process input and serve their output to be retrieved by clients. This is done in a request/reply manner; for every request to the service, a reply is generated and served. They may be executed on workstations, dedicated servers or fully virtualized servers in a cloud environment. Frontends are made available as HTML markup files, JavaScript files containing execution code, and CSS stylesheets including style definitions. The software running on low-powered sensor nodes is much more comprised. For instance, a temperature sensor often only consists of the physical hardware measuring physical values, and a tiny piece of software that transmits the measured sensor data to a server, either directly or via a gateway. In contrast to the aforementioned request/reply pattern, such 'dumb' sensor hardware follows a publish/subscribe pattern. In a publish/subscribe system, events are sent to a central entity, which then notifies all subscribed actors in the system about the

event. To stay in the terms of our architecture, services deployed on the backend would typically be clients of such a publish/subscribe system, turning them into hybrid services. On the one hand they are available for request/reply style interactions, on the other hand they are notified of events happening in their network. A client willing to show the historical data in its frontend can then easily request the historical records from the service.

In this section and in particular in the example above we have seen the universal applicability of the Web environment. However, the question is, how to stabilize all the contextual forces and technological possibilities. It is also a key concern here, whether in the long term the standardization processes will be sufficient to stabilize information systems built on top of it, or whether the loose coupling will lead to a steady decline. On the one hand, open interfaces allow the composition of apps within the open Web. On the other hand, changes at the interfaces may cause these compositions to drift apart. One of the threats to the Web as an information infrastructure is therefore constant change. A possible answer to the continuous dichotomies described in Figure 1.1, and the interplay of APIs vs. standards, open vs. closed etc. is therefore exchangeability of elements. How we managed these on different layers of our community infrastructures is therefore discussed in the next chapters. For instance, the deliberate substitution of the protocol in the Direwolf framework from federated XMPP servers to distributed WebRTC peer-to-peer connections is discussed in Section 5.4. In this particular case, by exchanging the communication protocol and the gained latency reduction, we enabled a broad variety of new use cases like gaming, without touching the user interface layer. Similarly, replacing REST-based interfaces with GraphQL as described in Section 4.3.2 enables sustainability by being able to define the queried object properties from the requesting client.

2.7 Managing Continuous Evolution in Production

An industry that is particularly affected on many levels by changing technology landscapes is production. It is subject to constant disruptive changes due to breakthroughs in the areas of the Internet of Things, network technologies (cf. the introduction of low-latency 5G technology), and computational power (cf. quantum computing). Thus only when the possibilities of these radical digital shifts have been fully understood, controlled and stabilized, do they open up new opportunities. In production engineering, agile approaches like Scrum [SGBS16] and minimum viable products (MVP) [SDSc18] are being discussed. Beyond, following our basic rationale described above in the context of the Web, we see how the design, development and operation of a stabilized infrastructure can unleash its full potential together with its users. In this section we describe the parallels and possible application areas of the methodology and tools developed in this thesis.

The current manufacturing landscape is characterized by numerous domain-silos that comprise sophisticated and highly specialized models and data [RWTH18]. Figure 2.7 presents a typical situation in producing companies today: Data and models are only available within proprietary systems and not ready for cross-domain use. On the top left, many proprietary systems characterize the development environment. While in information systems development there are numerous programming languages, integrated development environments and runtime frameworks, the production (planning) landscape is characterized by an even more diverse set of design and planning software,

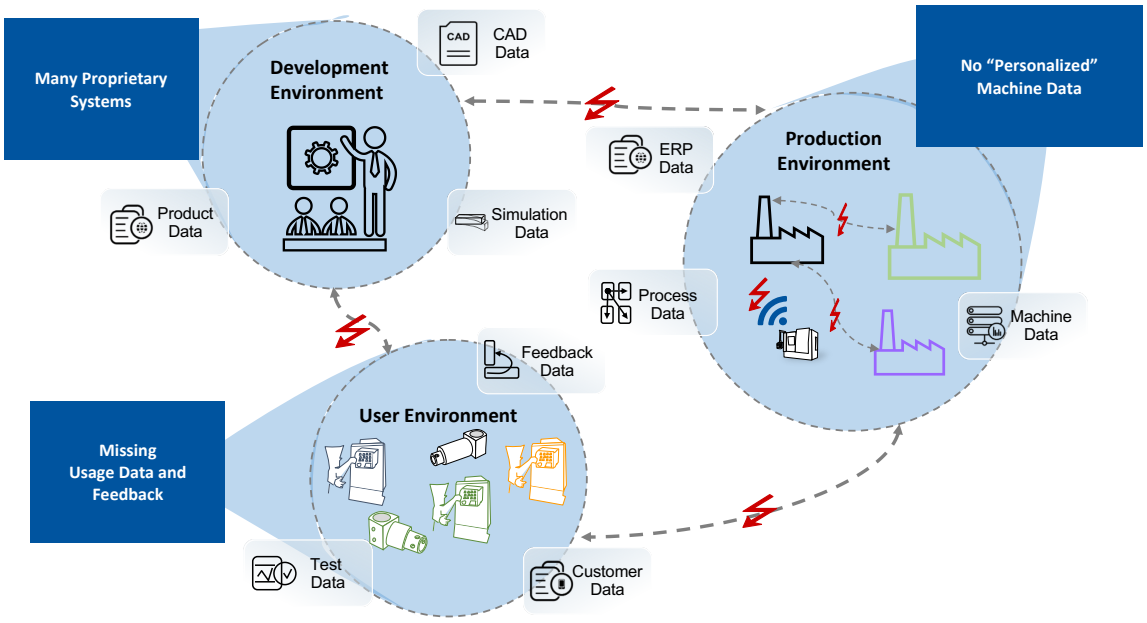


Figure 2.7: Data and Models Within Proprietary Systems in Producing Companies [RWTH18]

file formats, product specifications and runtime specifications. This leads to disruptive incompatibilities on the transition to the production environment, depicted on the top right. On the transition to the user environment, shown on the bottom left, further incompatibilities occur. An enormous amount of raw data is stored in various databases following these cycles, like enterprise resource planning (ERP), manufacturing execution systems (MES) and customer relationship management (CRM), to name a few. The heterogeneity thereby restrains accessibility of data and knowledge across domains.

The underlying model in Figure 2.7 resembles the DevOpsUse life cycle model depicted in Figure 2.2 in multiple ways. While the development and user environments match the development and end user stakeholders of our community of practice, the production environment can be considered as equivalent to the operator stakeholder. However, due to the inherently digital virtuality in IS engineering, the implications and use of materials are generally considered less resource-intensive. Nevertheless, the experience gained is highly valuable for possible adoption of our tools in the context of production. For instance, Chapter 4 describes our model-driven approach to link the data of various services to create new user interfaces. The fundamental premise thereby is the standardization of application programming interfaces in terms of a common description language. As we will show in Section 4.3, even the currently ongoing radical technological shift from resource-oriented architectures to query-based requests does not disrupt the core functionalities and achieved interoperability, but instead extends the applicability due to a model-to-model transformation.

The ongoing *Cluster of Excellence Internet of Production* at RWTH Aachen University set up a research agenda for the forthcoming years to work towards the eponymous vision. The vision of the Internet of Production (IoP) is to enable a new level of cross-domain collaboration by providing semantically adequate and context-aware data from production, development and usage in real-time, on an adequate level of granularity [RWTH18]. Furthermore, the project pursues an agile research

management approach, enabling a continuous adaption and advancement of the initial roadmap. To this end, we see parallels to our digital ethnography approach, where the researcher is part of the community of practice, as described in Section 2.4. Following this approach, we therefore regard the Internet of Production as a worthwhile future field of exploration for the DevOpsUse method and tools developed in this thesis.

2.8 Conclusion

In this chapter we have layed out the methodological foundation for the rest of the thesis. We first presented the two research areas of infrastructuring and end user development. We then introduced the ATLAS theory for community-driven information system development that integrates all community members and is driven by a self-monitoring component driving the further community evolution. We showed how Renzel extended ATLAS with the MobSOS toolkit for a rigorous community success modeling [Renz16]. We analyzed DevOps, a life cycle model that promotes closer collaboration between developers and operators, and have shown that even though the agile principle behind it is driven by the notion of closer end user orientation, it does not explicitly include the group of end users themselves. This becomes particularly evident when looking at the advantages of automation in continuous delivery and deployment. Thus, an approach combining both DevOps and ATLAS may automate the involvement of end users in the various aspects of their respective life cycles. For this purpose we have developed the DevOpsUse life cycle model that is community-driven and inspired by state-of-the-art software development methodologies. In particular it stresses collaboration between stakeholders. We presented how technology-enhanced learning is a fitting research subject, as it impacts the whole society through the ongoing digital transformation of workplaces. Then, we introduced the Web as our infrastructure testbed, enabling various architectural styles like request-reply systems and peer-to-peer computing. Finally, in the previous section we discussed how the production industry is currently facing manifold challenges due to the changing technology landscape; we are confident that the adoption of our DevOpsUse methodology may unlock the full potential that Industry 4.0 promises in terms of productivity [SPW*14].

In the remainder of this dissertation, we present the various components of the framework and show the prototypical tools we developed and used in trial communities for evaluation purposes. For every component, we show related work, before we detail on how it is embedded into a community-driven life cycle. Subsequently, the evaluation of the respective tool is presented. Finally, we demonstrate that the open character of DevOpsUse can also be applied to further use cases that are community-driven. As first component, in the next chapter we take a look at continuous innovation. In particular, we are interested in how we can scale up the requirements engineering processes to large professional communities.

Chapter 3

Scaling Continuous Innovation

Agile methods have significantly increased the speed of software development, delivery and deployment over the last years. While initially the software was bundled with the hardware, later portable storage media allowed it to be distributed independently, e.g. on a floppy disk or CD-ROM. In today's smartphone ecosystems, software is distributed over so-called *app stores*, where users browse through an excessive repository of available software. From there, users select apps to be installed on their devices, based on popularity votes and reviews from other customers, which demonstrates a glimpse of the new social dimension of software distribution. Commonly, the app store handles future updates of the software package by downloading new versions and installing the application at a time most convenient for the user; e.g., when the device screen is turned off and no user activity is perceived. Often the user does not notice that a new version of the software has been installed. Or he or she notices it because bugs suddenly appear; these are precisely the moments when the underlying infrastructure comes to light. While software distribution models were changing drastically over the last years, the feedback channels from user to developer have not followed in the same pace, though reviews are now being more visible in the app stores. The slow, iterative steps that are needed for community-driven design were not even possible with monolithic applications before the Web 2.0 era. Therefore, even though the whole agile life cycle was apparently enforced to better cater for the needs of users [FoHi01], there is still room and huge potential to improve the actual involvement of users through new and better ways of interacting with the users *throughout* the life cycle. What can be observed is that users turn to app stores to review applications, in particular when bugs appear or functionality is missing. According to Pagano and Maalej, around 30% of those reviews are in fact requirements [PaMa13]. However, these comments are not considered the best feedback channel for developers in terms of categorizing input and supporting users by answering their questions. For this reason, new and innovative solutions are needed to improve user involvement in the software design and development phases, and beyond.

As for frequent software updates, the architecture of Web applications is inherently more flexible compared to native applications on a desktop operating system. As opposed to software that needs to be installed, the resource-oriented delivery model of the Web follows a different model; documents and data are requested from a Web server on-demand before they are transferred to the user and then displayed in the user interface. This is done to the extent that, besides data, certain parts of the application code are downloaded only when they are needed. In the context of modern Web

applications, this distribution model allows the rapid roll-out of software. At the same time, the connected nature of the Web enables sophisticated analytics and fast feedback channels from user to developers and designers. In this sense, the key idea described in this chapter is to keep the communication channel for requirements open throughout the software life cycle, and to make it simple to participate in the evolution of information systems. This allows to reach out to more users, and even to more diverse users in the *long tail*. The term long tail, introduced by Chris Anderson, describes a business model, that uses the distributions of a power-law distribution and tries not to focus on the few best-selling items, but on the large quantity of undiscovered entries down the curve [Ande06]. Conversely, this means that the more diverse an idea, the less support there is for it. At the same time, however, disruptiveness increases.

In this chapter, we present how we empower communities to use their voice for getting their requirements developed. We cater both for popular requirements as well as the long tail of users with niche demands. The related work section comprises first the open innovation business model and its application to open source software, then social requirements engineering as a means to capture ideas in a Web 2.0 way, before we deal with applications for scaling co-design of Web applications. Subsequently, three design artifacts are introduced: Requirements Bazaar, the House of Quality, as well as Pharos. They all evolve around the research question, how to overcome the imbalance of the large number of end users on one side, and the limited number of developers on the other. While previous work on the ATLAS methodology particularly focused on the long tail of users with distinct, particular requirements, our goal is to extend the approach to Web scale. Requirements Bazaar is a *continuous innovation* tool, complementing existing *continuous* terminology in the realm of development automation like continuous integration and continuous deployment (cf. Figure 3.1) [DMGl07]. The social requirements engineering tool allows a social exchange of ideas on the ideation level, while making requirements traceable through ideation, conception and realization phrases. The House of Quality is an instrument for collaborative decision support. Based on the principles of *Quality Function Deployment* (QFD) that we will introduce in this chapter, it helps to identify the technical parameters of an information system that need to be adjusted in order to fulfill the requirements based on their prioritization. We then show, how Pharos supports designers by crowdsourcing design choices like the position of buttons to a large crowd of users. Finally, during runtime, we allow user feedback by establishing an input channel within applications, again using Requirements Bazaar. Due to open APIs, we were able to embed its functionalities into arbitrary websites using WordPress, as well as augmented and virtual reality environments. Overall, our methodology supports rapid prototyping and iteration phases.

This chapter is structured as follows. First, in Section 3.1 we start with related work about open innovation, social requirements engineering and crowdsourcing. We then present two systems developed in the context of this thesis, namely Requirements Bazaar (Section 3.2) and House of Quality (Section 3.2.1). In Section 3.3 we show the details of *Pharos*, a further innovative prototype created, which deals with crowdsourcing of co-design to large end user communities. We then conclude the chapter.

Figure 3.1: Continuous Innovation as new Player in the Software Development Life Cycle

3.1 Related Work

In the following, we present related work in the areas of open innovation, social requirements engineering and crowdsourcing.

3.1.1 Open Innovation

Traditionally, innovation within companies is pushed by in-house research and development departments. This is costly, as these put a lot of work in figuring out potential new technologies, and with them, business cases. A further negative point is that these ideas are often planned without meeting the real requirements. When the innovation process is opened up to other companies, and more importantly, the public, it is called “Open Innovation”. The term was originally coined by Chesbrough [Ches03]. Open Innovation naturally creates a duality between being too closed, when not letting in new ideas, and being too open, where ideas are flowing out of the company, potentially harming the original business model. If the company is too strict, it cannot take in innovative ideas, yet on the other hand, if it is too open, it might support competitors. To find this holy grail is therefore often the big challenge of open approaches. According to Chesbrough, open innovation is particularly relevant for companies with short innovation cycles.

In particular, open innovation principles apply to open source software. A detailed analysis of open innovation processes in open source communities is performed by Hannemann [Hann14b]. Tuomi sees open innovation strongly related to open source software [Tuom01]. He argues that within open source, knowledge is created and reproduced in communities, therefore knowledge creation only makes sense within such communities: “this view rejects the idea that knowledge can be decontextualized” [Tuom01]. Tuomi further argues, that the core, i.e. the pivotal point of the open innovation strategy needs to be institutionalized. Therefore, it is particularly important to standardize the tools revolving around open source, and to keep them running well, so that in the periphery, innovation can be created. For instance, in terms of the software development life cycle, central infrastructure such as code hosting repositories and continuous integration and deployment tools need to be set up and fixed early on. In the case of an ecosystem of tools, a central single sign-on infrastructure may help to foster new applications faster, as users do not need to sign up for each and every new application. Yet another example on the technical level are software

modules: once some core components are established, new innovative use cases can be developed by combining them. For instance, the often volatile frameworks around Web widgets have not allowed the creation of a standardized approach to create reusable user interface widgets that can be used on multiple Web sites. Now, that the HTML5 APIs around Web Components have stabilized, long-term innovation through combination of user interface widgets is possible.

Integrating end users directly into open innovation processes can take several forms. *Netnography* and *crowdsourcing* are two possible approaches, that both enable scaling the number of users significantly. Netnography, a compound word of Internet and ethnography, stands for the transfer of ethnographical principles onto the Web by observing online communities. Within computer science, information retrieval is a related discipline, where data sources on the Web are collected and indexed in a quantitative manner. In crowdsourcing, defined tasks are distributed to a 'crowd' of workers who are able to work on them. It refers to the integration of a *crowd*, i.e. an inherently large number of users, into various *sourcing* processes, where monetary or ideas flow back from the crowd to the originator. In short, it aims to assemble a crowd to use the collective expertise of a large number of individuals [Howe06]. In the following, we expand the term to a general-purpose problem-solving method, following the approach by Doan et al. [DRHa11]. They present nine dimensions or aspects for the classification of crowdsourcing as listed below.

- **Nature of collaboration:** *Explicit* if the user is aware that he or she is belonging to a crowd and its collaborative nature, e.g. through rating or commenting. *Implicit* if the collaboration is not a goal in itself, but rather a secondary effect.
- **Type of target:** It is defined by the system owners, for instance building artifacts or executing tasks.
- **Recruitment of users:** Can be done by requiring users, paying users, asking for volunteers, make users pay for service and piggyback on the users' traces of a well-established system. After recruitment, the users should be encouraged and retained. There are many different retaining schemes, e.g. through immediate gratification, pleasant experience, necessary service, establishing a reputation or ownership.
- **Type of contributions:** The challenge here lies in defining the possible types of feedback. The factors that need to be considered are the difficulty for users and for bots, the impact, and the complexity of the user interface.
- **Result:** How can the massive numbers of contributions be combined? For numerical input, an average can be built, but textual feedback e.g. from reviews needs to be combined differently.
- **Evaluation of users and contributions:** This mostly involves managing malicious users, e.g. by deleting, blocking or banning them.
- **Degree of manual effort:** Can go from simply providing ratings to more substantial feedback, e.g. providing drawings or source code.
- **Role of human users:** Slaves for simple divide-and-conquer tasks, perspective providers for giving possible new insights, content providers that contribute new content like images or

videos, and component providers where humans act as components in the target artifact e.g. a social network. Humans often play multiple roles in the crowdsourcing system.

- **The architecture of the system:** Can be standalone or piggyback; in the latter, data from an existing system may be used, e.g. for correcting auto-correct entries in a search box.

The Web has increased the availability and accessibility of users, therefore it enabled crowdsourcing on the large scale that it needs to succeed. There are now several platforms on the Web on which tasks can be placed. The best known is *Amazon Mechanical Turk* [BKGo11]. Another marketplace for crowdsourcing task is *Topcoder* [LGLo10]. It is specialized in opening up tasks related to software development and data science. Section 3.3 is giving more examples of externalizing design processes in the development life cycle to a large crowd of users.

Lastly, citizen science can be mentioned as another approach of crowdsourcing tasks within the context of science. The term stands for opening up the collection of research data to the general society, which serves two main purposes. On the one hand, the amount of input subjects is significantly increased. On the other hand, and possibly much more importantly, awareness for science is created within the minds of participating citizens.

In this section, we got to know Open Innovation as well as its several possible forms. However, the approaches presented are focused on the core topic of attracting external innovators and binding them. We believe that for a sustainable open innovation strategy, the opening must be well-integrated into the methodological foundation. This applies in particular to its measurement and analysis. One possibility in the special case of software development is social requirements engineering, which is presented in the next section.

3.1.2 Social Requirements Engineering

Requirements engineering deals with the collection and management of requirements for software engineering [Pohl10]. There are various tools that support the requirements engineering processes. In industrial practice, most widely used tools are highly specialized, such as *JIRA* by Atlassian and the issues functionality within *GitHub*. The target users of these platforms are experienced developers that are familiar with the terminology such as *user stories*, *epics* and *issues*. Non-developers are more likely to avoid participation in these essential requirements engineering tasks [DRN*15], in particular if the usability is not adapted to them. What we often perceive instead, is that users enter their feedback as reviews to the distribution channels in app stores. In an empirical study, Pagano and Maalej analyzed that these app stores comments typically contain multiple topics ranging from bug reports to feature requests [PaMa13]. An issue with traditional requirements engineering techniques is that they do not scale to a large number of users. Another downside in traditional approaches is that requirements engineering is restricted to organizational boundaries. Even if the requirements collection is opened up in co-design sessions together with potential customers, it only affects active stakeholders. What is not considered is the context of communities. The innovation potential of niche communities without direct access to the requirement analysts remains untapped.

All these points are the motivation for *Social Requirements Engineering* (SRE) [LCRK12]. It refers to the opening up of the requirements engineering process in the sense of open innovation. It serves

both providers and end users. Providers, or app developers, get to know the real customer demands. End users are able to submit their requirements, and therefore get their requirements developed. Ideally, both are enabled to start a discussion to clarify the specific embossing of the requirements. To accomplish that, often existing requirements engineering tools that are specialized on supporting the software development are opened up to end users. However, they are not adhering to the lingua of end users, and therefore users may feel lost in the tools. For instance, it is hard to categorize the input of users into the correct software module, as for the user, it often makes no difference or is not evident whether a backend or frontend component is affected by a bug. Another issue affects the unintended creation of duplicates. For example, though many applications feature the functionality to mark an issue as duplicate, they do not allow the automatic detection of duplicates through tags or text mining approaches. In general, social requirements engineering employs social software concepts known from the Web 2.0. These include commenting and sharing. Sharing makes particular sense with regard to specialized requirements engineering software, to facilitate the adoption for developers. But SRE can also unfold its potential when it is embedded into the whole software development process and life cycle, for instance by making it accessible from within an app. In an e-letter by the IEEE Special Technical Community on Social Networking, various aspects on large-scale social requirements engineering are presented [ReKl14b]. A related topic in the research area of requirements engineering is CrowdRE [GSA*17]. It describes automated or semiautomated approaches to integrate a large number of users into the requirements engineering process [GDAd15]. Its goal is to allow the crowd to give feedback and discuss their ideas. Here, we see a key difference to the aforementioned social requirements engineering and our DevOpsUse methodology: While CrowdRE is explicitly discussing pull/push feedback patterns where developers as outside stakeholders monitor the usage and derive requirements from user feedback [GSA*17], in DevOpsUse we consider the developers as part of the community. In particular, requirements gathered from long tail communities are explicitly part of our methodology.

3.1.3 Scaling Co-Design

In addition to the requirements collection phase, the software development life cycle offers further stages that are applicable for involving end users by open innovation practices. The visual design of an application can be mentioned here naturally, since the user interface is the first point of contact for end users. In this section we therefore analyze existing research work and tools, that allow for the inclusion of end users in the design phase. In particular, we look at crowdsourcing for this type of collaborative process in order to involve a large number of participants. While the term primarily stands for different websites where money can be collected for disruptive ideas executed by startups other organizations, in recent years more and more tools have been created that also use the term for collecting training data for machine learning tasks. In the following, we use it for opening up the visual design of software. Using crowdsourcing in the design process has the potential of massive participation towards fulfilling the expectations of users.

InVision is the first representative of a class of collaborative platforms for designing prototypes together with users [InVi17]. The browser-based tool enables the creation of interactive prototypes of user interfaces together with animations for transitions between different views of the application. There are two different modes in the tool: build and preview mode. In the build mode, user interface elements can be drag-and-dropped around, while button or label sizes and texts can be

freely adjusted. On the other hand, the preview mode enables users to try out built designs with the aforementioned interaction possibilities through view transitions. Besides creating and editing designs, annotations can be added together with comments, to highlight certain aspects of the design. A digital whiteboard is the central artifact of participatory design. Boards store any kind of images, stories and ideas. Collaboration is achieved in both synchronous and asynchronous ways. Awareness measures for synchronous near real-time collaboration make sure, that all participants within a space can follow the changes of collaborators. Concerning asynchronous collaboration, notifications of changes, comments and annotations are sent to users. The code generation capabilities of InVision include the export of a CSS style file from the prototyping platform. In the free plan, InVision allows the creation of a single prototype only. Further pricing plans for enterprises provide better scalability in terms of the number of projects and number of team members; they range from 15 to 99 USD for a small team of up to five designers per prototype. InVision is primarily targeted to designers to improve the workflow and not having to exchange *Adobe Photoshop* or *Sketch* (both being popular softwares for the graphical design of Web pages) files from workstation to workstation. Integration of end users is limited to interactive click-through prototypes, that can be commented upon. However, there are no means included for managing large numbers of end users that can quickly occur in typical crowdsourcing systems.

Gallery by Google is a free collaborative platform for uploading designs, gathering feedback and tracking different versions of designs [Goog17b]. It is tightly integrated with other tools made by Google like Google Docs. The integration helps to manage the development of the design and related content. The tool is focused on giving support to implement the Material Design guidelines [Goog18], a styling system that aims at a uniform user experience across mobile and Web apps, independently of the device. Gallery therefore provides visualization and inspection capabilities for different device sizes and orientations. Collaborators can be invited that are able to change the projects and upload new images. Another type of collaboration is possible for being able to comment the previews to give feedback. Rapid prototyping by changing properties such as font sizes is not supported, therefore it heavily relies on external design software. Image files can be uploaded and their iterations can be versioned within the Web application. Feedback can only be gathered by comments through Google Docs. Towards the implementation of the designs, the tool offers links to developer resources like documentation, but no automatic code generation can be performed within the app.

DisCo is a tool by Walsh et al. that is targeted towards small geographically distributed teams working together on designs [WBJ*12]. It implements some ideas from participatory design in the context of remote collaboration. These are storylines, annotations, and comments within the tool. It concentrates on layered elaboration, allowing a team to focus on single design problems at a time. Similar to Gallery from Google, DisCo does not allow prototyping within the app. It is also optimized for collaboration within a small team of designers.

Participatory Design Online Tool (Pdot) is a prototype explicitly designed for online participatory design proposed by Heintz et al. [HLG*14]. The goal of the browser-based tool is to gather manageable user feedback in terms of comments, annotations, and sketches. The drawback of the tool is that users can only evaluate predefined applications without being able to create new projects. In this sense, co-design is not possible as users are not able to provide their own design files; they can only give feedback by comments and annotations. As future work, the authors propose to give designers aggregated analytics data of their users' activities, together with collaborative annotations.

Table 3.1: Comparison of Online Co-Design Tools

System Property	Framework					
	InVision [InVi17]	Gallery [Goog17b]	DisCo [WBJ*12]	Pdot [HLG*14]	CrowdCrit [LTW*15]	**Pharos**
Roles	●	○	○	●	●	●
Collaboration	●	●	●	●	○	●
Mass Feedback	○	○	○	○	●	●
Comments	●	●	●	●	○	●
Annotations	●	●	●	○	○	●
Sketching	●	○	●	●	○	●
Voting	○	○	○	○	○	●
Prototyping	●	●	○	○	○	●
Feature Requests	○	○	○	○	○	●
Sharing	●	●	○	○	○	●
History	○	●	●	○	○	●
A/B Tests	○	○	○	○	○	●

● = provides property; ○ = does not provide property; – = unknown

CrowdCrit is an online tool oriented towards designers [LTW*15]. It allows to submit early design proposals in order to collect feedback from a non-expert crowd. Afterwards, the results can be explored using different visualizations. The upload happens through image files, then contextual information can be entered. The actual feedback is collected through the use of small questionnaires rolled out via the crowdsourcing platform Mechanical Turk by Amazon. Finally, the designers get to see the aggregated data and are able to upload new images in an iterative way. While designers can start new crowdsourcing campaigns with a modified version of their designs, participants from the crowd are neither able to redesign proposals, nor do they see their feedback on previous versions of the prototype.

Table 3.1 presents an overview of the aforementioned online co-design tools and *Pharos*, our prototype that will be discussed in detail in Section 3.3. The desired system properties are on the left, while the right page compares the tools and whether they fulfill a certain property. In the following, we first provide an explanation of the mentioned properties, before we give an overall interpretation.

Roles refers to the capability of the tool to accommodate different subcommunities like designers, developers and users. *Collaboration* stands for synchronous or asynchronous collaboration capabilities such as shared editing and awareness of what others are doing. *Mass feedback* is possible if a tool allows scaling up to more than just the usual limited number of co-designers such as in face-to-face sessions. *Comments* are allowed by a tool if whole files or certain actions can be commented on, while *annotations* refers to selecting specific elements or coordinates in an image to add textual comments on them. *Sketching* is the possibility of free-hand drawings or annotations. *Voting* stands for the ability to have simple questions and corresponding visualization of results. *Prototyp-*

ing refers to the capability of doing in-tool drag-and-drop actions such as moving a button or label. *Feature requests* are contextualized textual comments asking for new features that are not yet represented in the visual designs. *Sharing* is the ability to export designs to third-party social websites. *History* refers to versioning and thereby provisioning of previous versions of designs. *A/B tests* are a common way in usability research to provide two focus groups alternatively designed versions of the same design in order to figure out which one fits better.

What can be observed from the table is that asynchronous or synchronous multi-user collaboration is supported by almost every tool. However, there are two classes of applications. The first class is specialized in versioning designs and discussing them within a small group of peers. The other class allows to open up the discussion to a large group of users through crowdsourcing. None of the tools support A/B testing out of the box, even though it is a valid and often used method to support the roll-out of new features. In fact, many app stores allow the similar procedure of *staged rollouts* of new versions of apps, where new functionalities are slowly rolled out to an ever increasing percentage of users. This allows the detection of bugs in real-world conditions that could not be tested in laboratory settings before the whole user base is affected; but it also allows reverting certain design decisions if there is immediate negative feedback. What is evident that there is a gap in the current state-of-the-art of tools combining the creation or modification of prototypes to a large audience, enabling designers, developers and users to create better designs together, and manage feedback. Section 3.3 describes our implemented prototype that supports all of the aforementioned properties (cf. the rightmost column of Table 3.1).

In this section, we presented related work in the areas of open innovation, social requirements engineering and co-design. In the following, we now discuss our implementations, that we used to apply these concepts in a community context. First, we begin with Requirements Bazaar, our tool for social requirements engineering.

3.2 Requirements Bazaar

Requirements Bazaar is an online continuous innovation tool developed at our chair that provides an ideation platform for communities, to discuss ideas for existing or newly developed products. It is driven by the principles of social requirements engineering, open innovation and long tail, to enable community members to express their ideas about a specific product or service in the form of requirements. These requirements can be discussed together with other community members, in order to select the most interesting requirements to be realized. The idea of the platform originated in the context of a DFG-funded project in the area of innovative software [KJHR11]. While Requirements Bazaar is mainly applicable for software projects, the idea is also applicable to societal matters in general. It was originally developed because existing requirements engineering solutions such as JIRA turned out to be insufficient for the particular use case of letting any kind of end users work with the tool [ReKl14b]. For instance, professional issue trackers demand too many specific details that the user needs to enter, for example the specific software module or the severity of the bug or feature. The basic idea behind Requirements Bazaar is that it resembles popular social Web 2.0 tools like Facebook and Instagram, with functionalities such as liking, commenting and sharing. Therefore, it is a small learning effort for new users to get familiar with the functionalities of the tool. The structure of the tool follows a project/category/requirement pattern, where each project

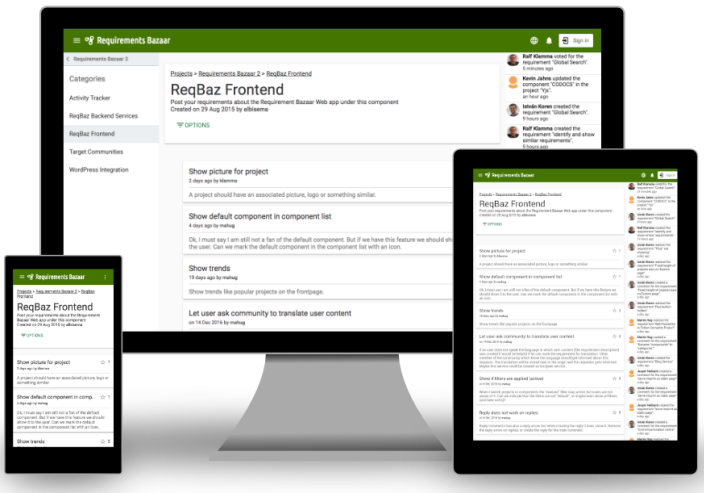


Figure 3.2: Responsive Web Design of Requirements Bazaar

can have multiple categories, and each requirement can belong to multiple categories within the same project. Any requirement can store attachments, such as drawings, screenshots or user interface mockups. Users can openly see ideas created by others without creating a user profile. To vote or comment on them, they need to log in. Sorting and filtering functionalities are available for lists of projects, categories or requirements. Thereby, the most active or alternatively the least active requirements can be highlighted on the screen.

We employed Requirements Bazaar in various collaborative research projects. First, the ROLE project[1] used it as a requirements management platform; in the Learning Layers project (cf. Section 2.5), we complemented it with the House of Quality tool that is presented in Section 3.2.1. Recently, it was employed in the WEKIT H2020 project to capture requirements of an innovative Augmented Reality learning solution.

Since 2011, three design iterations following the design science research approach have been developed, evaluated and improved. The first prototype was presented by Lai-Chong Law et al. with the use case of a widget store for personalized learning environments [LCRK12]. Every requirement could be voted on a five-step scale from -2 to 2. A second prototype was then developed in a master thesis by Behrendt [Behr12]. The prototype, its evaluation and third-party tool integration was discussed in various articles in an e-letter published by the IEEE Special Technical Commitee on Social Networking [ReKl14b]. In the second prototype, voting was possible using up- and downvotes. Finally, the current version that is deployed on https://requirements-bazaar.org was redeveloped using our community-driven DevOpsUse methodology, following the peer-to-peer service architecture of las2peer [KRLJ16]. It is relying on multiple las2peer microservices. As database it uses a MySQL instance hosted at RWTH Aachen University. The microservices are connected to the single sign-on system described later in Section 5.2.2. The main advantage is that users only need to create one set of login credentials and are then able to use them within various applications. Technically this opens the possibility to access Requirements Bazaar functionalities from third-party apps also

[1] cf. http://www.role-project.eu/

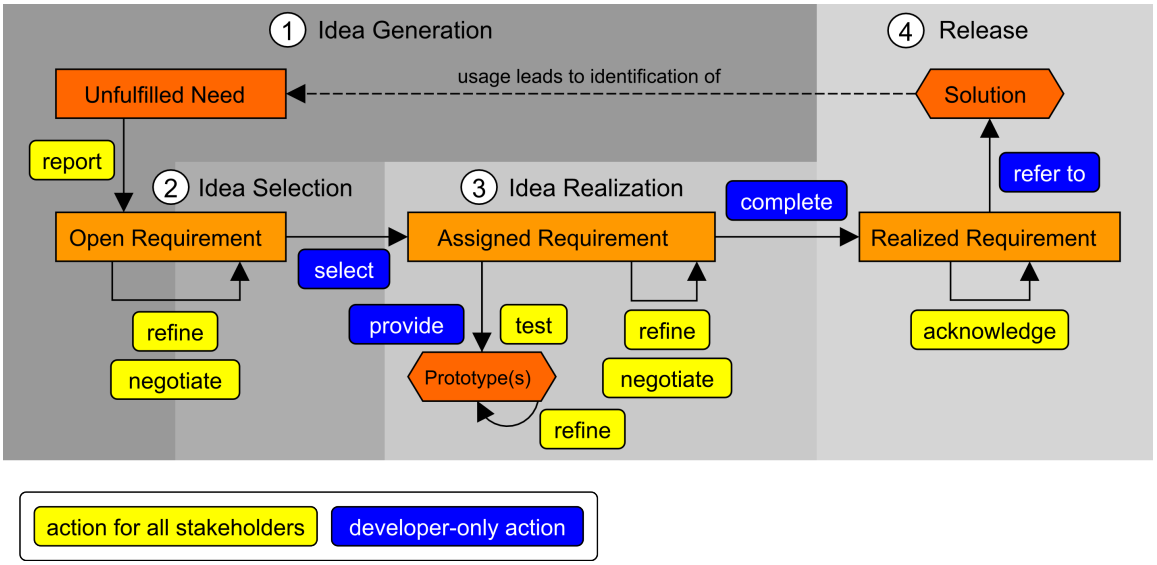


Figure 3.3: Workflow of Requirements Bazaar [RBKJ13]

using the same single sign-on system. On the frontend, a responsive Web application accesses the microservices via a REST-based interface. The user interface and retrieved data from the services are adapted to the available screen size, as shown in Figure 3.2. For instance, the navigation bar is automatically hidden behind a menu button once the available screen real estate becomes too narrow. The app can be saved on the home screen through the implementation as a progressive Web application (cf. Section 5.6.2). It then behaves like a native application installed on the mobile device.

Figure 3.3 shows the workflow of Requirements Bazaar. It starts with the idea generation phase, when a requirement is entered into the system. Once a developer decides to implement a requirement, he or she can be assigned as lead developer. Once the requirement is developed, the requirement can be marked as realized. Then, the user may check if the requirement was fulfilled. If yes, a comment can be left, otherwise the cycle starts again by entering a new requirement.

The original implementation was a monolithic Web application with a *Neo4j* graph database backend and a HTML application built on top of the *Bootstrap* user interface framework. In order to support and promote the agile development methodology DevOpsUse, we gradually introduced some changes in the software architecture. It is now built upon several interoperable las2peer microservices and is available at https://requirements-bazaar.org. Similarly, we applied the componentization paradigm for reusability across different usage contexts to the frontend. The current implementation supports reusability on many levels. The functionalities can thus be shared across applications on the Web. The *Activity Tracker* service was built as general activity awareness tools that can be used by various applications by calling its interface; therefore its service is not restricted to Requirements Bazaar events. It offers an interface that accepts activity descriptions, such as “Alice has created a new requirement” which are stored context-specific. A publish-subscribe mechanism forwards the activities to interested parties. Technically, we use the *MQTT* protocol, known from the Internet of Things, to notify subscribed clients. However, activities can also be polled. By relying on a REST interface, we support various application areas beyond the Web.

Each user interface element within the larger Requirements Bazaar Web application is developed using standalone Web Components. The resulting modularity allowed us to integrate the original components without overhead to third-party websites to display parts of Requirements Bazaar there. With the modularity we enabled a high maintainability, as only specific parts of the code have to be touched on changes, and they can be independently tested. An example of such reuse can be found on the WEKIT Community[2] webpage, as well as on the las2peer[3] website. Both embed `requirements-grid`, a custom element displaying requirements within a specified category. The single-sign-on solution turned out to be advantageous, as it nudges users to login to Requirements Bazaar that already have a user profile on connected pages. A recent addition is the possibility to integrate idea submission functionalities into Unity-based mixed reality applications. More details on this extension can be found in Section 3.4.

3.2.1 House of Quality

In the previous section, we presented Requirements Bazaar, a tool to continuously capture new ideas, bug reports and other artifacts from end users. To effectively turn these new insights, changing requirements, but also technological advancements into value in a product, it is essential to make sound decisions on which parts, respectively modules, to focus on, or in case of an existing product, which ones to change in a software. These actions need to be informed by both functional and non-functional requirements of all stakeholders. A technique that facilitates this is Quality Function Deployment (QFD) [Akao90]. It is a methodology that turns requirements into engineering features by matching a previously obtained list of weighted functional requirements with features offered by (optionally a subset of) the baseline technologies that establish the system to be produced. A specific QFD tool is the House of Quality (HoQ) [HaCl88], that we adopted and implemented for software engineering. The HoQ instrument has been used by many large companies, including Ford, Xerox and AT&T for their product development activities. The overall approach is to populate a matrix with weights, and to compare existing systems in terms of how well they meet user requirements. One of the main insights offered by a HoQ is an understanding of the importance of each feature based on the requirements. We successfully employed the methodology and tool in several European research projects over the last years. In the following, we show the basic structure of the House of Quality and present the developed tool support.

As the name suggest, the House of Quality spans a house with several spaces dedicated to various input factors. Figure 3.4 depicts the basic schema of the diagram. For constructing the HoQ, the list of user requirements from the social requirements engineering process can be used as a starting point. In our case, these are the outputs from Requirements Bazaar. This happens on the left part, where customer requirements are filled out line by line, together with a weight factor that represents their importance relative to the other requirements. Then, on the right wing of the house, competing products, for example found via an extensive desk-based research, are entered, one per column. Based on the previously entered requirements, the products are scored, typically with grades from 1 to 5, where 5 means that this requirement is fully met by the corresponding product. The most important part of the HoQ methodology is setting the requirements in relation with product features.

[2] https://wekit-community.org
[3] https://las2peer.org

This is achieved using a numbering system, which impacts the resulting ranking of features through a formula, taking into consideration the previously entered weight factors.

Figure 3.4: The Schema of a House of Quality

In order to effectively use the House of Quality methodology in various research projects, we identified and analyzed several existing tools from academia and industry, offering House of Quality functionality. However, to the best of our knowledge, no tool exists that offers a browser-based realization; neither commercial nor free. For this reason, we implemented an HTML5-based Web application able to run on the Web. To support user communities to fill in the matrix, we integrated near real-time shared editing functionalities. Technically, this was realized using *Google Drive Realtime API*, a JavaScript library that connects the application to a central replication engine offered by Google. This had the added advantage, that created matrices were directly accessibly from within *Google Drive*, a cloud-based storage system by Google. Due to the deep integration, the House of Quality app blended seamlessly into the familiar Google Drive user interface.

3.2.2 Case Study: WEKIT Community Idea Collection

To illustrate how the continuous innovation process described above works in practice, we show below how it was structured and carried out in an innovative joint European partnership project in the field of wearable-enhanced learning. The process was started at the kick-off meeting of the WEKIT project (cf. Section 2.5) and was later continued offline. Project partners created accounts for Requirements Bazaar and started entering initial requirements extracted from research project contract documents like the description of actions. The process then continued on the Web through the means of the *WEKIT community*. The community is an instrument that collects relevant stakeholders interested in the topics, and connects them to affiliated researchers, designers and developers. A dedicated webpage for the community, together with outposts at popular social media websites such as Twitter, Facebook, LinkedIn, ResearchGate and YouTube was set up to collect new

and sustain existing members. During the first months of the project, the community was asked to submit ideas on the WEKIT community website. Technically, the idea collection was realized via a WordPress plugin showing content from Requirements Bazaar. To handle users, we employed the same single sign-on-based login system for both Requirements Bazaar and the WEKIT community (OpenID Connect, cf. Section 5.2.2). On the WEKIT idea collection page, people could enter their new ideas, as well as comment and vote on previously entered requirements.

Later, the technical partners responsible for the development of the prototypes have met in a face-to-face meeting, in order to specify their technological affordances as well as limitations. All the collected input has been processed using the House of Quality instrument. Considering the relevance of the collected requirements, the project partners designed and developed an AR-based technological platform for knowledge-intensive vocational training. The most promising scenarios for the project were implemented and the result was employed in multiple big trials organized together with application partners. During and after the trials, new requirements were entered into Requirements Bazaar and prioritized there. Besides requirements for the existing software, also new idea categories were still posted. Throughout the project, and as the scope, capabilities and influence of the WEKIT community grew, new requirements were elicited in Requirements Bazaar.

Overall, the special microsite on the WEKIT community portal, realized through the REST-based integration and a WordPress plugin, was considered essential to get feedback from the beginning from end users, designers and developers, and to involve them in the decision processes. Throughout the project, we continually reviewed and updated the resources in Requirements Bazaar to keep the list of requirements accurate and up to date.

This section showcased, how innovative solutions can enhance community participation in the requirements engineering phase of a project. In the following, we continue with the visual design of the frontend application, to fulfill these requirements.

3.3 Crowdsourcing Co-Design

So far, we discussed the continuous innovation workflow with Requirements Bazaar to capture and discuss feedback from end users, and House of Quality to classify the received input and generate a list of weighted technical attributes that need to be targeted. We now turn towards the visual design of Web applications. There are already various established means of integrating end user feedback in the visual design of an application. In usability research, these are for instance A/B tests to compare two variants of the same design and return the most promising one. Wireframing may also be performed in co-design sessions together with end users. However, most co-design approaches with real users tend to be performed with a limited number of users. In the following, we therefore go to the bottom of the question, how end user feedback can be scaled up to large numbers of users. A related field that involves a substantial number of customers is *crowdsourcing* [Howe06]. In Section 3.1.3 it has been defined as a crowd (i.e., large number) of users collaborating to work towards an artifact which then will be beneficial to the whole community. Within the realm of co-design and infrastructuring, communication between a large number of users can be challenging, in particular when coming from different backgrounds [MaBo17]. Thus, appropriate tools for managing this crowd are needed. In the following, we present *Pharos*, our prototype

platform to crowdsource the co-design of Web applications. It was developed in the context of a master thesis by Bonilla [Boni18], and published as journal article in the context of co-design in technology-enhanced learning [BKKl19]. One of our primary objectives is to fill the communication gap between designers and end users within their community. Thus, designers should be able to submit designs and request feedbacks not only from other designers or a limited number of users, but from an entire community of people with different backgrounds. They should be able to describe their own mental model of how the design should look like. Overall, this will help designers to create better designs to increase user satisfaction. On Pharos, every community member involved or related to a project should be able to build and maintain relationships to other people with common goals. The goals can be identified by gathering expectations and requirements from the crowd right from the beginning of the design process, thus integrating functionalities from Requirements Bazaar. Users are allowed to perform redesigns and show the evolution of the project visually. Finally, large-scale tests on the resulting designs can be started, as "collective wisdom is often accurate" [DRHa11]. By scaling the collaborative design process to accommodate large numbers of participants, it establishes connections between people from different backgrounds. Pharos allows users to modify screens, create annotations, and write comments that affect the design, functionality, usability and structure of applications. It also makes it easier for less experienced designers with little to no programming skills to show their ideas and share them with users to get feedback. Simultaneously, it helps end users to describe, understand and support their interests.

In the following, we show the development process of the prototype. We started with a survey rolled out to several professional designers that we found on dedicated design community websites. The goal was to capture the state-of-the-art of co-design and infrastructuring activities. We then present the requirements, the conceptual design and the implementation. Finally, we evaluate the platform.

3.3.1 Survey

To capture the current state and the potential benefit of co-design and infrastructuring from application designers and developers, we conducted an initial survey. The goal of the survey was two-fold. On the one hand, we were interested in what practices, methodologies and tools application creators employ that involve users of their products. On the other hand, we wanted to gather information, on what they think could be an additional useful tool in the software development cycle; i.e. what functionality they expect from a support tool that is able to scale up feedback to potential large crowds of users. In the following, we present the survey design and outcomes. The survey was carried out on Google Forms and was open for four weeks in the beginning of 2018. To recruit participants, we sent out several posts on Twitter, as well as created forum posts in different design communities like *Open Source Design*[4], *UX Mastery Community*[5], *Web Designer Forum*[6] and *Site Point Forums*[7]. Additionally, we distributed the survey link among colleagues and personal contacts. There was no exclusion criteria like age or current job position. Out of the twenty-two questions, nineteen where closed-ended to ensure that the survey could be finished in reasonable

[4]https://discourse.opensourcedesign.net/ (last accessed on 2019-06-10)
[5]https://community.uxmastery.com/ (last accessed on 2019-06-10)
[6]https://www.webdesignerforum.co.uk/ (last accessed on 2019-06-10)
[7]https://www.sitepoint.com/community/c/design-ux (last accessed on 2019-06-10)

time. Out of them, eleven questions used a five-point Likert scale; the remaining eight questions were single or multiple choice. The rest of the questions were open-ended. The questions were grouped into five sections: 1. demographics, 2. current techniques 3. infrastructuring 4. user feedback 5. remarks on prototype. In total, we received 34 qualified responses. The following sections present the results and the findings we derive. The results are presented in the following sections.

Demographics: The demographics section collected age, gender, and field data, i.e. to which subcommunity, either developers, designers, both or none, the users were belonging to. All the participants were 18 years or older; 25 participants were between the age of 25 and 34. 25 participants were male, 6 female, and 3 preferred not to say. 30 participants were developers, 15 were designers; 3 respondents had management or leadership positions.

Current Techniques: The second section asked about current design practices used by the participants for developing and designing applications, and how many applications they handle per year. 76% of the respondents said they are able to develop an application; 70% are able to design applications. Only one of the participants being in a management position is not able to develop or design an application. On average, 3.1 applications are designed or developed by the participants per year. Half of the participants develop applications for a company or organization, while 15 create applications for personal use and 13 for end users directly. Three respondents exclusively design apps for personal use. Around 80% of participants are not relying on tools that allow to develop applications without coding skills. While 32% would like to automate parts of the design process, none of the participants already use automation as part of their design phase.

Infrastructuring: In the third section, infrastructuring, we asked if the respondents were already actively involved in any infrastructuring activities, as defined above. Since the term itself can be confusing as it is more likely to be used in research, we have not explicitly mentioned it; instead, we asked the participants typical concepts like *design-in-use*. Half of the participants involve end users in the design process. Figure 3.5 shows the survey results concerning the time of end user involvement. 65% involve users throughout the entire process, while 8.8% work without asking users in the design or development phase. 30% of our survey participants allow meta-design in their designed or developed applications that allows end users to adjust parts of the app.

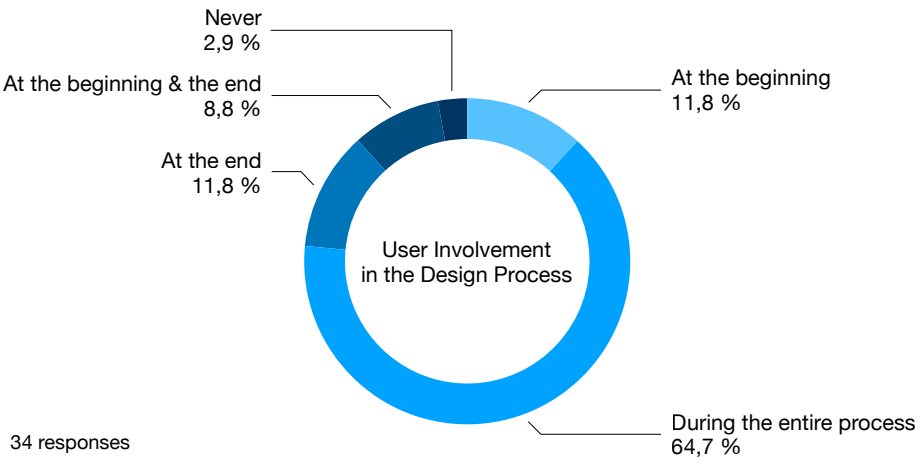


Figure 3.5: Point in Time of User Involvement in the Design Process

With regard to the affiliation to a company, we saw that most of the respondents working in a company or organization involve end users at some point of their design process. Concerning participants with both coding and design skills, only one does not involve users at any point of the design and development process. 19 participants develop applications on top of other systems already in use, indicating that a certain level of infrastructuring is already present in design.

User Feedback: The fourth section contained questions about the participant's methods for collecting user feedback. 85% of our respondents considered it important to obtain input from users. Only two did not consider user feedback relevant, out of which one is not able to either design or develop an applications. Two of the three participants that only design for personal use do consider user feedback important, the other being uncommitted about the subject. 70% allow user feedback to improve their designs. Concerning the amount of users involved, 12 respondents receive feedback from 1-5 users. Only 4 participants involve more than 20 users. This is in contrast to the target of 20% of participants who would like to receive feedback from more than five users. Seven participants are not interested in involving more users, one being the individual without design or coding skills. Five of these seven respondents work for a company or organization. 66% of our participants use interviews to gather user feedback, followed by using existing platforms and design workshops. Interviews are the most popular method to collect feedback, followed by surveys and A/B tests. Only one participants claimed that end users should not be considered at all when designing applications. Overall, the results of this part highlight the need for crowdsourcing tools to obtain design feedback from a larger crowd of end users. Currently, only three participants already rely on crowdsourcing design tasks.

Remarks on Prototype: In the last section, we described our goal of creating a tool filling the gap for crowdsourced design; we then collected expectations on the proposed platform. Overall, participants showed interest in gathering feedback from more than five users. 70% of the respondents were interested in a new platform that combines prototyping and gathering feedback from a large number of users. When asked about the expectations of a platform where they are able to prototype and crowdsource design feedback, our participants envisioned structured, categorized and summarized feedback, live updates, exportable logs, live crowdsourcsing sessions, versioning, and compatibility with existing tools.

The results of the initial survey asking 34 designers, developers and managers about crowdsourcing in design were very promising. It showed the need for a framework to allow end users to take an active part in the design process to extract their knowledge to offer more relevant and higher quality services. It became clear, that structured feedback is the most pressing requirement for crowdsourcing design, as stated by the survey participants. Improving the relationship between designers, developers and end users is worthwhile to enhance user experience on the long run. In the next section, we show the conceptual design of our crowdsourcing design platform Pharos.

3.3.2 Crowdsourcing Design Concept

Co-design is often conducted in small groups with no fixed structure in order to get as much input as possible by taking every single opinion into account without missing any feedback. When simply increasing the number of participants, open-ended critique can lead to low-quality data gathered [LTW*15]. However, by obtaining structured feedback, designers can leverage the crowd

without diverse input getting lost in the mass, as studies have shown [LTW*15]. The proposed platform Pharos thus should enable designers and end users to express themselves on different levels of creativity. Designers will be able to generate better concepts to meet the expectations of the end users. As infrastructuring allows the redesign during rutime, it connects usage and participation in one space, our goal is to combine the concepts of co-design and infrastructuring, to seek the knowledge and experiences of users. The underlying concept of Pharos should remain open, to achieve more than just co-design, but to let users modify projects on various levels from style to functionality features, depending on what the actual project aims at. The main goal is to facilitate the collaborative design process between designers and a large number of users with different backgrounds from end users to professional developers. On a functional level, this includes prototyping, collaboration and managing a large amount of feedback. As designers seldomly recognize the importance of side communication channels [Parm17], we want to make them explicit by offering commenting tools and placing them prominently. The aim is to gain new insights into the design of applications to create more stable iterations of a project. As participatory design and infrastructuring are able to complement each other to expose and resolve issues [DaDi13], Pharos will not only allow for co-design, but it will also create a network of various collaborators with different roles. Therefore it supports evolving social networks between these people [IvDi14].

In the following, we expand on the user roles Pharos will support. The three roles are maintainer, co-designer and voter. They are involved at various stages and their tasks have different levels of complexity.

Maintainer: The maintainer is the manager of a design project. He or she seeks feedback from colleagues and users to involve them throughout the design process. The maintainer is responsible for setting up the initial workflow by creating a new project, and entering the initial idea, an early prototype, or both. He or she can also participate in the feedback process additionally to reviewing the feedback at the end. To structure the feedback received from the collaborators, the maintainer may setup simple preference tests or short questionnaires to share them with the user crowd. Collaborators can be invited as co-designers or voters. These roles are presented in the following.

Co-designer: Co-designers are able to create new visuals for the design project. They can also modify existing views of prototypes. Co-designers can give open feedback by commenting on the entire project or a particular view. By requesting new features or giving strong opinion on the features requested by collaborators, they have the ability to influence and push forward the overall structure of the project. Instead of changing existing views or uploading new versions, he or she can also annotate the prototypes by free-hand drawing annotations, or simply by voting on the design. Co-designers can share the project with a crowd of voters, whose role is presented hereafter.

Voter: The main role of voters is to complete design tests started by the maintainer. This can be a preference test, where voters choose between two different designs, or a detailed questionnaire test, where they answer simple questions about a single particular view. Voters can participate in these two test types without restrictions. They are also able to share tests with further users if instructed to do so by the maintainer at the initial introduction.

Figure 3.6 presents a UML use case diagram about the presented roles and their possible actions. We can see, that the actions authenticate, prototype, feedback and share are both performable from the roles maintainer and co-designer. The difference is, that the maintainer has administrative rights. He or she can also launch design tests that voters can access. The feedback task is subdivided into

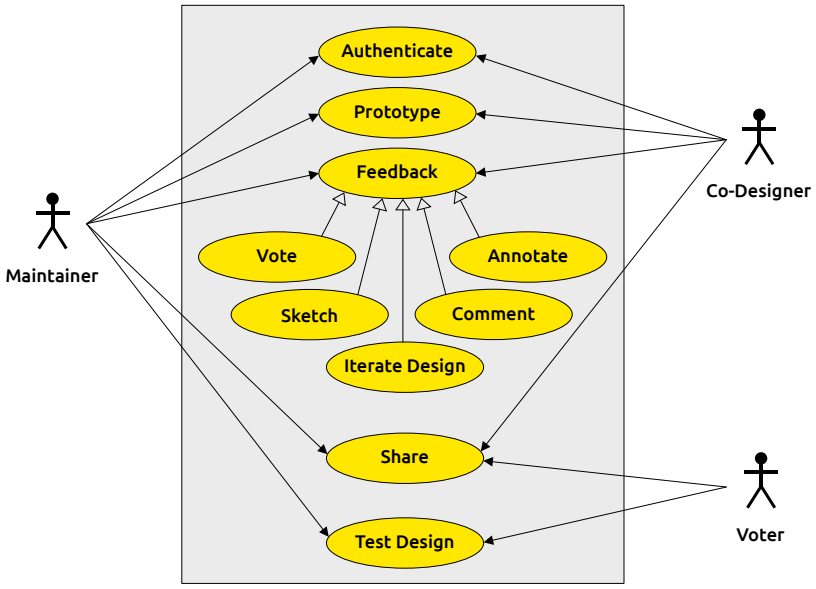


Figure 3.6: Use Cases of Pharos Prototype (adapted from [Boni18])

vote, annotation (and free-hand drawing respectively), commenting and design iterations (versions) We will see later, that the authentication of voters is done by the external crowdsourcing platform Amazon Mechanical Turk, it is thus not considered as part of the Pharos platform itself.

In the following, we expand on the functional and non-functional requirements needed to meet the previously stated goals. We based these requirements on the previous related work as well as the formulated expectations by our survey participants.

Collaboration: To ensure that designs meet the requirements and demands of end users, but remain implementable by developers, a close collaboration between different groups is needed. On the platform itself, the user groups are the three roles presented above. Pharos enables maintainers and co-designers to work on prototypes, place annotations and submit comments about ideas and raised conflicts. The initial survey found that only 9% of the participants already use crowdsourcing to meet end user demands. At the same time it has the potential to provide completely new ways of looking at things as input. Therefore we want to include the crowd into the collaboration. We chose to do so via the role of voter. They can perform two types of tasks, being preference tests and questionnaires. These kinds of structured tests can be easily overlooked by the maintainer and co-designers. To enable collaboration between all users, new designs can be created by anyone regardless of their programming skills. New views of a project are added by using a prototyping tool with drag-and-drop functionality, or by simply using an upload form.

Comments: Co-design often requires designers and users being physically close to each other to discuss the raised issues and to gather their thoughts and needs [WBJ*12]. Separating them can hinder communication. Comments are an accessible way to obtain the thoughts about the design. Additionally, leaving textual notes on a design allows other collaborators to read and react to the comments, allowing for discussion threads about topics they would have otherwise not met. In Pharos, the maintainer and other co-designers are able to comment on the overall project, as well as on each view. They can reply, up- or download any comment and edit or delete their own comments.

Annotations: While comments give co-designers a way to express their thoughts in an extensive and detailed way, annotations and sketches allow for instantaneous, visual feedback. Additionally, it is possible to annotate more specific elements of a design like a button or input field. Thus, annotations are a simple way to give feedback, e.g. about repositioning a button. It also allows adding short texts on top of the proposed design.

Votes: While comments and annotations require co-designers to engage with the design in detail, voting on simple questions allows a much faster feedback channel for end users crowds. The maintainer can quickly visualize the responses and decide based on them. Voting is enabled for all types of users. Co-designers may upgrade or downgrade various views of the design, and rate comments and annotations. They are also able to quickly give their opinion on feature requests when under time pressure. For users with the role of voters, voting happens through preference or question checks published by the maintainer, as explained above.

Feature Requests: In Pharos, designs are organized into projects, thus collaborators can not only give feedback on the visual design, but also modify the structure and functionality of the project itself. For instance, they can be connected to other services or platforms. As early phases of a project allow more openness as compared to later stages, we decided to introduce a requirements service to identify early expectations of an idea [GrVa17]. In the platform, contributors are able to request any new feature or even demand the removal of features considered not necessary. Therefore the feature request section enables discussions between maintainers and co-designers about the direction of the project in terms of features and requirements.

Prototypes: We aim for a flexible management of prototypes. Early designs should require little effort to create and share. Maintainers and co-designers should be able to easily create designs that can then be modified with minimum effort by themselves or other collaborators. After this, co-designers should be able to modify the designs and create different versions of them. In the new versions, new functionalities or different mental models of what the design should look like can be accomplished. Prototyping designs is available for both maintainers and co-designers. Thus, it is not limited to people with development skills. Any user should be able to create new views or new variations of a design to participate in the design process. Creating a new design prototype is the most time-consuming and engaging type of collaboration on Pharos, since co-designers are not only commenting on an existing design, but create a new version based on their own mental model. Therefore prototyping is the most meaningful action on the proposed platform.

Versions: The prototyping tool allows creating new designs and new views of a design. To effectively manage the different versions of a design, we aim for a versioning system similar to that of a source code repository. Thus, whenever a maintainer or a co-designer changes the design, a modified version should be saved as a new version. This will enable all collaborators to see the evolution of a design. It also enables to see which parts of a design remained stable over versions, and which parts had to be changed several times. By that, designers are able to see what needs to be worked on to convert it to a stable part of the design. The changing characteristics of a design should be highlighted across its versions. When a new version is created in the prototyping tool, it becomes the base for any further co-design activities like adding comments and annotations. Depending on whether the design artifact is an image file or a view prototype, the new version is of the corresponding type. Versions should also enable maintainers to easily compare different versions of a view and track differences. All versions are visible and accessible to the maintainers and co-designers.

Design Tests: Following the requirement of structured feedback that is necessary for large-scale crowdsourcing efforts, we aim for a solution that can give concrete results. While end users may be aware of the general requirements of a project, they might not be able to express it correctly to developers and designers [XHBa14]. Furthermore, we found in the initial survey, that the most popular forms of end user involvement are interviews, followed by surveys and A/B tests. Thus, we aim to integrate these forms of end user participation in our platform. In line with this requirement, maintainers should be able to test more specific attributes of their designs by being able to ask direct questions to users through design tests. There are two variants of design tests. The first is a preference test, where maintainers can choose two different versions of a design and users can then select their preferred one. Varying text sizes or positions of buttons, images or sample text could be the difference between both versions. The second type of design test are questionnaires. This enables to gather feedback, as well as to collect an understanding of a specific design characteristic. Together, this helps maintaining a structured feedback.

Non-Functional Requirements: We conclude the list of requirements by looking at the non-functional requirements of the Pharos design crowdsourcing platform. They are derived from common non-functional requirements of Web applications and are discussed in the following. First of all, Pharos should be an open source application, thus all the code developed is available for review and modification from the community. As the proposed platform targets a number of users ranging from developers to non-expert end users, we aim for a high user experience. The user interface should therefore be straightforward and not complicated, avoiding a steep learning curve. Pharos enables distributed co-design by allowing design teams to be geographically dispersed so that high availability is required at all times. The platform should be available over the Web without requiring users to install a specific software on their client devices, other than a common Web browser. We aim to provide speed and stability, to make the process of creating new projects and saving new designs fast. Furthermore, errors should not lead to a complete shutdown of the platform to ensure stability. Finally, while we do not expect large numbers of users with the role co-designer to work simultaneously on the Web application, it needs to handle large crowds of workers for the design test functionality. The latter is the main target of Pharos.

These non-functional criteria conclude the requirements analysis of Pharos. In the following, we continue with implementation aspects.

3.3.3 Implementation

Based on the functional and non-functional requirements established in the previous section, we discuss the implementation of Pharos in the following. We first present a detailed description of the architecture of Pharos. Then, we turn towards its components and the database used. We explain how all services and components were developed and which technologies we used. Figure 3.7 shows an architectural view on the Pharos system for design crowdsourcing. On top, we see the extensive frontend. In the center, a relatively small backend is shown. On the bottom, all the third-party services Pharos is using are included.

The main element and user-facing component of Pharos is its frontend, that was developed as an HTML5 Web application using the *Angular*. Angular is a state-of-the-art Web framework for building rich client applications with HTML5, CSS and JavaScript. It is developed using the program-

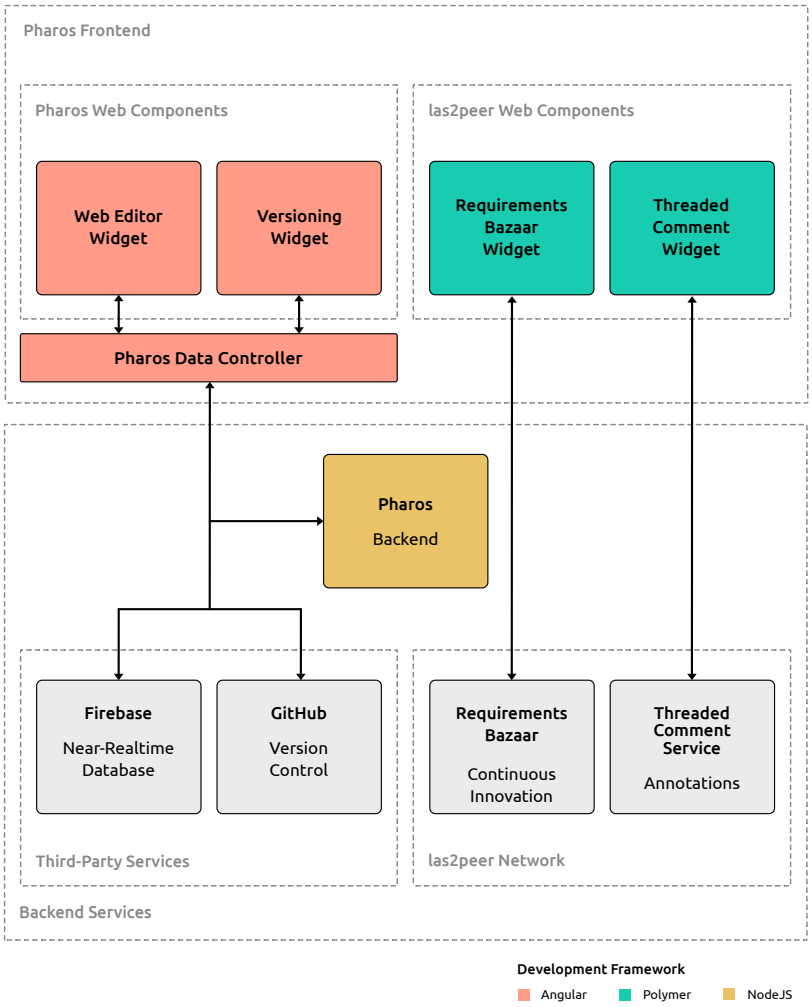


Figure 3.7: System Architecture of Pharos (adapted from [Boni18])

ming language TypeScript, which can be transpiled into JavaScript that runs in any modern Web browser. The basic building blocks of an Angular application are modules, which provide a compilation context for components. Each module thereby consists of components. A component defines a view and associated application functionality. It further consists of templates or views, classes, and their metadata. To share functionality across components, services can be used, which are not directly related to views. We used Angular for several reasons. First, because of the code generation functionality of Angular we have the benefit of a clear separation of concerns between a model and its view. We do not have to manually synchronize the view with its model, making the code readable and maintainable. The model-to-view synchronization is transparent and optimized for being evaluated in the JavaScript engine of the Web browser. As Angular is working with templates that are only parsed once per HTML page, the user interface performance is increased. The parsing is done either at application load time or ahead of time when the application is built. The generation and modification of the view are done in an optimized way, so that only parts of the DOM tree with modified data are affected, while the rest of the page remains the same.

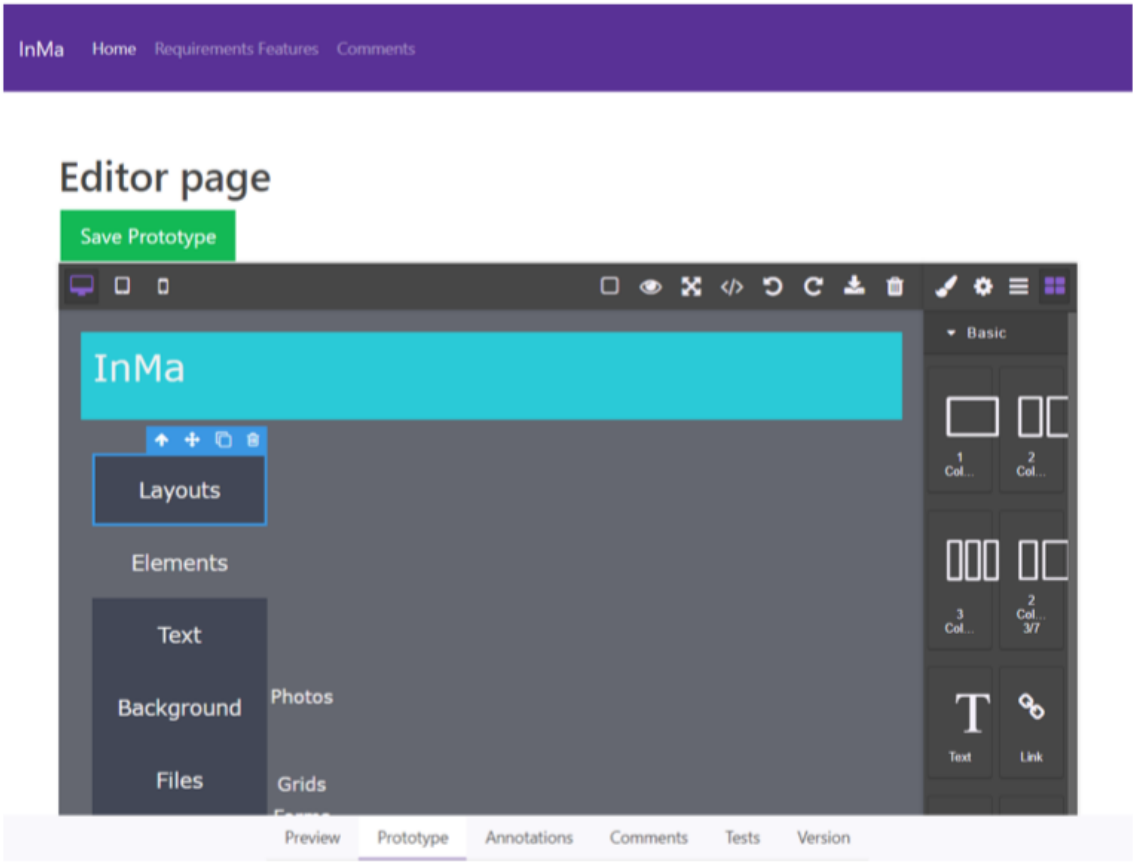


Figure 3.8: Web Editor Component of Pharon [Boni18]

The Pharos frontend project consists of three main parts: the components, the data services, and the data models. The main components are the project component and the screen component responsible for the different views of a design project. Internally, they in turn consist of several components for authentication, prototyping, versioning, annotations, voting and the design tests. On the backend, we developed a light-weight Node.js based application that exchanges version information with the GitHub backend. By using GitHub for storing the different versions of the design, we are able to leverage the git versioning system that developers are familiar with. In the near real-time Firebase database, we store any data created by the users of Pharos, for example annotations.

To fulfill the requirement of feature requests within Pharos, we decided to reuse functionality available through the Requirements Bazaar API. Additionally, we were able to reuse existing Web components that are in use on the main Requirements Bazaar website. In order to use these components in Angular, it was necessary to use the external Origami library that makes Web components available to the programming model of Angular. The two added Requirements Bazaar widgets are for contributing new ideas as well as for commenting. They are accessing the Bazaar API with the help of the user profile that is also used for managing Pharos users.

3.3.4 Evaluation

In this section, we present the evaluation of Pharos. First, we discuss the setup of the evaluation before the results are listed. In total, we recruited 121 participants to our user evaluation. Two of them were assigned as maintainers of a project, nine users were placed as co-designers. The majority of the users, exactly 110 people, were part of the crowd. To use Pharos, users had to authenticate with both their Learning Layers OpenID Connect single sign-on solution, as well as a GitHub account to store the different design versions. To prevent user reluctance when using their personal GitHub accounts, we created several test accounts on GitHub. They were then added to the Pharos organization before the tests to authorize them to push new designs. As the main task of our evaluation, all users were asked to work on the same project about a fictional prototype of an infographics platform. Infographics are visual representations of information, data or knowledge; their goal is to be present their underlying data quickly and clearly. The name of the prototype was *InMa* and users were told to co-design this platform.

In order to test all three user roles of maintainers, co-designers and voters, we divided the evaluation into these three parts. First, we began the user evaluation with the maintainers. They were asked to participate in two areas of the evaluation. In the first part, the project needed to be created together with an initial design and design tests. In the second part, the crowdsourced feedback had to be reviewed. We presented InMa as consisting of two views, namely the welcome page and the editor page. They had to create these two designs and were given the freedom to create their own designs with the prototyping tool, following their own ideas and mental model of how the prototype should look like. Then, the maintainers were given two test designs of the editor page in the form of images. With the help of these images, they created design tests to be released to a crowd of voters. The first maintainer created a preference test; in the first image, the toolbar was located on the left, while on the second image, it was located on the right. The crowdsourced task was then to vote on these two alternatives. The second maintainer created a questionnaire. As goal we wanted to know, whether users were able to locate the toolbar and one of its functionalities. Finally, the two maintainers analyzed the results of the design tests and gave their thoughts on the crowdsourced feedback.

Co-designers were involved in the second part of the user evaluation. After being explained all the functionalities of Pharos with a short description and a screencast, we introduced the idea behind InMa to them. Afterwards, the co-designers performed similar tasks as the maintainers. They were able to create new versions of the editor and also new views. We asked them to give feedback and ideas with all the tools provided, namely comments, annotations and feature requests. Six of these user evaluations were done remotely, three were invited to a face-to-face session.

Finally, to test the crowdsourcing functionalities, we recruited 110 non-expert participants through the commercial Amazon Mechanical Turk platform. They were given the tests created by the maintainers. The goal of this part of the user evaluation was to gather structured feedback for the maintainers using Pharos. All three user roles answered a System Usability Scale (SUS) questionnaire after the evaluation. SUS is a questionnaire used for measuring the perceived ease of use of a given system [Broo96]. It consists of ten questions that can be answered with a Likert scale. The average SUS score within industry standards is 68. A higher score can be translated as good rating, while a score higher than 80.3 represents an excellent score [LeSa09]. All resulting designs can be found in the GitHub organization of Pharos. The source code is available our research group's GitHub

profile.[8]

To evaluate the technical aspects of Pharos, we successfully verified its functional perspective with the definition of co-design by Bradwell et al. [BrMa08] and the guiding principles of infrastructuring by Pipek [PiWu09]. We also certified the key functional requirements to manage user feedback set by Heintz et al. [HLG*14]. In the user evaluation, different user roles participated in the creation and design of a sample project as described above. In general, all types of users found Pharos to be consistent and well-integrated. As average SUS score we received 77.7. We can consider this a positive result, as all users rated Pharos as a satisfactory experience. Among the three user roles, the voters showed the highest ratings for crowdsourced feedback, while the co-designers scored the lowest. Most of the concerns raised were directed towards the prototyping tool. We attribute this to our impression that co-designers did not seem to have developed real interest towards the infographics tool they needed to develop, which lead to less meaningful feedback. However, the maintainers preferred the structured feedback from the crowd over the less structured annotations from the co-designers.

The above evaluation of Pharos closes the discussion of the first part of the DevOpsUse life cycle concerning the requirements engineering and visual design phases. In the following we now present an innovative use case of the available Requirements Bazaar APIs outside the browser environment.

3.4 Requirements Engineering in Mixed Reality

The tools presented so far in this chapter revolved around HTML5- and JavaScript-based applications. Due to the provided interfaces, however, a use outside the browser is also feasible. Especially in the highly innovative field of mixed reality, i.e. the span between augmented and virtual reality, a focused orientation towards end user needs is important, as these technologies are likely to experience a wide adoption in the coming years. In this section, we therefore discuss the application of our continuous innovation approach within a gamified medical mixed reality application. In medical education, traditional formal learning concepts of anatomical courses include book illustrations and physical objects. While with images alone it is hard to convey a vivid understanding of spatial structures, physical objects are restricted since they are fragile and often hard to access in libraries and anatomy institutes. Interactive Web-based learning environments allow students to see, transform and annotate 3D models in a computer-generated world. Concerning the pedagogical aspect, research indicates that learning anatomy with a 3D model can improve the student's test results [NCFD06]. Additionally, students can use the tool in self-regulated learning [ScZi98]. Bedside teaching is considered an ideal clinical teaching modality [PeCa14], where medical students examine patient conditions in the patient's room. Augmented reality (AR) is of particular interest here, since anatomical models can be overlaid with the patient to display additional content. While AR enriches reality with computer-generated content, virtual reality refers to the total immersion into a virtual environment. In this context, mixed reality is the combination of these two concepts; we use this term to represent the whole spectrum between augmented and virtual reality [MiKi94]. In research literature as well as industrial practice, mixed reality is one of the

[8] https://github.com/rwth-acis/pharos

prime use cases for edge computing (cf. Chapter 5), as many operations are quite computing-intensive [SBCD09, Saty17, Micr19], creating a stark contrast to the requirement of having light-weight headsets that come with a relatively weak processor performance. Also, because of the high innovation factor of the largely unknown territory of mixed reality applications we see a special applicability and need for involving end users early on. However, users need to be motivated to participate in the co-design, as they are not yet accustomed to how interactions work, and what the benefits are using a particular app. The gaming industry has been a driver of virtual reality systems over the last year. Fun and the feeling of competition can be a motivational factor in the use of software. Gamification is defined in literature as the use of gaming concepts in non-gaming contexts [DDKN11]. It is the introduction of game elements like scoreboards and badges into serious software that has the potential of attracting and keeping users within the mixed reality applications. However, there is lacking developer support in terms of established best practices, gamification frameworks and user interface libraries that allow building gamified mixed reality applications.

In a comparison of existing applications and prototypes for medical 3D models [HKKH18], we saw that most applications are still limited to the desktop. Additionally, most of the tools work with pre-defined content like 3D models and quizzes for reflection. This restricts their applicability to the given fields of study. While students can use the reviewed applications for learning, they do not support the student in long-term engagement. This can be achieved by the concept of gamification. Thereby, elements from games like quest systems, points, achievements or badges help to motivate and reward users for performing tasks [DDKN11]. A sense of achievement is triggered by goals that have been set and the progress made in achieving them.

To bridge the existing gap in learning with 3D models, we developed the GaMR gamification framework that allows flexibility and extensibility on multiple levels. First, custom 3D models can be easily integrated. Second, annotations and quizzes to support the reflection phase can be created on arbitrary spatial content. Models within the GaMR framework can be viewed on the Microsoft HoloLens in mixed reality. We applied the gamification framework and successfully evaluated its conceptual and technical applicability with real medical students at Maastricht university. The resulting collaborative learning application is available as an open source solution.[9] During the evaluation sessions, we received multiple ideas that could be integrated into our prototype in later developments. In the user evaluation of the mixed reality framework performed at Maastricht university, we received many feature improvement suggestions. To gather feedback more systematically and formally, we then integrated Requirements Bazaar; the fusion of these two prototypes was only possible due to open Web-based interfaces. However, it was difficult to ask the students to submit their ideas on the Web-based version of Requirements Bazaar, as it led to a media discontinuity issue: Users had to either switch to the planar Web browser inside the HoloLens, or even by removing the headset from the head and transferring to a laptop or smartphone to open the existing Web application. With the media break, the contextual reference disappeared as soon as the headset was removed. As a consequence, we aimed for an environment, where ideas could be directly added within the mixed reality environment. Subsequently, the GaMR framework was extended with the possibility to directly enter requirements within the mixed reality environment. In particular, we now offer a standalone library for embedding a requirements input form in any Unity application. Here, we benefited from the existing OpenID Connect based user management

[9] https://github.com/rwth-acis/GaMR/

of the GaMR framework. As the app users were already logged in with their Layers login, we are able to reuse the security token to authenticate against the API of Requirements Bazaar.

Figure 3.9 shows a model of the fusion of the two applications GaMR and Requirements Bazaar in the context of co-designing in mixed reality [KHKl18b]. The swirl starts with the initial requirements in top left of the figure. Through co-design sessions (both co-located and remote), first versions are beta tested and evaluated. The working contexts and practices of user communities are highly influential on the development of the application. Monitoring ensures both a formative and a summative evaluation of the (learning) content. The other source for user involvement is through the use of gamification, introduced on the bottom right of the picture. Its exemplary artifacts are found in the center of the figure.

The library is available open source on GitHub.[10] The repository contains user interface elements to be used in mixed reality applications, an example is shown in Figure 3.10. We thus demonstrated, that we are able to combine different principles like gamification, mixed reality, requirements engineering and co-design with the open principles of the Web [KHKl18b].

3.5 Conclusion

In this chapter, we discussed how open innovation principles can accompany and moreover guide the software development in professional communities. First, relevant literature and terms were analyzed concerning social requirements engineering, co-design and crowdsourcing. We then showed, how the Requirements Bazaar and House of Quality tools were developed and adapted in order to support the DevOpsUse process. In particular, we stressed the open, modular and service-based architecture of Requirements Bazaar, itself developed using DevOpsUse best practices. In a case study, we described how both these tools were successfully used in the European WEKIT project. The main research of this chapter, meanwhile, evolved around the question, how we can include designers into software development in order to integrate their expertise in an open way. Therefore, we fused established practices from open source software development on the one hand, like versioning and collaboration; and on the other hand, we integrated design methodologies like participatory design and A/B tests. To test that conceptual design, we developed Pharos, a browser-based tool that allows creating and evaluating graphical Web application designs on the Web. To integrate

[10] https://github.com/rwth-acis/RequirementsBazaar-Mixed-Reality-UI

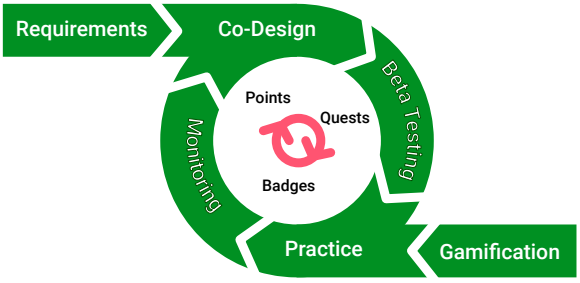


Figure 3.9: Intertwined Requirements and Gamification Life Cycle

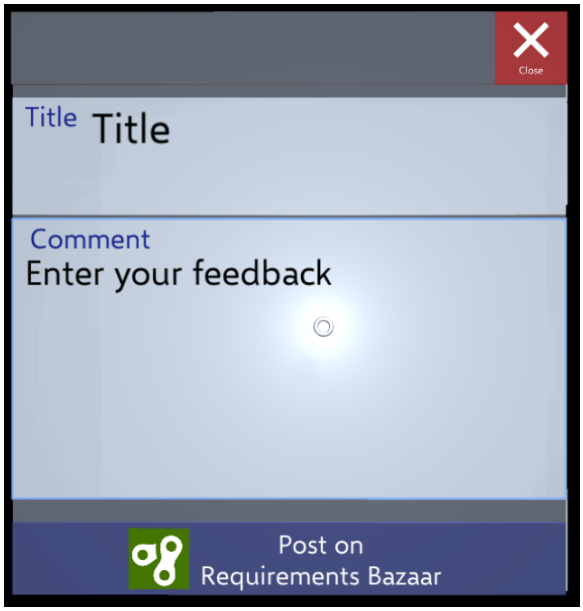


Figure 3.10: Screenshot of Requirements Bazaar Feedback in Mixed Reality

design-in-use principles following the ideas of infrastructuring, the application features a what-you-see-is-what-you-get (WYSIWYG) editor with drag-and-drop functionalities. Further, it interfaces with the popular crowdsourcing platform Amazon Mechanical Turk, in order to leverage the wisdom of the crowd for structured feedback in the form of preference tests and questionnaires. The initial survey carried out with real world designers, and the final open evaluation of our prototype showed, that the functionalities are both needed, as well as satisfactorily implemented in our tool.

The three Web-based software tools that were presented as contributions of this chapter have in common their prime focus on integrating further communities in the software development process, namely end users and designers. They all enable browser-based synchronous and asynchronous collaboration. Continuous innovation principles are important to keep up disruptive capacities of information systems and thus create a sustainable long-term development process, able to integrate new aspects in the future. We showcased that with desktop and mobile based Web applications. As wearable devices and Internet of things hardware are more and more connected to the Internet and through open protocols to the Web, we are confident that our methodology can also be extended to these use cases. In Section 3.4, we thus showed how we implemented feedback means for a mixed reality user interface. It enables continuous requirements engineering, even after users leave the environment of their Web browser and immerse into mixed reality. As a consequence, we conclude that the foundation relying on open interfaces can be reused with little effort even in proprietary, closed systems. Ultimately, therefore, our argument and recommendation is to rely on open systems for reaching sustainable societal impact.

The next step after agreeing on requirements and the design of an application, i.e. *what* needs to be developed, is the reification in code, i.e. the question is *how* to do it. To do this in a community-aware way, our goal, similar to the one described above, is the integration of end users. For this reason, the next chapter discusses how to scale information systems development to a myriad of

users and device types. We show, how model-driven methods and associated benefits such as generalization and code generation can be leveraged in a community-driven end user development approach.

Chapter 4

Model-Driven End User Development

In the previous chapter, we described how to support end users in contributing ideas, and participating in the requirements engineering part of the software development process. We also showed how we can support the design of Web applications using crowdsourceing to help designers adapt their creative works to community requirements. The next step towards the reification of community needs in terms of software is the architectural planning as well as the actual implementation, i.e. the creation of code. While for the requirements finding phases, end users can be integrated rather easily, as their contribution is the formulation of ideas in verbal or graphical form, the cognitive hurdle is much higher for writing code. Today, programming is to a large extent limited to a few experts with an appropriate training, i.e. software developers through self-study, or educational programs such as training and university courses. While indisputably creativity is needed for this, many information system functionalities follow repeatable patterns in their inner working. An indication of this is the use of design patterns in software engineering, as well as the ongoing componentization and modularization. And when functionality is externalized and made reusable, automation is not a distant goal either. Extensive service repositories, accessible via the Web, show how simple operations such as sending e-mails or even complex functionalities like text mining and pattern detection can be outsourced to third-party providers, often through the cloud computing paradigm. What is missing, however, is the possibility of easily adapting these out-of-the-box functionalities to a concrete community-specific information system, whether through customizing the design or adjusting features. This involves architectural design decisions, for instance, which services to develop internally and which ones to access remotely through interfaces.

While communities address the challenge of finding out, what their users really want, they often struggle with the development process. In future, the gap between the number of applications needed and the number of developers available will grow even faster. This scarcity may lead to increased costs for professional agencies and software developers. Thus one of the deductions is that together with the aforementioned automation, better tools for creating software are needed. Behind these, however, a structured approach must prevail. Model-driven software development techniques can be considered the baseline of a sustainable software engineering life cycle. Use cases of models in software engineering include validation, runtime interpretation, and code generation. Code generation refers to the automated creation of runnable code from models. This code can be used for various purposes, for instance documentation, configuration, and testing. While there are elaborate

model-driven technologies, many of the methods are highly expert-driven and not accessible to end users. Several authors highlight that even highly abstracting service-based modeling notations are very far from the practice of end users [DaMa14, NNAn10, NNAn10b]. Therefore, our main research question in the following is to what extent and how end users can participate in a sustainable model-driven software development life cycle in order to ultimately achieve continuous innovation.

In this chapter, we look at a model-driven methodology to create software artifacts. We start with a formal description of different parts of the architecture. Our architecture is roughly divided into a frontend and a backend part. This corresponds to the initial client-server model of the Web. To our benefit, many technical innovations have been introduced to the Web environment in the last years. *Web Components* is a recent group of standards that allows bundling reusable HTML5 user interface widgets with their functionalities in order to be shared between Web applications. While making the Web more flexible, they often pose challenges to developers. We believe, that on the long run, their adoption can only succeed if they are embedded into a community-driven development methodology, stressing their reusability across devices and use cases. Similarly, they can only be fully leveraged if they are part of a model-driven development methodology that again has reusability in its core. Another methodological foundation of our approach is embedding services through their API descriptions. For this reason, we analyze the state-of-the-art OpenAPI description language and show, how in combination with Web Components, modern Web user interfaces can be built that satisfy community tasks. Having end users work with real-world APIs is an essential strategy towards the realization of infrastructuring. In our Direwolf development framework, we bridge the gap between developer experts and end users by the means of collaboration. Based on a formal model of interface description languages and the Interaction Flow Modeling Language, end users may wire together functionalities based on their needs together with experts. In the following chapter we will then look at how to break up the dichotomy of client and server, i.e. HTML5 frontends and microservices, by introducing peer-to-peer technologies on the client side.

4.1 Related Work

This related work section is organized as follows. First we look at end-user-oriented development methodologies. We are particularly interested in metaphors that end users can use, to overcome the cognitive gap between the understanding of what a service can do and how it does it. Further aspects are existing systems that describe the possibilities for end users to participate in the development process. We conclude this section with an overview of technologies and languages that describe service interfaces and user interactions.

4.1.1 User Interface Modeling

Regarding cross-device user interfaces, our research review reveals that several approaches have been made to create universal user interface descriptions applicable to various application areas. The presented modeling languages are generally considered important in bridging the communication gap between developers and domain experts [LPKW06]. We introduce multiple representatives of user interface modeling languages in the following.

Mecano is an early system to automatically generate a user interface based on a domain model by Puerta et al. [PuEi02]. By analyzing the relationships among objects, the declarative interface model includes not only interface objects, but also dynamic behavior. The model is then translated into a concrete dialog. This process can be done together with domain experts as end users, to custom-tailor the layout. Bodard & Vanderdonckt introduced *Abstract Interaction Objects* as standardization of interaction objects on dialog systems at that time [BoVa96]. The goal is to overcome different concrete user interface widgets, in order to increase reusability and maintenance across operating systems and user interface toolkits. A prototype that is building up on this work is *Quid*, a Web-based what-you-see-is-what-you-get editor that allows end users to see the designed interface in near real-time [Moli19]. Internally, it uses Web Components for its preview pane. *ConcurTaskTrees* are a tree-like model notation by Paternò et al. to enable designers to define tasks [PMMe97]. They provide tool support for editing tasks and semi-automatically transform task models into architectural models for implementation. *JUST-UI* is a specification language for conceptual user interface patterns [MMPa02]. Molina et al. present it to extend object-oriented analysis with presentation and navigation information. A graphical notation is introduced to enable graphical designers to participate in the layout of the information system. Translators are available that generate user interface code for a number of programming languages. An endeavor targeting the Web is the *Cameleon* reference framework [CCT*03]. It defines a multi-step procedural model from task models over abstract UI specifications to a concrete UI, and uses the ConcurTaskTrees notation. *UMLi* adds user interface notations to diverse diagram types of the UML [SiPa03]. Besides the aforementioned modeling languages, there is a number of user interface modeling languages that are based on the declarative Extensible Markup Language (XML). *XIML* is a markup language and framework for the definition and interrelation of interaction data; the goal is to enable interoperation within an integrated user interface engineering process [PuEi02]. Similarly a markup language, *UsiXML* enables a model-to-model transformation workflow by allowing to model user interfaces on different abstraction levels [LVM*04]. The approach is incorporating the Cameleon reference framework mentioned above. Finally, the *User Interface Markup Language* (UML) is designed to manage a family of interfaces for different input modalities and device form factors [APB*99].

In general, a shift towards declarative description languages can be observed. Here, IFML is another representative, which is considered in detail in Section 4.2.3 because of its importance for our further concept. It is also interesting that some existing approaches like Mecano and JUST-UI provide for the possibility of integrating non-analysts, such as designers and end users. In the next section, we discuss related approaches that let end users model interactive behavior.

4.1.2 End-User-Driven Modeling

In literature, integrating end users into the development process is a widely discussed possibility of dealing with the stark differing number of needed applications vs. the number of available developers available to implement custom functionalities. Yet, the cognitive burden of existing programming languages and environments like JavaScript and Java is often too high for non-experts. With the help of objectification, programming concepts can be abstracted. Paternò & Santoro [PaSa17] divide the available prototypes and frameworks in two classes, end user development metaphors and end user development styles. Metaphors aim to lower the cognitive load of users as they are dealing with familiar symbols or games. Examples are click bricks (elements that are com-

bined with each other by plugging them into each other), rules (simple action-reaction patterns that define what the system should do when a specific event is happening), timelines (chronological sequence of actions), pipeline (nodes correspond to services or activities are connected with directed graphs representing events; an example is presented in Section 6.4), jigsaw (only pieces with matching interfaces can be puzzled together, cf. [TMDi17]), and cards (best known from the HyperCard system by Apple). Programming styles comprise tangibles (real-world objects with sensor- or camera-based detection), spreadsheet (prevalent in modern office applications), natural language (e.g., smart speakers), mashup (cf. user interface mashups on the Web), trigger-action programming (connecting events with an appropriate action), mockup (a user interface is drawn that can later be superimposed with functionality), and programming-by-example (cf. recording configurable macros).

Despite the availability of numerous prototypes that work with these metaphors and programming styles, studies have shown that users often find it difficult to understand them. That is why our prototypes follow a community-based approach, where we rely on end-user and developer collaboration and their mutual engagement.

4.1.3 App Prototyping

After discussing metaphors and programming styles, we now turn towards systems that allow to create complete applications. Most of them target mobile Web applications, but there are also solutions that are able to generate fully native mobile applications that can be uploaded to the app stores of Android and iOS.

The most obvious manifestation of app creation platforms are content management systems for the Web. They can serve static textual and multimedia content, but many can be extended with functionalities like contact forms, event calendars and forums. The most popular representative is WordPress [NDMN14]. A commercial hosted Web-based mobile application creator is *Appery*[1]. It can create apps for Android and iOS; the underlying platform is Apache Cordova [Apac19]. *Buildup*[2] is a similarly cloud-based commercial platform that comes with drag-and-drop functionality that outputs source code for XCode and Android. The *IBM Bluemix Mobile dashboard*[3] is a framework for creating mobile apps that focuses on functionalities like database access, chatbots and other artificial intelligence functionalities provided by microservices in the IBM cloud. The *Google App Maker*[4] also targets business apps and works with drag-and-drop and declarative data modeling.

Interestingly, in this segment of application creation systems, there are particularly many providers of commercial solutions, many of them startups. We suspect that this is due to the fact that developing apps to market products is worthwhile but expensive for companies. Therefore, they like to use solutions that facilitate the process; after all, the required functionalities are provided out-of-the-box. On the other hand, these platforms are not able to target specific community needs, therefore

[1] https://appery.io
[2] https://www.buildup.io
[3] https://cloud.ibm.com/developer/mobile/dashboard
[4] https://developers.google.com/appmaker

the innovation factor is often very low. Also, vendor lock-in is a concern for these proprietary platforms. Once having invested in training for a specific tool to gain expertise, the hurdle is high to switch to another provider. This particularly applies to the solutions that generate code for iOS or Android.

4.1.4 Service API Driven Development

After presenting general programming paradigms for non-developers, we now take a look at approaches that specifically work with Web services on the backend and create UIs for them.

Maximilien et al. introduced *Swashup*, a domain-specific language for Web API and service mashups [MWDT07]. It leverages service descriptions like WSDL but focuses on the service composition. Similarly, He and Yen [HeYe07] and the *ServFace Builder* [NFH*10] rely on WSDL files. The ServFace approach generates user interfaces, but is driven by annotations written by the service developer. Vaziri et al. generate chatbots out of Swagger Web service documentation to interact with APIs in natural language [VMS*17]. Their system is able to improve the specification by letting users interact with the chatbot. The authors report that many REST APIs feature complex JSON structures, while their chatbot is designed for scalar input such as strings and dates. Swagger is also used by La Torre et al. [LMC*16] whose solution suggests the user APIs based on contextual properties. For each service, an Android user interface is generated; the goal is to provide uniform interfaces for a wide variety of context-dependent services.

There are several approaches that connect Internet of Things (IoT) device APIs to the Web. *Simurgh* uses the RAML API specification language to discover and integrate Internet of Things (IoT) devices [KDBu15]. Their functionalities are then combined by end users. Software products in the area of multi-vendor aggregation of IoT devices are for example *Node-RED* [JSFo18] and *IFTTT* [IFTT18]. However, while offering powerful features and a great user experience, these tools are neither multi-user capable for synchronous collaboration nor based on a vendor-independent, standardized notation.

4.2 Technical Foundation

In the following, we present the technological background needed for the conceptual design and implementation of our prototypes concerning community-driven modeling. As we will see in Chapter 5, these technologies are also the foundation of the API and app-based ecosystem on the Web. Because the REST architectural style is itself not a standard, often, services do not fully adhere to general guidelines [RSKl12]. There are a number of efforts to standardize service interactions. They help developers that use Web services to correctly target the functionality. In our implementation shown in Section 4.4, we argue, that they can also be helpful for automatically deriving user interfaces accessing the services. The *Web Service Description Language* (WSDL) 2.0 is an XML-based W3C recommendation standardized in 2007 [CJAS06]. Another XML-based API specification language is the Web Application Description Language (WADL) [Hadl09]. The *RESTful API Modeling Language* (RAML) [Mule17] and *API Blueprint* [Apia17] are description languages for REST APIs. While the former is YAML-based, the latter uses a Markdown-like syntax. For both

languages, support in terms of tool support for tests and development is available. In the following, we show two further representatives of API descriptions. The first is *OpenAPI* that comes with a high number of support tools. Then, the recent *GraphQL* format is discussed in detail. Finally, the *Interaction Flow Modeling Language* (IFML) is discussed. For all three cases we present a formalization. The goal is to be able to formally specify possible transformation approaches from and to these formats.

4.2.1 OpenAPI

The OpenAPI Specification is a description format for RESTful APIs on the Web. It is both human- and machine understandable, which leads to a wide range of use cases. On the one hand, developers can quickly see how to address a service accurately without having to read detailed textual documentation. On the other hand, the standard may also be evaluated via a machine, for example to develop program libraries. These can then be used for the automatic verification and conformance checks of queries. The specification document can be written in the text-based formats JSON or YAML. As their structure and content are identical, both are convertible to each other.

The listing in Appendix A shows the example of an address book service. The service allows to retrieve the full list of contacts, and to query or delete particular contacts. For reasons of simplicity, we exclude an API for adding new contacts. After the OpenAPI version identifier, the document starts with the `servers` block that contains pointers to server addresses the API is running on. The subsequent `info` header defines the basic information of the API it is describing, such as the version, the name, the license and the authors. Then, the `paths` of the REST API are listed. Each path item object describes the method that is possible, together with possible request and response parameters, as well as response codes. For describing data types, the items refer to the components section of the OpenAPI documentation. Among other definitions, it contains JSON schema descriptions of parameters used to validate input or output data. Examples that are compatible with the respective schema can be provided in the document. A schema object defines a type that is an extended subset of JSON schema [WrLu16, PRSU16]. This schema itself uses the JSON format itself; several versions have already been published as IETF Drafts. A schema object can define arrays, objects or references to other schema.

The predecessor of OpenAPI was called *Swagger*. While the specification was renamed to OpenAPI, the name Swagger now stands for a whole framework that revolves around automated tools built around the specification. *Swagger-UI* is an online tool for creating demo pages with try-out functionality of an API, where requests can be created with the help of simple text boxes and buttons. Figure 4.1 shows a screenshot of a generated user interface to the OpenAPI specification shown in Listing A. As can be seen, the user interface is very technical and therefore suitable for developers who want to explore the capabilities of the APIs. It includes dropdown boxes of HTTP content types and appropriate command line instructions to execute a request. In addition, there are code generators for obtaining code libraries for a wide variety of programming languages. They create methods, parameter objects and validators. The OpenAPI specification is language-agnostic and allows to be extended by vendor-specific extensions. With the newest version of the specification, OpenAPI 3.0.0 introduces link definitions. The purpose is to to provide known relationships between responses and other operations [Open18]. That way, the API consumer can be informed, that

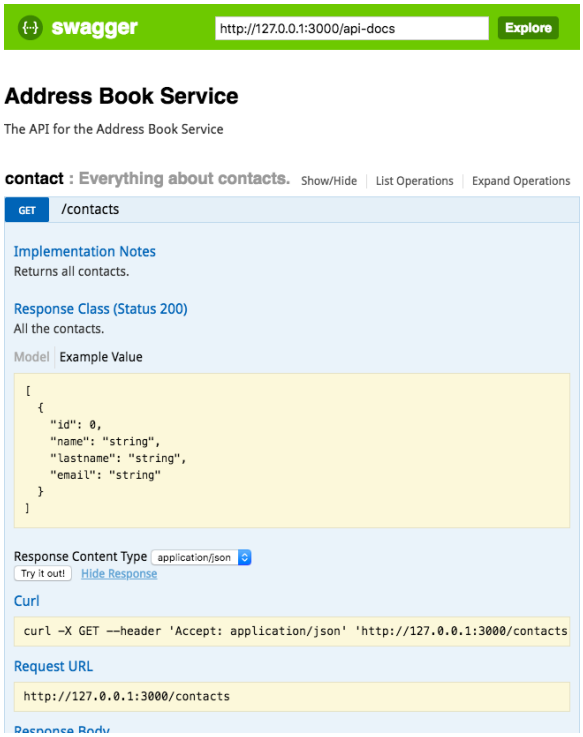


Figure 4.1: Address Book Example in Swagger UI

at another endpoint, further information about an object is available. For instance, in the example above, the `/contacts` endpoint could define a link to the `/contacts/contactId` endpoint for a particular response. In that case, the response object would mark the id of the contact object as the key, using runtime expressions. Links are not dynamic and do not depend on the transferred data.

There are a number of approaches to automate the generation of service descriptions. *SpyREST* intercepts and analyzes example service calls to create the documentation [SAMa15b, SAMa17]. It uses a proprietary documentation format instead of OpenAPI. *Swagger Inspector*[5] allows users to perform requests manually; it then creates the according documentation from the information gathered. In a similar way, *Autoswagger.lua*[6] uses example requests to derive the documentation. According to an evaluation by Suter et al. the latter performs well on some APIs but does often not produce usable results [SuWi15]. Instead, the authors propose to infer "Web API Descriptions", which is a proprietary specification that can be translated to OpenAPI. A tool introduced by Ed-doubi et al. [ECCa17], as well as *APIDiscoverer*[7] are further service description generators. Finally, *APIMATIC*[8] allows converting between different description formats, including API Blueprint, WADL, WSDL, RAML and Swagger/OpenAPI. As these documentation standards are largely interoperable, we are confident that our conceptual work is transferable to any API descrip-

[5] https://swagger.io/tools/swagger-inspector
[6] https://autoswagger.lua
[7] https://github.com/opendata-for-all/APIDiscoverer
[8] https://apimatic.io

tion specification.

A metamodel of OpenAPI is introduced in [ECCa17]. The authors' goal is to automatically discover OpenAPI specifications through example requests. Thus the paper only describes the structure of the schema. To our knowledge, a formalization of OpenAPI does not yet exist. Therefore, in the following we discuss a formal model for OpenAPI, initially presented by Kus in a bachelor thesis under our supervision [Kus19]. We base our model on the official current OpenAPI specification version 3.0.2 [Open18]. Let us first define the set of HTTP methods supported by OpenAPI:

$$\begin{aligned} Methods_{\mathcal{O}} \quad := \quad & \{GET, PUT, POST, DELETE, \\ & OPTIONS, HEAD, PATCH, TRACE\} \end{aligned}$$

We then consider four infinite countable disjoint sets: $Paths_{\mathcal{O}}$, $Arguments_{\mathcal{O}}$, $Types_{\mathcal{O}}$ and $Fields_{\mathcal{O}}$. Furthermore, we consider a subset $Scalars_{\mathcal{O}} \subseteq Types_{\mathcal{O}}$ of scalar types, a set $Vals_{\mathcal{O}}$ of scalar values and a function $values_{\mathcal{O}} : Scalars_{\mathcal{O}} \to 2^{Vals_{\mathcal{O}}}$ that assigns a set of values to every scalar type.

We define an OpenAPI 3.0 schema over finite subsets of the above sets. To this end, we define four finite sets as follows:

$$\begin{aligned} P_{\mathcal{O}} \quad & \subset \quad Paths_{\mathcal{O}} \\ A_{\mathcal{O}} \quad & \subset \quad Arguments_{\mathcal{O}} \\ T_{\mathcal{O}} \quad & \subset \quad Types_{\mathcal{O}} \\ F_{\mathcal{O}} \quad & \subset \quad Fields_{\mathcal{O}} \end{aligned}$$

We assume $T_{\mathcal{O}}$ to be a disjoint union of $O_{T_{\mathcal{O}}}$ (schema object types), $A_{T_{\mathcal{O}}}$ (array types), $C_{T_{\mathcal{O}}}$ (compound types) and $Scalars_{\mathcal{O}}$. The set of array types $A_{T_{\mathcal{O}}}$ contains only items that are arrays whose items are of a type $t \in T_{\mathcal{O}}$. We denote such an array type as $[t]$. This implies that types that are implicitly declared in the item definition of an array are also in T. The set $C_{T_{\mathcal{O}}}$ consists of types that are built out of other types using the `anyOf`, `oneOf` or `allOf` keyword. Therefore, it can be partitioned into the set of `anyOf` types $C_{T_{\mathcal{O}}}^{\geq 1}$, the set of `oneOf` types $C_{T_{\mathcal{O}}}^{=1}$ and the set of `allOf` types $C_{T_{\mathcal{O}}}^{=n}$. We assume that enum types are part of the $Scalars_{\mathcal{O}}$ set. We define the set $Op \subseteq P_{\mathcal{O}} \times Methods_{\mathcal{O}}$ to contain all operations that are defined over paths in $P_{\mathcal{O}}$ in the OpenAPI documentation. The response set $R_{\mathcal{O}}$ is now defined as the set of valid combinations of an operation and a HTTP response code. It contains items of the form (o, c) for an operation $o \in Op$ and a status code c that is defined for operation o in the OpenAPI documentation. An OpenAPI 3.0 schema $\mathcal{S}_{\mathcal{O}}$ over $(P_{\mathcal{O}}, A_{\mathcal{O}}, R_{\mathcal{O}}, T_{\mathcal{O}}, F_{\mathcal{O}})$ is now defined by the following mappings.

$params_{\mathcal{S}_{\mathcal{O}}} : P_{\mathcal{O}} \to 2^{A_{\mathcal{O}}}$ assigns a set of arguments to every path. Those are the arguments for the HTTP GET request defined in the OpenAPI schema. Note, that this definition does not consider optional arguments. This is done to keep the definition simple.

$type_{\mathcal{S}_{\mathcal{O}}} : (R_{\mathcal{O}} \cup A_{\mathcal{O}} \cup F_{\mathcal{O}}) \to (T_{\mathcal{O}} \cup \{\epsilon\})$ assigns a type to every response for a path as well as to every argument and to every field. The OpenAPI specification allows responses to be defined without specifying their data type. Those responses are mapped to ϵ.

$fields_{\mathcal{S}_{\mathcal{O}}} : O_{T_{\mathcal{O}}} \to 2^{F_{\mathcal{O}}}$ assigns each object type the fields that this object has.

$links_{\mathcal{S}_{\mathcal{O}}} : R_{\mathcal{O}} \to 2^{P_{\mathcal{O}} \times 2^{A_{\mathcal{O}}}}$ assigns a set of paths with sets of arguments to every response. The set of arguments describes which arguments to the target path are already provided when following the link definition.

$components_{\mathcal{S}_\mathcal{O}} : C_{T_\mathcal{O}} \to (2^{T_\mathcal{O}} \setminus \emptyset)$ assigns each compound type to the non-empty set of types it consists of.

Request bodies are modeled as parameters in this formalization. To this end, there is a distinguished set of parameters $RequestBodies_{\mathcal{S}_\mathcal{O}} \subseteq A_\mathcal{O}$ that described the request bodies of operations in the OpenAPI document. As every operation can have at most one request body defined, it holds that $|params_{\mathcal{S}_\mathcal{O}}(o) \cap RequestBodies_{\mathcal{S}_\mathcal{O}}| \leq 1$ for each $o \in Op$.

Note that the argument translation that is performed for a link is not explicitly modeled. If $(p, a) \in links_{\mathcal{S}_\mathcal{O}}(r)$ for a $r \in R_\mathcal{O}$, this implies that $a \subseteq params_{\mathcal{S}_\mathcal{O}}(p)$. The set a is the set of arguments to the link p that are already provided when following the link, either from the response body or from the parameters to the query. Therefore, only the existence of arguments is modeled but not their source.

4.2.2 GraphQL

Over the last years, REST-based APIs have become a prevalent way of accessing and manipulating data through Web services. The resource-oriented nature of the endpoints, however, often leads to many consecutive HTTP calls from clients in order to collect and show the necessary information of an overview page. Recently, new approaches for client-server interactions have emerged that target different aspects like latency or structure of calls. Examples are Google's gRPC, Netflix's Falcor. The more widely recognized new protocol is GraphQL, an open source query and manipulation specification. Originally developed internally within Facebook in 2012, GraphQL was published in 2015 and open sourced in 2018, when it was handed over to be maintained by the Linux Foundation. Since then, it has been adopted by many big players in the industry; GitHub, for example, built its latest API version 4 with GraphQL. In the spirit of REST architectures which are based on HTTP, it offers an SQL-like structure that allows it to define the exact structure of the queried data. The conceptual proximity to SQL makes it possible to use GraphQL for the external data model, thus massively simplifying the developer workflow [Brya17]. GraphQL stands for a schema language for defining types as well as a query language for specifying queries. The queries thereby have to conform to a schema. Although responses are usually delivered in the JSON format, there are no strict requirements in this regard, except that the data model structure returned must reflect the one queried. The schema and query language have been influenced by JSON. The query language can be parameterized, which offers the greatest advantage over REST, since no complete data structures of a resource have to be transferred. Instead, for each request and entity, the client may specify different subsets and networks of attributes. Specifying the exact query results allows to conserve bandwidth, which is especially beneficial for mobile deployment scenarios. Compared to REST, on the one hand, the high granularity prevents *overfetching*, as only those attributes of an object are transferred that the client needs. On the other hand, it avoids *underfetching*, i.e. one request can span multiple (sub-)resources. A recent study asserts a reduction of 99% in the query result size in the median when comparing GraphQL APIs with their REST version [BMVa19]. Queries are usually sent as HTTP GET or POST requests to a single HTTP endpoint [Grap19]. There are already multiple libraries available for a big variety of languages that provide developer support for accessing GraphQL interfaces. For JavaScript, the most popular is Apollo. It exists for both client and server.

In the following, we introduce the GraphQL formalization proposed in [HaPe18]. We start with three infinite countable sets: $Fields_{\mathcal{G}}$, $Arguments_{\mathcal{G}}$ and $Types_{\mathcal{G}}$. Those sets contain all possible fields, arguments and types, and are assumed to be disjoint. Furthermore, we consider the set of scalar types $Scalars_{\mathcal{G}} \subset Types_{\mathcal{G}}$ which is finite. $Vals_{\mathcal{G}}$ is the set of scalar values and $values_{\mathcal{G}} : Scalars \rightarrow 2^{Vals}$ assigns each scalar type a set of possible values.

The formalization of a GraphQL schema is now defined over finite subsets of those sets. To this end, we define three finite sets as follows:

$$\begin{aligned} F_{\mathcal{G}} &\subset Fields_{\mathcal{G}} \\ A_{\mathcal{G}} &\subset Arguments_{\mathcal{G}} \\ T_{\mathcal{G}} &\subset Types_{\mathcal{G}} \end{aligned}$$

The set $T_{\mathcal{G}}$ is assumed to be the disjoint union of the set of object types $O_{T_{\mathcal{G}}}$, the set of interface types $I_{T_{\mathcal{G}}}$, the set of union types $U_{T_{\mathcal{G}}}$ and the set $Scalars_{\mathcal{G}}$. Furthermore, let $L_{T_{\mathcal{G}}}$ be the set of list types $\{[t] \mid t \in T_{\mathcal{G}}\}$ constructed from types in $T_{\mathcal{G}}$. Note that although enum types are not scalar according to the GraphQL specification, they are very similar to those scalar types as they also represent leaves in a GraphQL type system [Face18]. Due to this similarity, we consider enum types to be part of the $Scalars_{\mathcal{G}}$ set. A GraphQL schema $\mathcal{S}_{\mathcal{G}}$ over $(F_{\mathcal{G}}, A_{\mathcal{G}}, T_{\mathcal{G}})$ is defined by the following mappings:

$fields_{\mathcal{S}_{\mathcal{G}}} : (O_{T_{\mathcal{G}}} \cup I_{T_{\mathcal{G}}}) \rightarrow 2^F_{\mathcal{G}}$ assigns a set of fields to each object type and to each interface type.

$args_{\mathcal{S}_{\mathcal{G}}} : F_{\mathcal{G}} \rightarrow 2^{A_{\mathcal{G}}}$ assigns a set of arguments to every field.

$type_{\mathcal{S}_{\mathcal{G}}} : (F_{\mathcal{G}} \cup A_{\mathcal{G}}) \rightarrow (T_{\mathcal{G}} \cup L_{T_{\mathcal{G}}})$ assigns a type to every field and to every argument.

$union_{\mathcal{S}_{\mathcal{G}}} : U_{T_{\mathcal{G}}} \rightarrow (2^{O_{T_{\mathcal{G}}}} \setminus \{\emptyset\})$ assigns each union type the set of object types it consists of. This set is always non-empty.

$implementation_{\mathcal{S}_{\mathcal{G}}} : I_{T_{\mathcal{G}}} \rightarrow 2^{O_{T_{\mathcal{G}}}}$ assigns each interface all the object types that implement it.

There is also a distinguished object type $root_{\mathcal{S}} \in O_T$ called the (query) root type.

Note that while the original formalization in [HaPe18] requires $type_{\mathcal{S}}$ to assign only scalar types to arguments, we do not require this condition. The GraphQL specification requires arguments to be of an input type [Face18]. While our formalization does not adhere to this requirement, we can duplicate our set of types T into a new set of input types and use only these types for arguments. This would make our formalization adhere to the specification. However, we want to refrain from modeling the input types directly to keep the model simple. For the same reason, the notion of non-null types is also not considered in this formalization.

4.2.3 Interaction Flow Modeling Language

The Interaction Flow Modeling Language (IFML) is a visual domain-specific modeling language for the visual design of abstract user interactions and data flows within user interfaces [BrFr14]. Developed in 2012 and 2013, it is standardized by the Object Management Group OMG), the same

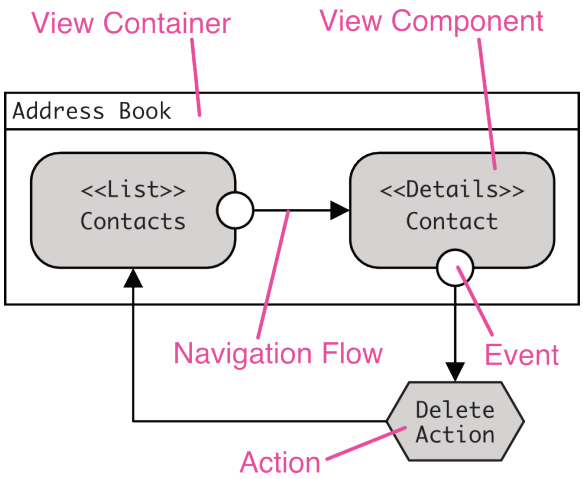


Figure 4.2: Address Book IFML Model

standardization body that is also governing the UML standard prevalent in software engineering. An IFML model is a platform-independent representation of a graphical user interface on devices such as mobiles, laptops or wearables. Figure 4.2 showcases the IFML model of the very simple address book application. It displays a list of contacts; upon selecting a particular contact, the details are shown in a view right next to the list. Below the details, a button allows deleting the selected contact. The concepts behind the model elements are explained in the following.

An IFML model has one or more possibly nested View Container at its root (“Address Book”). A View Container can represent for instance an application window of a desktop application, or the main view of a Web page. It can contain View Components standing for certain user interface elements like a list or a table (“Contacts List” and “Contact Details”). These model elements can be associated with Events (“selected” and “delete”) or Actions (upon selecting the “delete” button). An Event can be generated by the user or system, for instance when clicking a button. It can trigger an Action, for example the transition of one view to another. State transitions and data flows are modeled as edges with *Navigation* or *Data Flows*. Navigation Flows are happening upon user interaction; it then represents the transition from event to another view. Data Flows represent any data passing from one element to another, for example, the identifier of a selected menu item.

While the standard is platform-independent, it can be used to generate concrete user interfaces. The standard document contains multiple example mappings to user interface description languages, for instance to the Windows Presentation Foundation, Java Swing, and HTML. For example, a View Container may be translated to an HTML `<div>` element, and the nested View Components to respective child elements. The official IFML website lists several commercial and open source model editor implementations. Particularly noteworthy is *IFMLEdit.org*, a Web-based modeling tool able to generate Web clients out of IFML models [BCFr17]. A formal model of IFML is provided in [BCFr18].

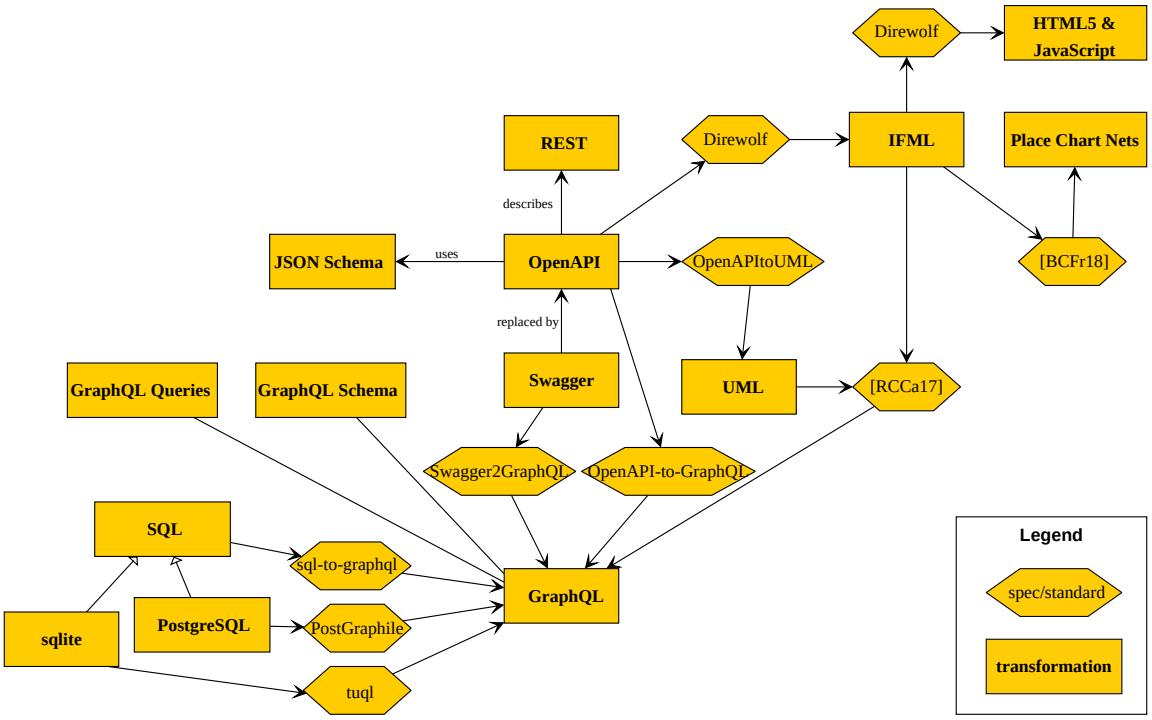


Figure 4.3: API Transformation Approaches

4.3 Model Transformations

Figure 4.3 shows transformation approaches between different database query language and user interface description formats commonly used for API automation. In the square fields, concrete technologies are named, while the diamond fields contain transformation approaches. The picture can be read from the bottom left to the top right. First, database query languages are shown, like SQL, PostgreSQL and sqlite, a database specification prominently used in the Android operating system. GraphQL offers significant advantages over REST-based approaches as discussed in Section 4.2.2. Therefore, possible migration paths from REST to GraphQL are of particular interest. Two solutions are provided in the figure, Swagger2GraphQL as well as OpenAPI-to-GraphQL. They represent two possible ways to accomplish the migration from one API type to the other. The first is to use a model-based approach. The second possibility is to create a GraphQL wrapper around the existing interface. An example of a model-driven approach is presented by Rodriguez-Echeverria et al. where UML and IFML are used to model the API [RCCa17]. For the transformation, first an UML model has to be created from the REST specification beforehand. To this end, there are various tools like OpenAPItoUML [ECCa18], *Pikturr*[9] and *Swagger to UML*[10]. A wrapper offers a GraphQL endpoint and performs the transformation on-the-fly as soon as a request is received. Depending on the query and how much it is nested, a single GraphQL query may be translated to one or several REST requests. In the case of multiple requests, the wrapper then has

[9]https://github.com/nrekretep/pikturr
[10]https://github.com/nlohmann/swagger_to_uml

Table 4.1: Distribution of APIs.Guru API Specifications (adapted from [Kus19])

	Swagger/OpenAPI				Other	Unknown	Total
	1.1	1.2	2.0	3.0			
Absolute	1	44	1106	52	185	190	1578
Relative	0.1%	2.8%	70.1%	3.3%	11.7%	12.0%	100%

to construct the response object accordingly. There are various tools that can generate a GraphQL wrapper from a machine-readable REST API specification. *Swagger2GraphQL*[11] cite takes Swagger documentation as a starting point. OpenAPI-to-GraphQL also uses Swagger 2.0 or OpenAPI 3.0 as input. A detailed analysis and comparison is performed by Wittern et al. [WCLa18]. In contrast to Swagger2GraphQL, it also leverages newly introduced features of the most recent version of the specification, resulting in a more expressive and reusable schema. It generates fewer issues and produces more intuitive names. In the following, the OpenAPI-to-GraphQL wrapper generator [WCLa18] will thus be our main focus of interest.

4.3.1 Prerequisites

To get an overview of real-world APIs, we first conducted an empirical analysis of service documentation specifications. We found the public Swagger/OpenAPI directory APIs.guru to be the most comprehensive list of public OpenAPI documentation on the Web[12]. It is an open source project with an active community and frequent updates maintained on GitHub. Every entry in the directory links to the source documentation, which can be of any format that is convertible to Swagger 2.0. APIs.guru itself provides a public API to retrieve the collection of specifications. The returned data includes information about the respective APIs and a link to the source documentation, as well as to a converted Swagger 2.0 version hosted by the directory. At the time of writing, the directory contains 1574 documentations; four of the entries each provide two specifications, resulting in a total of 1578 specification documents. To perform our analysis, we first downloaded all 1578 documents. Table 4.1 gives an overview of the original specification standards used. In total, 1203 APIs already use the Swagger 1.1, 1.2, 2.0 or OpenAPI 3.0 standard. A number of 185 use a different standard like RAML, while 190 specifications were added that use a non-standard documentation; thus the latter were added manually to the directory. Only 52 specifications are in OpenAPI 3.0 format. We thus needed to carry out a scripted conversion with the swagger2openapi tool[13]. During the conversion, we found that three specifications contained errors, thus, we were not able to automatically convert them and disregarded them. To have a common starting point for all the APIs, we then automatically converted all Swagger 2.0 documentations to the OpenAPI 3.0 specification. In the 52 cases mentioned above, this step was skipped.

We then looked at the existence of link definitions. As discussed earlier, the links introduced in OpenAPI 3.0 provide a traversal mechanism for service interfaces [Open18]. The large part of

[11] https://github.com/yarax/swagger-to-graphql

[12] https://apis.guru/openapi-directory/

[13] https://github.com/Mermade/oas-kit/blob/master/packages/swagger2openapi

Algorithm 1 First Step of the Link Generator (adapted from [Kus19])

Input
ops List of HTTP GET operations
Output
$pairs$ List of potential link pairs

function FINDPOTENTIALLINKPAIRS(ops)
  for all $o \in ops$ **do** ▷ Remove operations without 'success' response
    if $responseCodes(o) \cap \{200, \ldots, 299\} = \emptyset$ **then**
      $ops \leftarrow ops \setminus \{o\}$
    end if
  end for
  $pairs \leftarrow \emptyset$
  for all $o_1 \in ops$ **do**
    for all $o_2 \in ops \setminus \{o_1\}$ **do**
      $p_1 \leftarrow path(o_1)$
      if $\neg(p_1$ ends with '/'$)$ **then**
        $p_1 \leftarrow p_1 +$ '/'
      end if
      if $path(o_2)$ starts with p_1 **then**
        $pairs \leftarrow pairs \cup \{(o_1, o_2)\}$
      end if
    end for
  end for
  return $pairs$
end function

the crawled documentations do not include link definitions as they were not part of the previous version of the standard. Out of the 52 original OpenAPI 3.0 specifications, only three contained link definitions. Thus, the specification released in mid-2017 did not yet reach mainstream adoption among public APIs. We considered this an obstacle to our effort to automatically generate GraphQL schemas. Therefore, we came up with a tool called *Link Generator* that scans OpenAPI 3.0 documents and automatically adds link definitions where applicable, and then outputs an extended OpenAPI 3.0 document. We use it as a pre-processing step before generating the GraphQL wrapper. It relies on heuristics to determine the best fits to introduce links in a given specification. They are introduced in the following.

First, we try to find pairs of paths that are related to each other. It is an established best practice to design REST interfaces in a way that use paths to organize objects in a hierarchical relationship [Mass12]. We leverage that to use in our first algorithm presented in Algorithm 1. Second, we filter the potential links based on their request parameters. Out of the available HTTP methods, GET fetches the state of a resource [Mass12]. Thus we only consider GET requests and parameters in our method. It is presented in Algorithm 2. For every link candidate, we search for matching parameters between link source and link. If a matching parameter is found, we add it to the parameter app. After all the parameters have been considered, we check if all the target parameters have been paired; if yes, the pair is added to the list of links, otherwise the pair is discarded. An exception are 'cookie' parameters, as they are not specifically related to a request. Instead, we assume that cookies are automatically sent by the client in subsequent calls. Third, and lastly, the Link Gener-

Algorithm 2 Second Step of the Link Generator (adapted from [Kus19])

Input
pairs List of potential link pairs from first stage
Output
links List of filtered link pairs with parameter mapping

function FILTERLINKPAIRS($pairs$)
  $links \leftarrow \emptyset$
  for all $(o_1, o_2) \in pairs$ **do** ▷ o_2 is the potential link target
    $paramMap \leftarrow \emptyset$
    for all $a_2 \in \{a \in params(o_2) \mid location(a) \neq cookie\}$ **do**
      for all $a_1 \in \{a \in params(o_1) \mid location(a) \neq cookie\}$ **do**
        if $name(a_1) = name(a_2) \wedge schema(a_1) = schema(a_2)$ **then**
          $paramMap \leftarrow paramMap \cup \{a_1 \rightarrow a_2\}$
        end if
      end for
    end for
    if $requiredParams(o_2) \subseteq image(paramMap)$ **then**
      $links \leftarrow links \cup \{(o_1, o_2, paramMap)\}$
    end if
  end for
  return $links$
end function

ator adds the links to the OpenAPI document from source to target. If there are multiple success response codes, a link is added to each.

We performed a runtime analysis of the three steps of the generator algorithm. For the first step, we assume n to be the length of os. As it first iterates over all operations and then over all operation pairs excluding pairs of the same operation, the runtime is in $\mathcal{O}(n+n\times(n-1))$. At the second step, we also look at the number of operation parameters. Let P be the maximum number of parameters defined for any operation in cs and let n be the length of cs. Then, the runtime of the second stage algorithm is in $\mathcal{O}(n \times P^2)$. In the last step, we only consider the output of the second stage. This only requires a single pass over the input, leading to a linear runtime. In conclusion, the runtime of the link generator is in $\mathcal{O}(n^2 + n \times P^2)$ and therefore polynomial.

4.3.2 Schema Translation

After the challenge with adding missing links, we are now ready to formally specify the schema translation from OpenAPI 3.0 to GraphQL. We use it when generating a GraphQL wrapper. We formalize it as a mapping $st : \mathcal{O} \rightarrow \mathcal{G}$ for the set of all OpenAPI schemas $\mathcal{O}$ and the set of all GraphQL schemas $\mathcal{G}$. This mapping can not be a bijection as information is lost when translating from OpenAPI to GraphQL; the details are discussed in the next section.

Assuming we have an OpenAPI schema $\mathcal{S} \in \mathcal{O}$ over (P, A, R, T, F). Recall, that according to the definition, $T = \dot{\bigcup}\{O_{T_\mathcal{O}}, C_T, A_T, Scalars_T\}$. We assume, that the sets of anyOf and oneOf types are empty: $C_T^{=1} = C_T^{\geq 1} = \emptyset$. If the OpenAPI document contains such a type, OpenAPI-

to-GraphQL fails in this step. In a pre-processing step OpenAPI-to-GraphQL uses the tool *json-schema-merge-allof*[14] to flatten the types in $C_T^{=n}$ by merging the individual schemas of the allOf types in the set $C_T^{=n}$ into one single schema for each of the types. This converts all those types into either object, array or scalar types. We assume that this process was already run on $\mathcal{S}$ and that therefore $C_T^{=n} = \emptyset$. A corresponding GraphQL schema is now constructed as follows.

Let $T'_\mathcal{G} := \dot{\bigcup}\{O_{T'}, I_{T'}, U_{T'}, Scalars_{T'}\}$ be the new set of GraphQL types given as follows:

$$\begin{aligned} O_{T'} &:= O_T \,\dot{\cup}\, \{Query\} \\ I_{T'} &:= \emptyset \\ U_{T'} &:= \emptyset \\ Scalars_{T'} &:= Scalars_T \end{aligned}$$

where $Query$ is the distinguished root type $root_\mathcal{S}$ in accordance to the GraphQL formalization. Note that due to our assumption, $C_T = \emptyset$ holds and furthermore $A_T \subseteq L_{T'}$. Therefore, all types get carried over into the new GraphQL schema.

We set $A' := A$. We define the set of fields as follows:

$$F' := \dot{\bigcup}\{F, P', L'\}$$

with $P' := \{p \in P \mid \exists 200 \leq c < 300 : (p, c) \in R\}$ and $L' := \bigcup_{r \in R} link_\mathcal{S}(r)$. Hereby, P' contains only paths that have a successful response defined. A successful response is characterized by a HTTP response code from 200 to 299. L' contains all link targets defined in the documentation. Based on this, we define the GraphQL schema $\mathcal{S}' \in \mathcal{G}$ over (F', A', T'). For $t \in (O_{T'} \cup I_{T'})$:

$$\mathit{fields}_{\mathcal{S}'}(t) = \begin{cases} fields_\mathcal{S}(t), & \text{if } t \in O_T \\ P, & \text{if } t = Query \end{cases}$$

which is well-defined because $I_{T'} = \emptyset$. For $f \in F' = F \cup P' \cup L'$, $args_{\mathcal{S}'}(f)$ is defined as follows:

$$args_{\mathcal{S}'}(f) = \begin{cases} \emptyset, & \text{if } f \in F \\ params_\mathcal{S}(f), & \text{if } f \in P' \\ params_\mathcal{S}(p) \setminus a, & \text{if } f = (p, a) \in L' \end{cases}$$

Note, that for link fields only the arguments that are not already given by the link become arguments for the field.

To define the $type_{\mathcal{S}'}$ mapping, we need a helper function:

$$succ(p) = (p, c), \text{for } c = \min\{200 \leq c < 300 \mid (p, c) \in R\}$$

for every $p \in P'$. Note that this function is well-defined as there exists a $200 \leq c < 300$ with $(p, c) \in R$ for every $p \in P'$ by the definition of P'. This function calculates the lowest success

[14] https://github.com/mokkabonna/json-schema-merge-allof

status code for a path. Using this, we now can define the $type_{\mathcal{S}'}$ mapping. For $f \in F' \cup A' = F \cup P' \cup L' \cup A$:

$$type_{\mathcal{S}'}(f) = \begin{cases} type_{\mathcal{S}}(f), & \text{if } f \in F \\ type_{\mathcal{S}}(f,c), & \text{if } f \in P' \text{ and } succ(f) = (f,c) \text{ and } type_{\mathcal{S}}(f,c) \neq \epsilon \\ string, & \text{if } f \in P' \text{ and } succ(f) = (f,c) \text{ and } type_{\mathcal{S}}(f,c) = \epsilon \\ type_{\mathcal{S}}(f,c), & \text{if } f = (p,a) \in L' \text{ and } succ(p) = (p,c) \text{ and } type_{\mathcal{S}}(p,c) \neq \epsilon \\ string, & \text{if } f = (p,a) \in L' \text{ and } succ(p) = (p,c) \text{ and } type_{\mathcal{S}}(p,c) = \epsilon \\ type_{\mathcal{S}}(f), & \text{if } f \in A \end{cases}$$

OpenAPI-to-GraphQL automatically maps responses with no type specified to the string type. The reason is that while a REST API may offer an interface that does not return any data, this is prohibited in GraphQL. To get around this limitation, these responses are still assigned a data type to make them available in the GraphQL API. As we do not have any interface or union types, the $implementation_{\mathcal{S}'}$ and $union_{\mathcal{S}'}$ mappings are empty.

4.3.3 Discussion

The Web is currently undergoing a technological transition from resource-oriented to query-based service interfaces. While the full impact is not yet known, the implications on the infrastructure are considerable. More expressive GraphQL queries can quickly lead to a server overload, as multiple nested resources can be queried at once. To help system designers, we thus need a good understanding of current system behavior. While the open source OpenAPI-to-GraphQL is able to generate GraphQL wrappers, there was no formal model behind this transformation. Additionally, it works best on OpenAPI 3.0 documents with present link definitions. To this end, we presented the Link Generator that, based on some heuristics, adds this missing information. It is not able to fully translate documents, as on the schema part, information gets lost. To get an understanding of how severe this affects existing APIs, we analyzed the OpenAPI schemas we downloaded from APIs.guru in our initial empirical study. Table 4.2 shows the results. We can see, that around 33% of all real-world documents use this schema information. In particular, the properties *minimum*, *maximum*, *minLength* and *maxLength* and pattern are used very frequently; each of them in about every fifth document. In absolute terms, the *uniqueItems* property is used quite frequently in the APIs of Kubernetes and Microsoft Graph API (which despite its name is based on REST). In the Kubernetes API, the property is only used with schema objects of type boolean, integer or string, but not with array, which is the only permitted usage according to the JSON Schema draft [WrLu16]. It thus has no effect on our transformation.

Existing architectural elements like caching and logging servers are essential for the operation of the Web at its current scale, yet today, respective tools are heavily oriented towards REST-based queries. A transformation towards GraphQL opens up new possibilities for both caching and logging, therefore they are particularly interesting for the research on Web infrastructure. For instance, as a GraphQL query exactly specifies the fields that are requested, it is possible to log access at a much higher grade of detail. Such a property-level assessment is not possible for REST APIs as the query always returns the whole resource. For instance, a client may only need the *author* field of a *book* resource, which can be mapped in GraphQL, but not in REST.

Table 4.2: Usage of OpenAPI Schema Properties on APIs.Guru [Kus19]

Property	Documents		Total occurrences
	Count	Ratio	
multipleOf	1	0.1%	2
minimum	287	18.3%	2238
maximum	241	15.3%	1566
minLength	326	20.8%	4925
maxLength	327	20.8%	5399
pattern	329	20.9%	3706
minItems	110	7.0%	610
maxItems	111	7.1%	729
uniqueItems	12	0.8%	10433
minProperties	29	1.8%	46
maxProperties	39	2.5%	62
oneOf	7	0.4%	26
anyOf	2	0.1%	963
not	0	0.0%	0

With the presented formalisms of OpenAPI and GraphQL and the transformation, it is possible to model the aspects of the specification that are relevant to APIs. For instance, we can create a function, that for a given GraphQL query and a REST schema calculates how many HTTP requests to the REST API would be necessary to query the same data. This again can be used to evaluate latency benefits of GraphQL over REST, especially when considering non-atomic queries in REST. Additionally, we can identify the type of queries that generate the highest load on the underlying REST API.

4.4 Model-Driven Composition of APIs and UIs

In the related work section of this chapter, we described various approaches in the area of end user development. There are many tools available on different granularity levels. Some allow to wire together service functionalities through APIs with user interface elements, others offer full-fledged application prototypes that work cross-platform. While with the former, rich functionalities can be implemented, the latter are often very specialized and limited in terms of reusability; i.e., it is hard to export the app creation to further work in them in a proper development environment. Both approaches, however, are specifically targeting end users. Although there is a limited number of applications that work on a standardized modeling language that can also be understood by professional developers, they are restricted to a specific modeling notation. Hence, we aim for a tool, that supports various domain-specific modeling languages, and is also able to export development artifacts that can be reused in other modeling, design, or programming environments. Conceptually, we aim for an application composer that is working on a formal model of the app to be created. To

offer different community members specialized editors, the requirement is a database-view-based data structure [JJMy09]. This data structure forms the lowest common denominator of, for example, end users and developers. Concerning the user interface and interactions between certain user interface elements, we presented IFML. For service design, we highlighted OpenAPI. In this part, we therefore connect these two standards.

4.4.1 Conceptual Design

In the following, we present *Direwolf Model Academy*, a Web-based collaborative model editing framework. It is a generally applicable editor infrastructure for the creation of models with edges and nodes. The name is based on the object-oriented principle that properties of model elements are passed through in the inheritance hierarchy; in this sense, they "learn" from each other. It is able to host several modeling spaces in parallel, similar to a document structure on a file system. Thereby, each instance of the editor on every browser is connected via a shared data model. We call each distinct environment a *space*. This works analogous to *ROLE spaces*, that are containers of widgets that usually consist of a frontend and a service in the background [NKR*14]. A space can have multiple simultaneous users that each have a different view on the underlying data model. In the following, we discuss each part of the editor and explain the conceptual architecture behind the view-based model editor.

A *Modeler* is a graphical editor that displays the graph- and edge-based composition of a model. In their entirety, all modelers create a tree-based data structure, with the root model on top. The data structure is synchronized in near real-time; following a publish-subscribe paradigm, we provide events for adding, updating and removing nodes and edges. Modelers bind to updates of their underlying data structure. They may decide to omit certain nodes or edges in their graphical representation, based on their defined viewport. Once a change happens in the underlying data structure, an event notification is sent to the modeler which then updates the graphical representation of the model element. Similarly, a change of the model element, triggered by a drag-and-drop action, updates the properties of the underlying model.

Palette elements provide a list of possible nodes and edges. They may be specified at design-time, or dynamically generated at runtime for specific transformations. Conceptually, they may also be downloaded from a "model store", an online app store of model elements. In the user interface of Direwolf, they are located on the left. From the palette, model elements can be moved to the modeler via drag-and-drop.

Property Browsers are form-based representations of a selected model element's properties. It can also display settings derived from a view. Technically, the browser gets access to the shared Direwolf data store of the model's class. In the user interface, the properties are displayed in a table structure. Properties that are not marked as read-only can be directly edited in the table. Once changed, the underlying data structure and publish-subscribe architecture notifies the model element's visual instance in the modeler view, to change properties, e.g. the size. Similarly, if a model element is changed visually for instance through resizing, the appropriate property in the property browser's table is updated instantly.

Transformers use the same event API as modelers, however they transform the model either to a different metamodel instance, or to an application-specific representation. An exemplary transfor-

mation from model to HTML is shown in the case study presented in the Section 4.4.3. Technically, they present a subset of the model elements they are working on. Thus, they can be formalized as views on the underlying data structure. We chose the view-based approach, as it allows us to hide unnecessary detail for certain subcommunities. For instance, metadata relevant for developers do not need to be shown to end users.

4.4.2 Implementation

The Direwolf Model Academy framework and its derived modeling applications are implemented using the JavaScript programming language version ECMAScript 2015 [Ecma15]. The frontend is developed using HTML5 Web Components. A Web Component is a custom, reusable DOM element that can be used in place of native HTML elements like `<h1>` or `<div>`. Thus, parts of the application can be embedded in third-party Web applications leading to a cross-application cross-browser editing. Specifically, we use the Polymer[15] library that is maintained by Google. Polymer also comes with high-fidelity pre-designed template elements following the Material Design guidelines [Goog18]. They achieve high accessibility and a familiar user experience across devices. The graphical components of a model are implemented as *Scalable Vector Graphics* (SVG) elements. SVG is a widely supported image format for vector-based images. It is available on all modern Web browsers on all platforms, mobile and desktop. The prime reason for using SVG is that the resulting model can be directly exported from the browser for editing in a vector-based graphics program like *Inkscape* or *Microsoft Visio*. We use the SVG.js[16] open source library, as it provides plugins with various convenience functions such as calculating intersections of multiple SVG paths. Besides, most browsers support a raster graphic export in the PNG graphics format. Synchronization is done via the Yjs open source library [Jahn19]. The library implements a *commutative replicated data type* (CRDT) algorithm to keep application instances running at distinct locations synchronized. The peer-to-peer nature of the CRDT library ensures that we do not need to set up a centralized entity responsible for merging conflicts, leading to better scalability characteristics. The library is prepared to endure offline phases; in practice, models can be continued to be worked on while the system is not connected to the Internet; models are synchronized with peers after reconnection. Yjs offers multiple data types like Y-Text, Y-Array and the key-value based Y-Map. Our globally synchronized data store makes extensive use of the map-based data type; every model element gets a unique identifier which serves as the key in the synchronized model element. As associated value, we use an array that contains the model element's type and further extensible properties.

We have implemented three metamodels as examples in the Direwolf Model Academy. In the following two sections, we present two of them. First, we discuss the combination of OpenAPI and IFML. Second, we show how we can build user interfaces for Internet of Things devices. The last example, an iStar 2.0 editor, is not presented here; however, it was used in the context of a lecture for training students to model iStar 2.0 strategic dependencies [KKJa20].

[15] https://polymer-project.org
[16] https://svgjs.com

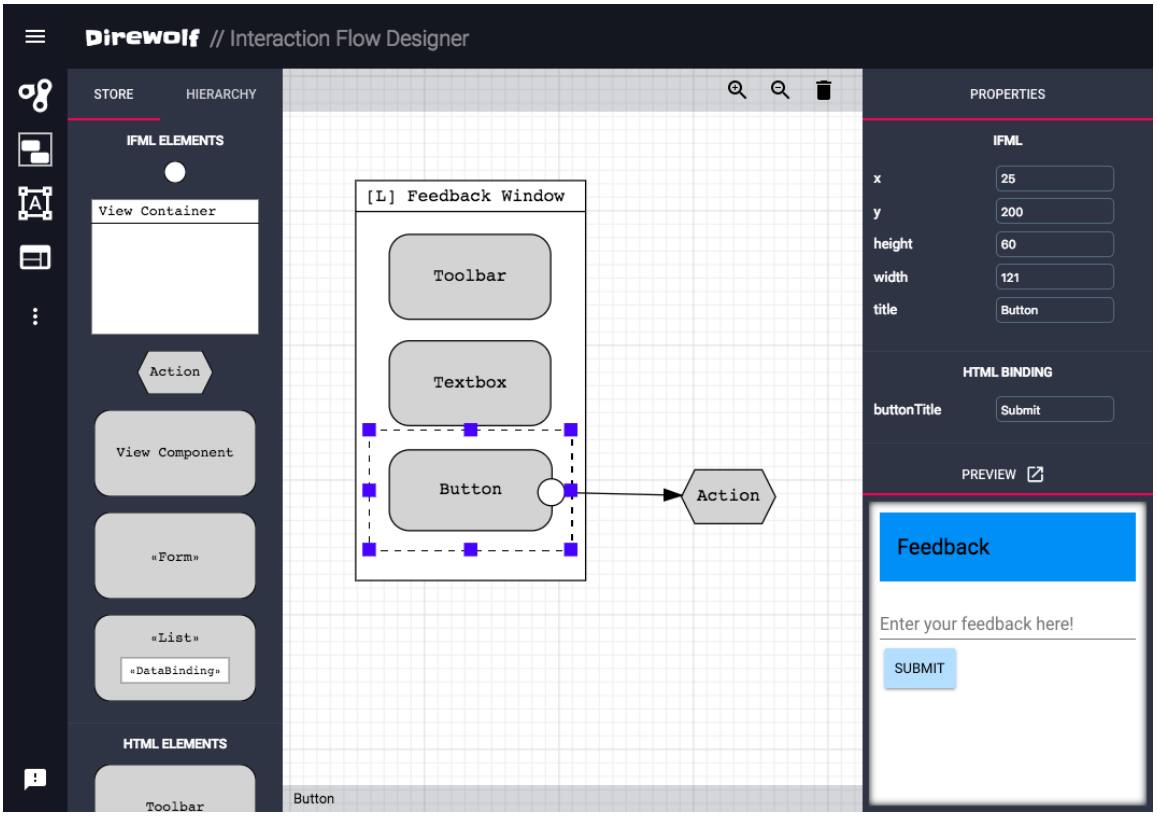


Figure 4.4: Interaction Flow Designer and HTML Preview

4.4.3 Generating User Interfaces for Web Services

In this section, we explore an example of the Direwolf Model Academy framework in use. As the underlying conceptual model, we use IFML as presented in Section 4.2.3. IFML is a visual domain-specific modeling language for creating visual models of user interactions and frontends. Its expressiveness allows to directly generate frontends in various frontend description languages. The main modeling concepts include a View Container which can contain multiple View Components. They can be associated with Events and Actions. State transitions and data flows are modeled as edges with Interaction Flows or Data Flows. Here, we realized an on-the-fly transformation to HTML. Additionally, the Direwolf Model Academy framework allowed us to develop a generic model palette that generates model elements based on the widespread API documentation specification language OpenAPI.

Figure 4.4 shows the developed IFML editor running in a Chrome browser. The application targets all stakeholders involved in the development life cycle of a Web application. On the left, a vertical tab bar provides icons to access different parts of our application. There are subviews for requirements within Requirements Bazaar, the model editor (selected in the screenshot), an end-user-oriented HTML designer, and a preview. The main view is vertically divided into three areas. The left part contains the model element palette and hidden behind a tab a hierarchy viewer with a structured list of model nodes. The center contains the actual visual modeling tool. Nodes and

edges can be selected, resized, and removed. Once a model element gets selected, its shared properties are listed in the property browser on the right. Below, on the bottom right, a preview pane shows the resulting HTML frontend which is generated on-the-fly.

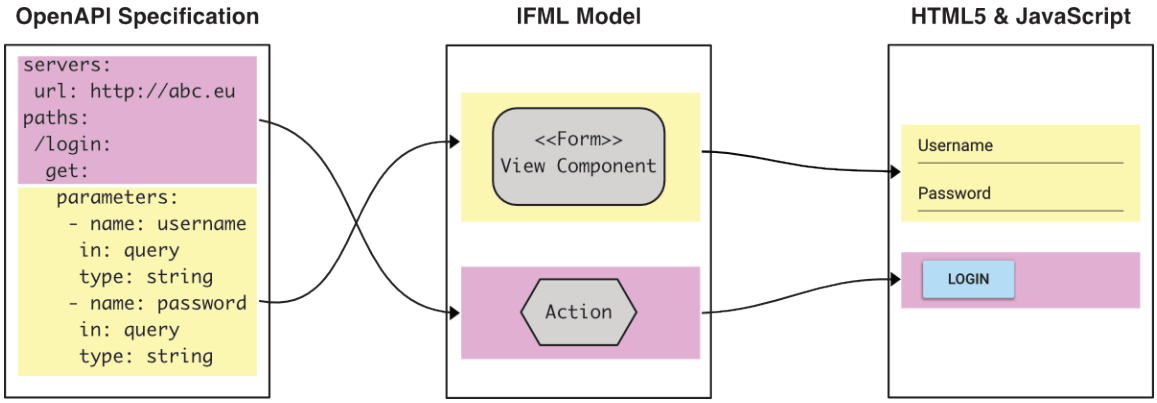


Figure 4.5: Transformation of an OpenAPI Documentation over IFML to HTML5 & JavaScript

To get the model elements into the palette, a transformation of methods described in OpenAPI is performed into IFML nodes. The procedure is shown in Figure 4.5. The figure highlights concepts that are transformed into each other with the same background color (from left to right). In the OpenAPI specification (simplified representation), the URL of a server endpoint is encoded through the combination of the server property and an item in the paths array (purple/dark highlight). This information is encapsulated in the Action of the IFML model. The HTML5 & JavaScript instance contains the URL information in the method that is called when submitting the form. The form itself is represented by a View Component in the IFML model (yellow/light background). It originates in the parameters property of the endpoint.

The HTML editor provides an end-user-oriented view on the conceptual IFML model. The visual appearance follows the modeling view, but instead of IFML model elements, it contains actual HTML widgets like a toolbar, an image element and a button. The HTML preview in the center is updated in parallel with the underlying IFML model. An editor mode can be activated to select HTML widgets. The shared properties of the generated HTML instances are then shown in the property browser on the right. Our Direwolf Model Academy instance provides a bi-directional synchronization of the IFML model with the HTML instance, i.e. once an HTML widget is drag-and-dropped on the HTML preview, a respective IFML view component is added to the model.

4.4.4 Connecting IoT Devices with Web Applications

As a last example for Direwolf Model Academy, we show how to make use of Internet of Things sensors in Web applications. This demonstrates the powerful nature of the model-driven approach that is in turn powered by open standards. Here, the combination of different formats from various domains interact together to combine the strengths of various technologies for innovative new use cases. While smart devices are currently finding their way into countless households, users are often frustrated; many of these devices, including smart lightbulbs, switches, and temperature sensors,

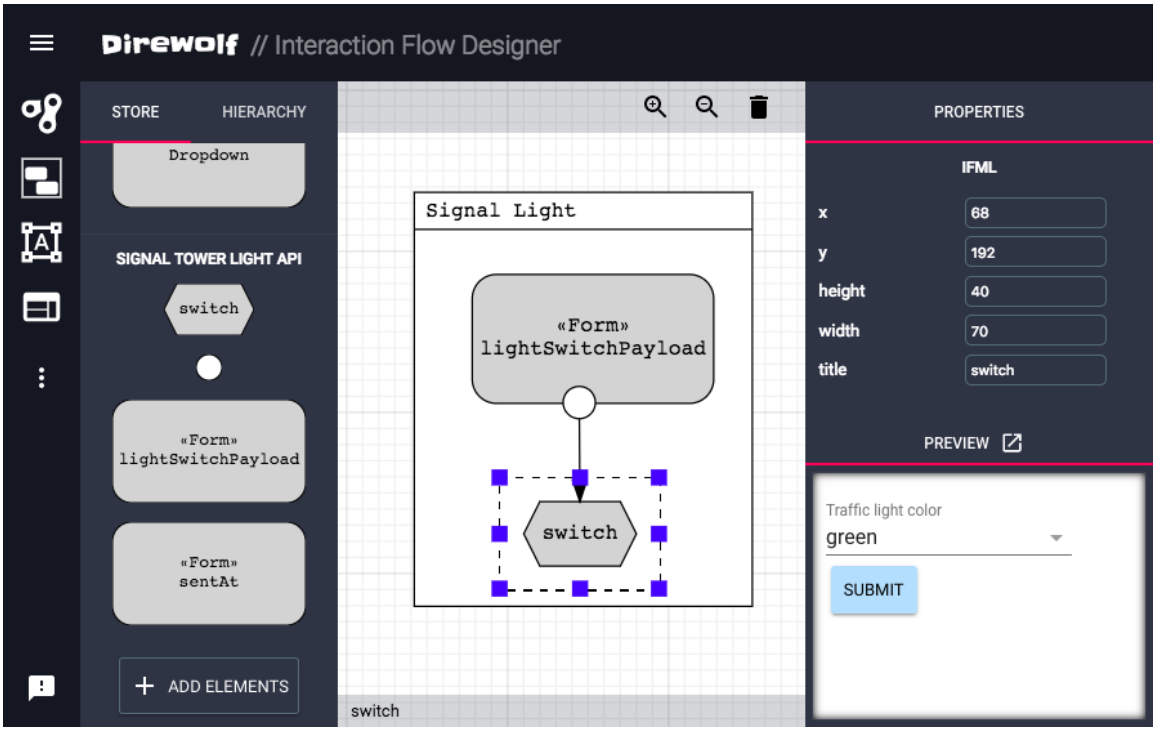


Figure 4.6: Direwolf Interaction Flow Designer and AsyncAPI

come with their own apps, and interoperability is hard or impossible. For instance, it is often not possible to connect a motion sensor of vendor A to a light bulb of vendor B. Even though both these devices are connected to the Internet and offer APIs. As manufacturers push their own ecosystems to establish them as the de-facto standard in the long run, usability suffers as a result. In literature, there are various examples of applications allowing communities to interact with their environment through the use of IoT technologies. Snap-To-It [FNC*16] allows users to opportunistically interact with any appliance simply by taking a picture of it. The authors of [MCJ*15] repurpose existing Web analytics technologies for IoT applications. Crowd-based analytics on top of a video archive is realized in [SSX*15]. Our goal is to achieve the same combination through a model-driven approach, similar to the use case presented above, where we connected user interface elements with service APIs.

In OpenAPI 3.0.0, callbacks can be defined to register out-of-band event listeners, to notify an external service in response to certain events. These event listeners are also known as *webhooks*. However, no real asynchronous behavior can be specified that is native to asynchronous event-based protocols like MQTT and CoAP. To this end, the AsyncAPI was created. AsyncAPI is an API documentation specification which is based on the recent OpenAPI 3.0.0 standard described in Section 4.2.1 [Asyn19]. It allows a human- and machine-readable documentation for APIs that are not based on request/response, but publish/subscribe. This is typically the case for sensors employed in the Internet of Things. Examples are temperature sensors who regularly update the temperature on a server. Here, the server publishes the AsyncAPI description. The clients can then parse out the needed endpoints, either based on a decision by the developer, or dynamically via

schema-matching.

Similarly to OpenAPI, the specification starts with an info part listing the version and license of the API. Then, server endpoints are listed. The asynchronous equivalent to paths are topics. AsyncAPI defines a best practice for topic descriptions in the following format:

```
company.service.1.event.user.signedup
```

Here, an event is emitted once a user signs up to a service by the company. The numeric literal stands for the version number of the API. Instead of event the specification can also list a command, which describes an operation that needs to be performed. Based on the concrete protocol the specification is used for, the topics are translated to equivalent descriptions in the development process. In MQTT, the dots in the above topic description are translated to forward slashes. Topic names may also include variable parts encompassed in curly braces. The components part at the end of the specification lists the data types and schemas of the input or output parameters. They are based on JSON schema [WrLu16]. Similar to the Swagger ecosystem, code generators for AsyncAPI are available.

To allow asynchronous IoT user interfaces in Direwolf, we created an AsyncAPI transformer as a view on top of the underlying IFML model. Analogous to the OpenAPI model transformation, the developed AsyncAPI transformer allows to create user interfaces for simple tasks. We have implemented an industrial traffic light circuit as an example. The screenshot of the wired interaction flow is presented in Figure 4.6. It shows a top-level view container “Signal Light” with an enclosed Form called “lightSwitchPayload”. The payload is directly specified in the AsyncAPI and represents an “Enum” field with the possible values *red*, *orange*, *green*, as well as *red-orange*. The form contains an event that is fired once the form is submitted. In the created IFML model, it is connected to the action “switch”. In the execution phase, the submission of the form thus results in the triggering of the action, which publishes an MQTT message to change the signal light. The traffic light is subscribed to the MQTT node specified in the AsyncAPI document.

A preview of the resulting user interface is shown on the bottom right of Figure 4.6. It features the aforementioned form, and a submit button. The event-trigger action is generated in JavaScript and executed in the background, once the submit button is clicked. As the code of the frontend is generated as a combination of HTML, CSS and JavaScript, combined as standardized Web Components, it can be used in manifold ways. First of all, it is embeddable into arbitrary third-party websites through direct insertion of the static HTML markup. A similar principle can also be applied within the WordPress system. As shown in Section 4.1.3, it is one of the most widely used content management systems. Thus in combination, Direwolf allows the insertion of certain specific user-interaction-based Internet of Things functionalities into any website.

4.4.5 Discussion

Eliciting stakeholder’s mental representation is one of the key objectives of modeling in order to abstract from the real-world. It is a highly social activity, yet in software development, it is often only performed by software architects or developers. Integrating domain experts is promising. In our literature review, we saw many examples of metaphors that are supposed to make it easier for

end users to understand the concepts. However, existing metamodels, modeling tools and their deployment is difficult and proprietary. To this end, we presented the Direwolf Model Academy, a collaborative Web-based framework. In our examples, we showed how it builds on the Interaction Flow Modeling Language. We argued, that IFML is very close to both developers and end users, due to its expressiveness root in Web-based interactions. Yet, it is based on UML and standardized by the OMG. Besides, it is also applicable to the iStar 2.0 modeling language. Its expressiveness is limited, thus it is not applicable to any type of metamodels. For instance, associations with three elements, common in UML, are currently not accessible for our transformation approach. It also does not deal with semantic conflicts happening to the metamodel. To this end, we plan to highlight invalid model elements.

4.5 Conclusion

In this chapter we continued the DevOpsUse life cycle and took a look at the role of service interfaces and models in the software generation process in a broader sense. To this end, we have utilized the requirements and ideas of end user communities as input, as described in the last chapter. The main focus of this chapter revolved around the research question, whether we are able to integrate end users in the code creation phase, similar to the social requirements engineering functionalities presented earlier. We answered parts of RQ1, namely the APIs and interfaces as fundamental constituents of the Web-based information infrastructure. We also showed and formally verified, that (legacy) REST-based services can be converted to state-of-the-art GraphQL interfaces, thereby demonstrating the exchangeability of infrastructure elements. The exchangeability of further protocol elements will be demonstrated in the following chapter with XMPP and WebRTC. Concerning RQ2, we then showed how APIs can be linked together with user interfaces to develop community-specific information systems. Based on our community-driven approach, our methodology stresses the collaboration within a community of practice, i.e. end users are not left alone, but work together with developers who are able to guide other community members. We have first recognized how the increasing modularization of monolithic information systems into mashups of reusable components and services plays into our hands. To this end, related work for wiring up existing services and user interfaces has been presented. We then introduced formalizations of different building blocks. In particular, we showed how with model-to-model transformations we can formally strengthen different generations of Web services. As an example, we discussed the current introduction of GraphQL as a query language for Web services. Even though many real-world REST-based interface descriptions are less expressive, we are able to generate a wrapper that is accessible over GraphQL. This allows REST services to continue to be used, with all the advantages that GraphQL brings to both frontend and backend parts.

We then presented a model-driven methodology for the transformation of API documentation formats into frontends with user interfaces. It is based on documentation formats of Web services, and a UML-based description of the interactions and UI elements on the client side. While an extensive evaluation of the Direwolf Model Academy framework is yet to be performed, we successfully published several articles about our work and were able to gain interest in research and practice [KoKl18d, KoKl18b]. However, graphical user interfaces are only one side of the coin. Voice-based human-computer interfaces in the form of intelligent speakers are already ubiquitous

today. In the future, we will face the challenge of having to operate a myriad of different device form factors: TVs, smart screens, head-up displays in autonomous cars and possibly, holographic displays that will pop up in front of us no matter where we are. Software will have to be created for these systems as well. This means that the already unfavorable ratio between users and developers will become even more unbalanced in the future. This has to be accompanied by a focus shift from the device on to the user's task. In turn, it shows the significance of model-driven approaches; once the underlying foundations are settled, it is easy to create device-specific transformations. Therefore, our findings will remain applicable in the future. Moreover, the results allow us to explore further questions, such as what tools we need to enable users to use multiple devices simultaneously and possibly switch spontaneously from device to device. We are confident, that this will be a multiplier for the usefulness of the underlying model and prove its scalability.

Although the particular (canonical) protocol can be replaced on the access layer, the role of an API therefore remains that of an essential bridge on the Web. The use of a wrapper, as we have done in this chapter, is always accompanied by a balancing of advantages and disadvantages. Thus, although the use of a wrapper can lead to some relief on the client side and on the connection (less data has to be transferred through efficient queries), on the other hand new bottlenecks can arise on the server. To counter this, both Web APIs and corresponding protocols must therefore co-evolve, which on a large scale can only be done by a sustainable development methodology. In the case of a largely centralized client/server architecture this is still justifiable, but becomes much more difficult in the case of peer-to-peer, where the protocol, i.e. the mutual understanding of peers, is even more essential. Therefore, as a next step, we look at how generated applications can be deployed on community-specific infrastructure. It is also of big interest to overcome the gap between Web frontends and Web services as entities commonly distributed in a client/server model. The next chapter will therefore introduce and analyze peer-to-peer technologies on the Web.

Chapter 5

Peer-to-Peer Computing on the Edge

The history of computing machinery began between 1950 and 1970 with mainframe computers in large air-conditioned rooms specialized on solving differential equations. In the 1980s, we then experienced a move towards smaller minicomputer-based machines, and finally, in the 1990s, to a mass-scale adoption of desktop computers. With the increasing availability of data and the limitations of personal computers to deal with its storage and computation, the place of large-scale processing then shifted back to central locations, the so-called *cloud*. This is often compared to a pendulum that swings between centralized and decentralized architectures. *Edge computing* is a topology coined recently. Its claim is that information processing, if possible, should take place where it occurs, and foremost, where its underlying data is collected. Additionally, the computing and network infrastructure from data source to cloud is used in a resource-friendly manner without the need to transfer large chunks of data across large distances and across the network infrastructure. The term "edge" conveys a spatial association and thereby the principle of decentralization. As more and more everyday devices generate data in the Internet of Things, processing and intelligence gets driven towards the source. The advantages of the model are manifold and include improved performance, less data volume to be transferred, more focused network utilization and thereby less latency, reduction of the potential error sources, improved compliance, simplified data protection, increased data security, and reduced operational costs. On the other side of the coin, however, it may lead to increased complexity of communication. Prime use cases are data-intensive areas such as smart cities, virtual reality, and last but not least, smart factories, where cyber-physical systems operating on real-time data aim to increase quality and efficiency of industrial machines. Critical data that needs to be protected particularly includes digital healthcare and integrated mobility information. These areas would benefit from decentralized storage.

Edge-oriented data processing does not only cater for industrial cyber-physical systems. Instead, it is also applicable for areas directly involving humans, e.g. natural language processing for the fast processing of voice commands and the associated use of artificial intelligence. Therefore, machine learning and neural networks are now moving down the network closer to the user. Through the use of decentralized technologies, networks such as wide-area networks are relieved, systems are generally more resilient and at the same time more flexible. However, fast data exchange with low latency protocols are needed. To this end, we currently witness the worldwide rapid replacement of still rather new 4G systems by the introduction of 5G. The latter technology promises above all a

lower latency. Another evidence of edge-oriented computing is the launch of new processing units for machine learning models right at the edge, for instance by Google [Cass19]. Also, minified libraries for the *TensorFlow* machine learning framework by Google are now executable on smartphones running on the Android operating system. Finally, *software-defined networking* is currently being researched to place certain light-weight data cleaning and aggregation tasks on infrastructure hardware such as switches and routers routers [PGH*19].

In this chapter, we discuss, how communities of practice can leverage edge-oriented computing and its promises like low latency and distributed computing using a peer-to-peer topology. Thereby, our understanding of "on the edge" includes everything at the periphery of the Internet that leverages its networking infrastructure for any application case. To demonstrate the inherent advantages of this computing model, this chapter highlights four aspects in particular:

- **Decentralized Service Deployment:** When changing the delivery model of Web applications from client-server to peer-to-peer, the deployment and delivery parts of the infrastructure also need to be taken care of.

- **Peer-to-Peer Video Streaming:** The exchange of multimedia data between clients is a particular challenge in terms of transferred amount of data. By shifting the load to clients, we have to carefully weigh upstream load with the benefit of lower latency.

- **Distributed User Interfaces:** The proliferation of end user devices and the multitude thereof requires new frontend architectures that leverage the particularities of each in- and output device. Here, the question is how to improve the coordination of these devices in near real-time.

- **Internet of Things:** Direct access to Internet of Things devices from nearby mobile devices is a particularly obvious case of how peer-to-peer technologies may decrease latency and increase privacy.

These use cases represent innovative boundaries on the Web requiring a stabilized community-oriented infrastructure. Within the DevOpsUse life cycle model shown in Figure 1.2, these mainly concern the deploy phase. For the challenges listed above, we developed four design artifacts as system prototypes to research the different aspects of peer-to-peer computing at the edge. We start with the *Layers Box*, a federated cloud-in-a-box that brings learning-oriented microservices closer to the communities using a hybrid cloud architecture. We then go to the protocol level and present how using the peer-to-peer WebRTC group of standards for browser-to-browser communication can be used for shifting the load from the cloud to clients.

Then, we show how WebRTC also reduces latency significantly in the context of distributed user interfaces, enabling new use cases like gaming across devices. Concerning the Internet of Things (IoT), we present our approach to integrate user interaction with IoT devices into HTML5 pages. Finally, we give an outlook, how the recent technological advancement of *serverless computing* fits into our infrastructure.

5.1 Related Work

This section discusses related work regarding the three edge-oriented areas mentioned above, i.e. service deployment, video streaming, and the Internet of Things. One of the first trendsetting articles on edge-oriented computing was published by Satyarayanan et al. in 2009 [SBCD09], lately also called the "founding manifesto" [Saty19]. Since then, the original authors and collaborators have presented various extensions of the original idea. One of them deals with "cyber foraging", the provisioning of virtual machines at the edge [HPR*13]. In the context of cloud computing, even more dispersed computing approaches talk about "fog computing" [Kova14]. A survey of decentralized approaches for the cloud was published by Ferrer et al. [FMJo19]. Didzdarevic et al. focus on communication protocols for the IoT in the area of fog and cloud computing [DCJM19].

In the next sections, we discuss related work, the associated technologies and their history in the areas of microservices, container-based deployment, video streaming and progressive Web applications. These are in relation with the aforementioned prototypes that we developed, and that are presented thereafter.

5.1.1 Microservices

Over the last decades, Web applications increasingly inherited more responsibilities and with that, adopted more functionalities. Together with the shift to more computation on the client side, architectures turned into feature-rich service-oriented architectures, where Web clients access certain functionalities over well-defined interfaces like REST. Often, existing services were extended with functionalities, turning them into bulky monolithic software systems. Many challenges are related to monolithic systems, like dealing with specific development languages and design patterns used within a company. This requires specialized developers that are hard to recruit in a competitive job market. The microservice architectural style aims at overcoming these challenges by dividing backend applications into separate, small-scale services. One of the main descriptions of the microservices style can be found in a seminal blog entry by Fowler and Lewis [FoLe14]. According to the article, the term 'microservice' was coined around the years 2011 and 2012. It describes an architecture where componentization is achieved on service-level by having a number of independently deployable services, each responsible for a specific part of the business domain. This brings a number of advantages of which we mention three in the following. First, the components can be developed independently of each other. Each service has its own responsibility, and can be developed using a specific programming language. This is particularly beneficial for functionalities for which certain programming languages or styles are more appropriate. Second, as services are deployed independently, there is no single point of failure, thus bugs in one service do not necessarily take down the whole set of services. Third, each service is responsible for its own data management, thus, similar to the first advantage, any kind of database technology can be used, ideally suited to the specific task. These can be relational or graph databases.

As the services are highly decoupled, the data exchange from the client to the services takes place over defined interfaces. The REST architecture principle is ideal here, as resources can be mapped ideally to individual services. The resource-oriented principles further help developers to separate functionalities in a large service landscape. Therefore it is widely employed in microservice im-

plementations [Newm15]. In a typical microservice architecture, a reverse proxy takes the role of handling and routing requests. It translates requests coming from the client and routes it the appropriate service. Depending on the configuration, the reverse proxy may handle load balancing by replicating certain services with high demand and iterating through them according to a specific strategy. It may also involve middleware to handle authentication and authorization of users, further removing this load from the microservices themselves.

Together with the clear service interfaces comes a high degree of possible automation. Not only can the services be independently deployed, they can also be automatically tested with respect to integration, performance and user acceptance. Thus, they can be rapidly prototyped and put into production. The environment then injects runtime-specific configuration options. It also leads to a high scalability, as the runtime can dynamically duplicate certain services in times of high demand. Compared to a monolithic application, where the whole application would have needed to be duplicated, the advantages are evident. The main selling point of this component-oriented architecture is the mutual independence of the microservices. As they can be developed in different programming language, services of various developers can be mixed together behind one reverse proxy. This mash-up-style deployment pattern is of high interest to communities, who rely on various functionalities in their highly-specific practices. It also fits heterogeneous learning settings in small and medium-sized enterprises (SMEs). Microservices can be deployed on different kinds of infrastructures, i.e. on local servers, as well as in private, public or hybrid clouds. Based on the reverse proxy pattern, they can be replicated in peak times. It is even possible to replicate certain services on external premises in the cloud. In addition to processing power issues, such externalization can also take place on the basis of data protection aspects. For instance, personal data may be processed and stored by a service running on an internal server, while services with high resource requirements but fewer privacy constraints, such as video scaling, may take place on servers with higher resources in the cloud.

State-of-the-art microservice development patterns are an integral part of agile DevOps life cycles (cf. Section 2.3.2). Answering the challenges of monolithic applications mentioned above, DevOps enables companies to recruit employees with various backgrounds, without being required to heavily invest in trainings into the company-specific language. Still, interfaces allow these manifold services to work together. The small-scale but highly controllable component-orientation is a means of the unified development and operation of software products. Despite the mashup-style operation mentioned above, however, it is still difficult for CoPs to bring services together in order to operate them uniformly. While the development within CoPs can be handled by microservice development frameworks, the deployment still needs to be solved. Our goal is therefore the automation of the provisioning part of developed services.

Together with the shift to microservices and faster development cycles, a need arose to automate the deployment of services, as the number of deployments increases with every new microservice. In the field of cloud based technologies, virtual machines gained momentum, as they abstract away the underlying hardware and provide a platform where services can be deployed. Container-based deployment is a step ahead, by abstracting away the virtual machine and, to a large part, even the operating system. They provide a means to separate application instances from each other, that are possibly running on the same (virtual) machine in parallel.

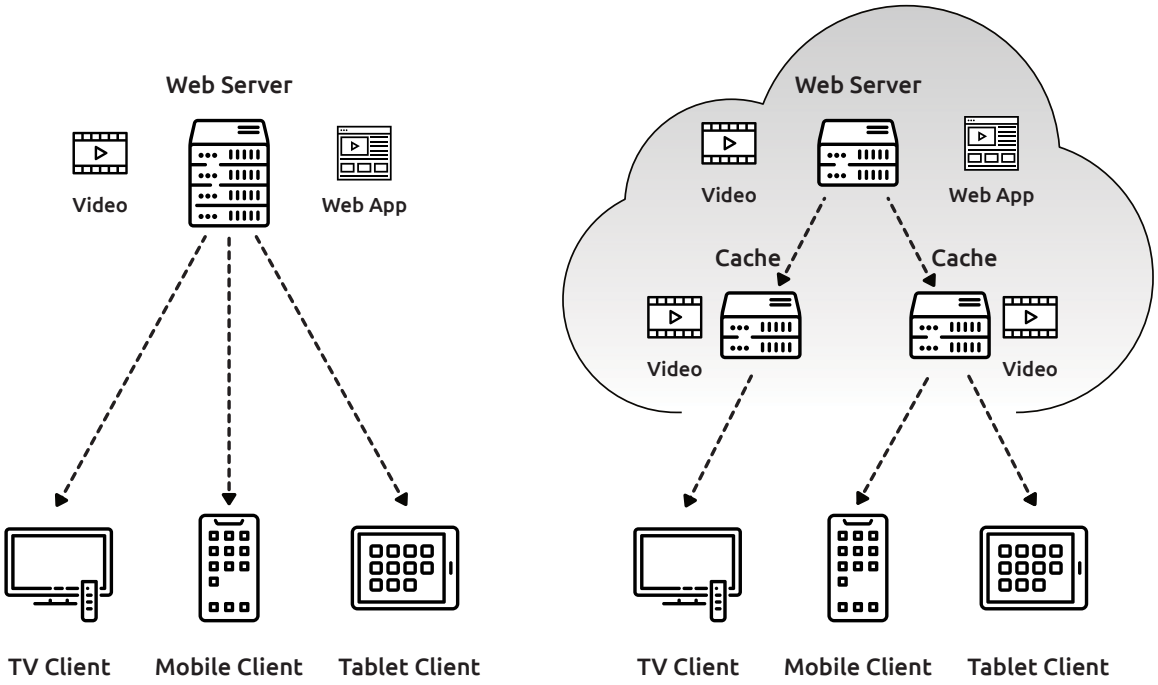
While there are multiple implementations of the principles of container-based software deployment, the environment called *Docker* represents one of the most prominent solutions in the industry. First

released in the year 2013, a whole ecosystem with management tools and even graphical user interfaces was developed around it in the meantime, like Docker Compose and Docker Swarm to manage fleets of Docker containers. *Kubernetes* is an application for the orchestration, i.e. provisioning, scalability and management of container-based applications. It supports a number of container systems like Docker. In the Kubernetes architecture, the "Kubernetes Master" controls a number of "Pods" running on "Nodes". The setup includes an API server that controls the other parts via JSON-messages over a REST-based architecture. Kubernetes orchestration is by now implemented by a number of cloud providers like Amazon AWS, IBM Bluemix and Microsoft Azure.

5.1.2 Peer-to-Peer Video Streaming

Global Internet video traffic has risen substantially in the last couple of years. The continuous increase of connection speed has paved the way for the success of multimedia-related businesses dealing with movie streaming and video-based short messages. In 2016, around 73% of all consumer Internet traffic regarding bits transferred was video traffic [Cisc17]. According to Cisco's Visual Networking Index, this number will most likely rise to 82% in 2021. Large Internet companies like Facebook and Twitter entered the video business, trying to gain a share of the consumer market dominated by YouTube [MoHe17]. There is also a movement towards live video on social networking sites. Meanwhile, a shift of multimedia technologies on the Web from proprietary plugins like Adobe Flash to a standardized set of HTML5-related standards has taken place. Necessary preconditions for cross-device multimedia on the Web were tackled, including licensing issues around media codecs such as WebM and MPEG H.264. However, technological challenges in distributing large video files today are mainly addressed by relying on central cloud providers with large storage capacity, so that the de facto video distribution model still follows the client-server architecture. Video files are first uploaded from client devices to a central cloud solution, before being downloaded or streamed to clients. This entails a number of drawbacks. First, live streaming takes a temporal indirection when routed over a server. Second, it generates a high load on central points of the network infrastructure. Last but not least, video creators must accept the terms and conditions of the cloud provider to serve the video.

Content delivery networks (CDNs), represented on the right part of Figure 5.1 take the burden from the server by caching the video file in a geographically distributed network. A CDN consists of a world-wide network of caches that replicate content of a main server. Requests to the central identity are then redirected to the nearest cache [BPVa08]. A commercial CDN provider usually delivers content for many different content providers. By going closer to the edge, a CDN reduces the latency to the users, reduces server load and eliminates high loads on central Internet hubs. For the Web user, CDNs are transparent; the content provider, however, needs to pay fees. However, because they are expensive to operate and thus are operated by highly professional enterprises, they introduce a further monetary burden, making it hard for small content providers to reach the quality and availability of big players. Coolstreaming and Akamai NetSession are two representatives of systems that relay the caching to the peers. Lin et al. described and evaluated the peer-assisted CDN Akamai NetSession [PGP*13]. The peers of the network need to run instances of the NetSession Interface application. By including a library into third-party applications like download managers, they can also benefit from the network. Running as part of a download manager, a NetSession Interface instance first tries to download the fragments of a file from other peers of the NetSession



(a) Client-Server

(b) Content Delivery Network

Figure 5.1: Comparison of Conventional Video Streaming Architectures

network. Fragments that cannot be received fast enough are downloaded from dedicated servers of the CDN. Additionally, a running NetSession Interface uploads already downloaded fragments to other NetSession Interfaces. For both systems to work, a client-side software needs to be installed on the local operating system, possibly adding security vulnerabilities. Depending on the number of devices employed by a user, different platform-specific versions must be manually downloaded and configured.

A possible solution lies in shifting the load from the cloud back to the clients, following general ideas of edge computing [SSX*15]. Peer-to-peer video streaming is a promising principle to meet these challenges. Although browser plugins like Adobe Flash and Akamai Netsession have accomplished this already several years go, they have not reached mainstream adoption due to usability and security shortcomings. Today, two recent group of Web standards are here to solve the challenges natively in the browser. On the one hand, the Media Source Extensions control feeding `<audio>` and `<video>` tags with multimedia data [WSW*15]. On the other hand, Web Real-Time Communication (WebRTC) as a group of protocols and APIs solve issues around cross-browser peer-to-peer data streaming [BBJ*18].

WebRTC is an HTML5 API that enables establishing direct peer-to-peer connections between two or more Web browsers. The *data channel* is a communication channel type of WebRTC which enables any binary data to be exchanged between Web browsers. In order to use the WebRTC data channel, a website does not need the explicit agreement from the visitor, however the Web application needs to be delivered over a secure HTTPS channel. To establish a WebRTC connection,

signaling data has to be exchanged between the peers [BBJ*18]. This includes the public IP addresses of each peer and the local IP addresses, if the peers are located in the same LAN. The data is typically exchanged with the help of a signaling server that is reachable from both ends of the connection. When using this approach, both peers connect to the signaling server and use it as a relay for transferring their signaling data to the other peer. The signaling data can also be transferred by other means. For instance, the peer-to-peer WebRTC connection can be established by manually typing in signaling data into an HTML form. Because of firewalls and the functional principle of network address translation (NAT), direct IP- based communication between peers is normally not possible on the Web. Therefore, if any two peers are located in different local networks and at least one peer is behind a NAT-enabled router, a so-called STUN (Session Traversal Utilities for NAT) server is necessary to pass through the router so that the WebRTC connection can be established. In the following, we show an overview of the research around peer-to-peer based video streaming using WebRTC.

Högqvist et al. describe and evaluate Hive.js [RoHo14], a WebRTC-based, peer- assisted video streaming solution. A central tracking server keeps track which player instances are currently connected. Their software then builds a random graph between all viewers of a video. Periodically, each peer selects one of its WebRTC connections and terminates it. Immediately after that, the peer connects to a new randomly chosen peer from a list of potential new neighbors. This list is provided by the tracking server. An evaluation has shown that their system performs well for peer-to-peer streaming with 30 or less peers. Hive.js does not check data integrity of downloaded video fragments.

Nogueira Barbosa et al. describe an implementation of a WebRTC-based, peer-assisted video streaming solution [NoGo14]. The system uses an ISP location and geolocation awareness concept to build clusters of nodes. Each peer in their system belongs to a WebRTC cluster. Peers who belong to the same provider's network and who are geographically close are preferably assigned to the same cluster. All peers in the same cluster are directly connected to each other through WebRTC connections. If a peer needs a media fragment, it first tries to receive it via broadcasting the request within its cluster. If the peer is not able to receive a desired media fragment from the peer-to-peer network in time, it requests it from a CDN. The authors' experiments demonstrate that their implementation leads to high fluctuations regarding the percentage of fragments that are delivered through the peer-to-peer network.

Meyn describes another WebRTC-based, peer-assisted video streaming system [Meyn12]. Due to the fact that at the time of conceptualization no browser vendor had implemented support for the WebRTC data channel, no prototype was created. Nurminen et al. describe a WebRTC-based, peer-to-peer video streaming system [NMJ*13]. Similarly, no working WebRTC data channel was available on mainstream browsers. Instead, the authors evaluated the performance of the MD5 hashing algorithm that is used to validate data integrity of media fragments that were delivered by peers. Finally, in addition to the related work mentioned above that deals with peer-to-peer video streaming systems similar to our goals, we would like to highlight work that examines general quality aspects of WebRTC-based video streaming in the browser. In particular, the MONROE[1] platform, a European testbed for such experiments, can be mentioned [PeMa17]. Moulay et al. use it to analyze the applicability of WebRTC in mobile scenarios [MoMa18]. The results are consistently positive,

[1] https://www.monroe-project.eu/

Table 5.1: Comparison of WebRTC-Based Video Streaming Frameworks

System Property	Framework								
	Hive.js [RoHo14]	Nogueira Barbosa et al. [NoGo14]	Meyn [Meyn12]	Rhinow et al. [RVP*14]	StreamRoot[2]	Viblast[3]	Peer5[4]	Swarmify[5]	**OakStreaming**
Working implementation	●	●	○	●	●	●	●	●	●
Detection of poisoned fragments	○	○	●	○	●	●	●	●	●
Small scale efficiency	●	○	–	–	●	●	●	●	●
Intelligent peer selection	○	●	○	○	●	●	●	●	●
Upload limit per peer	○	○	○	○	●	–	–	–	●
Open source (code/algorithm)	●	●	●	○	○	○	○	○	●

● = provides property; ○ = does not provide property; – = unknown

but the authors mention quality leaps in mobile networks due to insufficient network coverage. On the same platform Sulema et al. analyzed the Quality of Experience [SAAS18]. They come to the conclusion that the video transmission quality of WebRTC is not inferior to that of platform-specific commercial solutions.

Table 5.1 shows a comparison of the discussed related work concerning WebRTC-based video streaming. On the left, relevant system properties are listed that refer to the desired characteristics of such a system. We additionally added the following commercial solutions to the matrix: StreamRoot, Viblast, Peer5 and Swarmify. To the best of our knowledge, there is no major conceptual difference between these four solutions. All of them use a tracking server, advertising that their solution is effective in choosing the best peers for data sharing based on geography and network topology. The four systems use WebRTC signaling servers due to the characteristics of the protocol. The common factor between these companies is that the algorithms are not available open source. Out of the academic solutions, all but one have a working implementation. None are able to effectively capping the outgoing connections after a video has been entirely downloaded by limiting the upload per peer. In our OakStreaming library that we present in Section 5.3, we aim to fulfill all these properties.

[2] https://streamroot.io/
[3] https://www.viblast.com/
[4] https://www.peer5.com/
[5] https://swarmify.com/

5.1.3 Internet of Things

Finally, we take a look at the Internet of Things (IoT). Current IoT frameworks are focusing either on the consumer or the business market [CaGa14, GBMP13, McCa13]. Main differences are for example the acceptable level of security aspects of solutions. While in consumer markets the majority of vendors and customers are quite tolerant against security issues like data protection breaches and possible exploitation of gathered data, the situation in business-oriented markets is very different with respect to emerging Industry 4.0 and security threats through criminals and foreign agencies up to the national security level. While the latter is targeted at by new regulations, government institutions and new defensive agencies in the military, the intelligence sector and the security authorities, the former is expected to be self-regulated at the moment. Therefore, in the area of the Internet of Things, governmental investments are mainly targeted at defense and attack prevention capabilities in the cyberspace, leaving a lot of room for consumer-driven innovations, but also problems for the adoption of technology. Several initiatives have already discovered the enormous potential of the IoT for the Web (Web of Things - WoT) [GTMW11, KSB*17] or social applications (Social Internet of Things - SIoT) [AIMo11, LAGM12, AIMo14, LiDo17, PZCG14]. But only a few development platforms are available [ERAn09]. It will be a major problem for users to organize themselves in the face of the speed of technology, protocol and tool development (infrastructure), while keeping their own agency. For technology and infrastructure, the frequency of adoption waves following the diffusion of innovations (cf. Rogers [Roge03]) is increasing, as the next technological wave is already rolling while the previous one has only just spread.

This related work section introduced research and technological background in the area of peer-to-peer services. In the next section, we discuss Layers Box, an early implementation of a community-oriented service deployment framework based on Docker.

5.2 Layers Box: A Hybrid Cloud Computing Architecture for Learning Services

As a first testbed for edge computing, in the following we discuss the Layers Box concept. It represents a microservice-approach in the context of technology-enhanced learning. It allows to bring industrial-strength container-based solutions to the hand of often inexperienced professional communities. The in-house deployment of a physical box allows the tangible experience of edge-hosted services within SMEs in the specific sector of construction, an area that has not been very IT-aware in the past. Technically, we achieved this by relying on the Docker container industrial standard, that is also employed by large corporations. It ensures that created software packages i.e. containers from central repositories like *Docker Hub* can be downloaded into a locally installed Layers Box. At the same time, professional communities keep full control of their data and can decide if and what data they want to share. Using microservices, the Layers Box is a unified approach for developing a scalable and reliable architecture for learning services. We developed the Layers Box following the DevOpsUse best practices of end user ideation in Requirements Bazaar, customer need prioritization in the House of Quality, and finally continuous automation throughout the development process. The (non-)functional requirements were elicited from interviews with appli-

cation partners within the Layers project. The House of Quality links the individual requirements to the architectural characteristics of our approach.

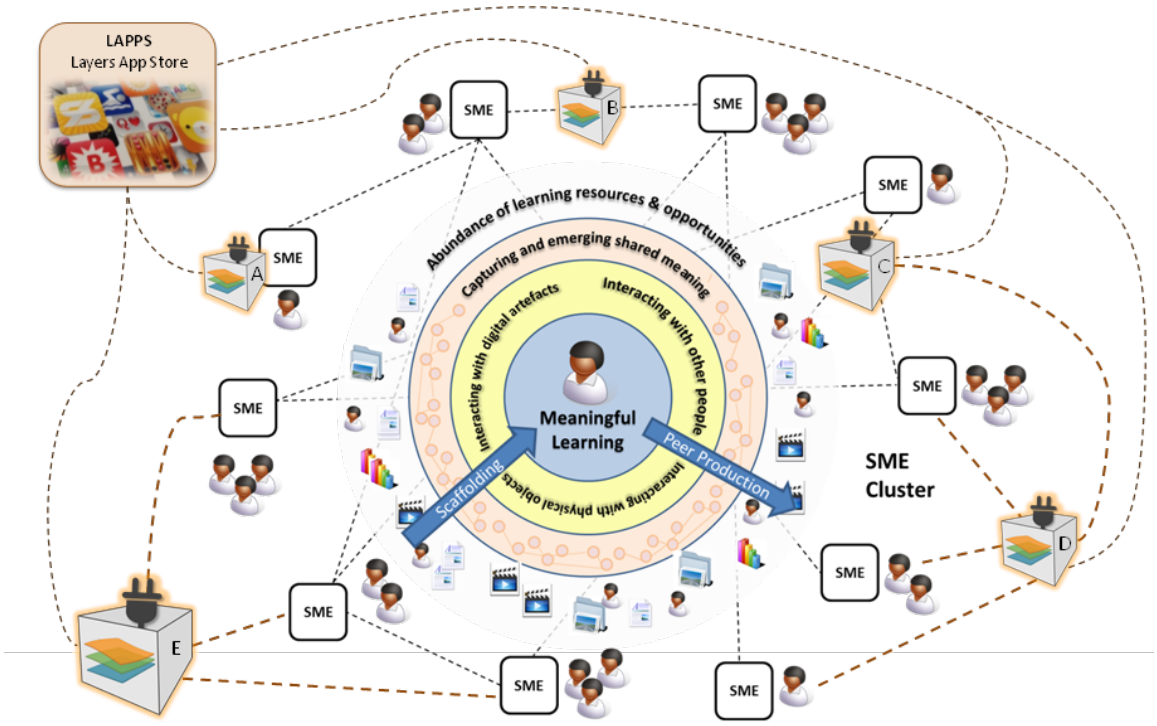


Figure 5.2: Layers Box in a Network of SMEs [DHK*14]

Each instance of a "service host" in the architecture is called a Layers Box. Figure 5.2 presents the Layers project's approach to scaling informal learning together with an architectural view. Each Layers Box can be delivered and installed in different usage contexts, either on existing or dedicated, new hardware. In the picture, the default Layers Box, i.e. the reference implementation of the Layers architecture, is hosted in a central cloud. Another usage option are dedicated installations for single SMEs (e.g., Box A). Two or more SMEs or clusters can share the same infrastructure (Boxes B and D in the figure). The Layers Adapter, presented in detail in the next section, is possibly connected to other Layers Boxes in a federated network. Additionally, it is connected to an app store, where new services can be downloaded by the communities. The Layers architecture foresees three deployment options:

- *Small-scale Layers Box:* It typically runs on commodity hardware such as a desktop PC and relies on installer software (e.g., step-by-step installer or disk images). This deployment setup is useful, for instance, in single-use contexts like in small companies.
- *Medium-scale Layers Box:* This setup relies on more sophisticated server hardware with significant internal and external traffic. For professional communities, the setup may require additional installation services.
- *Hosted Layers Box:* In this deployment model, the hard- and software is provided by a central provider, typically in a trusted cloud environment. The offering can be served with hosting

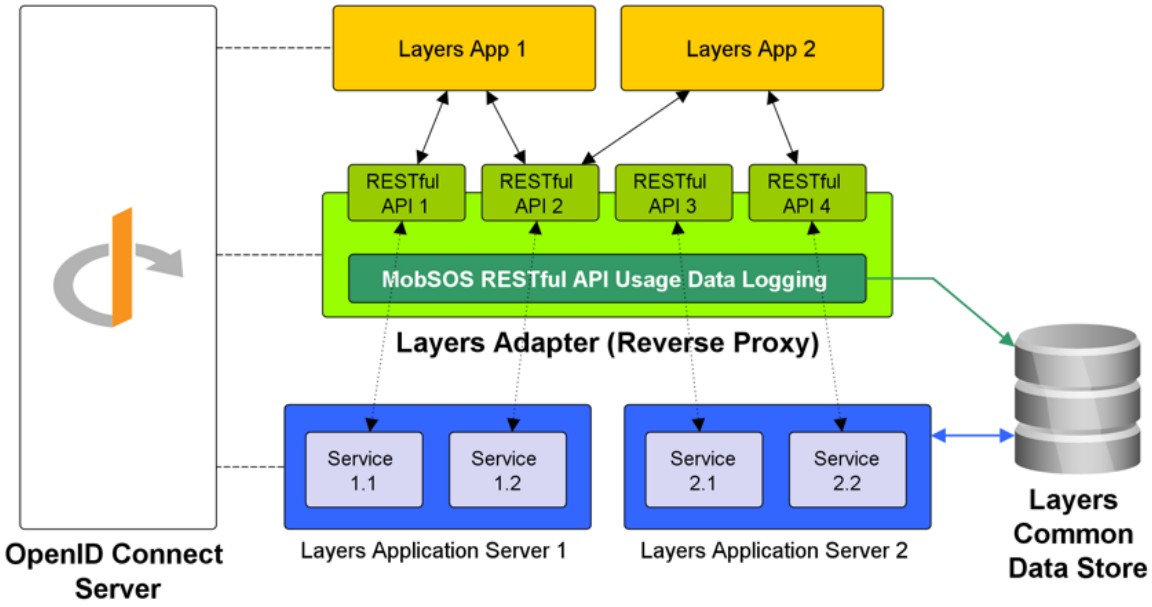


Figure 5.3: Layers Adapter [DHK*14]

fees that depend on a number of factors, e.g. number of users, SMEs, or disk and memory size and throughput.

In the context of the Layers project, the flexible deployment options offered avenues to sustainable business models for the Layers architecture, since the boxes can be marketed and sold as bundles of software, hardware and consulting services.

5.2.1 Layers Adapter

The Layers Adapter is the main controller of all application and service interaction within a Layers Box. It is responsible for the customizability and flexibility of the Layers architecture. Therefore it is part of every deployed instance of the Layers Box and provides unified access to the services included in the Layers Box. Technologically, the Layers Adapter relies on the reverse proxy pattern. In this pattern, a (replicated) single point is responsible for serving requests and to further distribute them among the registered internal services. The internal services and interfaces are not exposed at any time, as even the responses are handled through the reverse proxy before they are being sent back to the requesting clients. Besides forwarding request to internal services, the Layers Adapter may also dynamically rewrite calls to service instances in remote Layers Boxes. In that case, the calls go through another instance of the adapter, forming a layered network adapting to the current load.

Every Layers Adapter comes with a small set of preinstalled services in order to empower its community to install and use third-party services. One of them is a user authentication and authorization service, that we discuss as example in the following section.

5.2.2 Single Sign-On with OpenID Connect

A critical component in all user-facing applications is the identification of users and the associated assurance of privacy and security. Therefore, it is of particular importance to embed this functionality as core part within any infrastructure and to provide a stable API and protocol workflow. Authentication and authorization comprise tasks such as ensuring that any user only gets to see the information he or she is entitled to see, and continues with the identification of users across services for related functionalities such as learning analytics. On a technological basis, a number of standards have been proposed in the industry over the last years. The *Lightweight Directory Access Protocol* (LDAP) is an industry-strength solution that is employed by many companies in the form of the commercial *Microsoft Active Directory* software. Other solutions include Shibboleth. A distinction is to be made concerning authentication and authorization. While authentication verifies the identity of a user, authorization ensures that an authenticated user only gets to access data he or she is entitled to see. *OpenID* is a Web standard for authentication, while *OAuth2* is an authorization protocol. OpenID Connect (OIDC) [SBJ*14] represents a new concept as compared to plain OpenID it provides the identity layer on top of the authorization standard OAuth2. The goal is to specify a standardized protocol that would allow a user of a Web application to access an API, such as the Instagram API, on his or her behalf, without the client Web application having to know the username and password of the user at the API provider. In recent years, the protocol has become the de facto standard for client authorization in APIs and is used by many large providers such as Facebook, Twitter and Google. However, it is also increasingly becoming established in initiatives driven by society, research and politics, such as the international Auth0 and the German netID.

For the Layers Box reference implementation, we chose the open source MITREid reference implementation of OIDC[6]. We tailored it to our needs and provide Docker containers that make it easy to set it up on custom Layers Boxes. Underneath, the user database is linked to an LDAP backend run on the open source OpenLDAP. By the combination of OIDC and LDAP we allow the Layers Box to be used in many different environments. On the one hand, communities without an extensive IT infrastructure may use the lightweight embedded solution or completely rely on external identity providers like Google or Microsoft that both provide a public OIDC infrastructure. On the other, big companies with an existing user management that complement their services with a Layers Box instance may connect the Layers OIDC to their own existing LDAP solution. In the latter case, the OIDC software that is embedded into the Layers Box can securely be integrated with the company's LDAP system so existing user bases can simply log into the Layers services provided. Any application server environment providing Layers services through the Layers Adapter must be enabled to receive access tokens, interact with an OIDC provider to exchange tokens for OIDC user information, and make use of OIDC user information internally for authentication/authorization purposes. This ensures a horizontal and vertical integration with existing and new information systems hosted from edge to central cloud.

Although OpenID Connect and the linked OAuth2 enjoy great popularity, we lacked libraries to implement specialized application scenarios. We therefore conceptualized, implemented and published two related components to be used in the context of OpenID Connect as open source solutions. The first is a custom Web Component to be used within HTML5 Web applications. It is currently used on Requirements Bazaar and other las2peer frontends. The second application is a

[6]Project information is available at http://mitreid-connect.github.io.

server-side component that allows to login on a public display without direction interaction through a personal mobile device.

Section 7.2 discusses, how the Layers Box and its OIDC infrastructure is now widely used across European research projects, catering for innovative use cases such as augmented reality.

5.2.3 Community-Driven Deployment

While we presented some technological aspects of the Layers Box concept in the sections above, we have not discussed community-specific aspects concerning their empowerment to setup the box themselves. As we stated earlier, the goal behind the Layers architecture is to enable communities to deploy their own box, including basic services for identity management and an initial configuration of Web applications. Layers Box was developed using an agile, iterative approach with multiple evaluations together with the application partners within the Layers project. The first version of the Layers Box software infrastructure included separate applications put into different Docker containers, for instance, one for the Layers Adapter and another for a MySQL database. They were all collected in a central repository on GitHub[7]. To instantiate the containers, a shell script was created that helped telling the Docker host which container images to download, instantiate and link together. The Docker systems knows two strategies for connecting two containers. One is *linking* for allowing direct network communication between two services. The other possibility is through mounting *volumes*. Volumes are directories within a container that are available within others. For instance, container A may define its "/var/opt/conf" directory as volume and mount it into container B. We used this to separate configurations from the actual implementation, which is particularly useful in case of crashes: Should a container shut down because of a crash, a new version can be started quickly, mounting the same configuration volume into it. The "start shell script", however, turned out to be cumbersome to use by inexperienced network administrators at our project partner sites, especially when adding more and more functionalities as services over time. As a next step, we introduced the Docker Compose software. It is able to manage container links and volumes by a simple human-readable configuration file. Finally, we provided a simple API and Web frontend to manage Layers Box instances. The Web application simplifies the use of Docker Compose and allows to deploy new services with a few mouse clicks.

In this section we have presented the Layers Box approach as core infrastructural component for running services in a hybrid cloud environment. However, as discussed in the introduction, any centralization, comes at the cost of latency. Also, data-intensive processing inherent to multimedia streams require powerful hardware able to cope with the demand. To this end, we propose a further shift from a hybrid cloud environment to device-oriented data stream processing. In the next section we present an approach to enable the exchange of video streams from browser to browser.

5.3 Video Streaming Across the Edge

After looking at the Layers Box infrastructure that allows deploying services at the edge, we are now looking at video streaming, as multimedia transfer is the most bandwidth-consuming appli-

[7] https://github.com/learning-layers/Layers-Dockerfiles

cation on the Internet. Our hypothesis is here, that if we can manage this well, other application areas with lower bandwidth requirements can also be dealt with. Running services on the edge of the network naturally changes some non-functional characteristics. One the one hand, the latency between communication partners decreases. On the other, privacy increases, as data runs through own servers instead of centralized cloud providers possibly on other continents. In fact, in the special case of geographic closeness, latency and privacy are mutually supportive: When decreasing latency by going closer to the users, we eliminate the need of routing data through central points in the network architecture, which increases privacy. The natural evolution of this model is to completely give up the dependency on centralized services by enabling data exchange between client devices directly. In the following, we look at how we can break up the dichotomy of client and server by introducing peer-to-peer technologies in our framework stack. There are a number of use cases where this proves to be useful. Through deploying community-specific services on the Layers Boxes, we ran into multiple applications that focused on the discussed non-functional properties. One of them is video streaming, where the same video has to be made available to various end users. Video files comprise a lot of information and therefore have high bandwidth requirements. Besides entertainment and social networking aspects, there is also huge potential in further multimedia-based application cases like distant, collaborative workplaces or visual instructions using augmented reality. Applications in our partner SMEs in the Layers project included down- and uploading video training material, for example about new construction material.

Literature in this area identifies three main challenges for peer-to-peer video streaming [LiYi07, Stoc11]:

- **Fast initial startup time:** There should be no significant lag when starting a stream.
- **Random access:** It should be possible to switch the position in the timeline.
- **Complying with device and network limitations:** The resolution of the target device and the bandwidth need to be respected.

5.3.1 WebTorrent-Based Streaming

In the following, we therefore present our solution based on a WebTorrent tracker, named OakStreaming. Figure 5.5 shows the architecture of our system including its three main parts: the Web server delivering the initial Web application including the OakStreaming client-side library, the WebTorrent tracker responsible for sharing video fragment locations, and the peers interested in watching the video. A video fragment thereby represents a singular image frame of a video. First, peer 1 retrieves the Web application and the initial video fragments from the Web server (1). The server also keeps a torrent file which includes information about the fragments of the video. The client then announces the availability of fragments to the torrent tracker (2). If a second peer connects to the system by retrieving the Web application (3), subsequent fragments are already available on the first peer. Therefore, the torrent tracker announces the availability of fragment 2 to the second peer (4), which then starts a peer-to-peer connection to the first peer (5). The torrent tracker is also responsible for negotiating the direct peer-to-peer data channel between the peers (in the signaling phase of WebRTC).

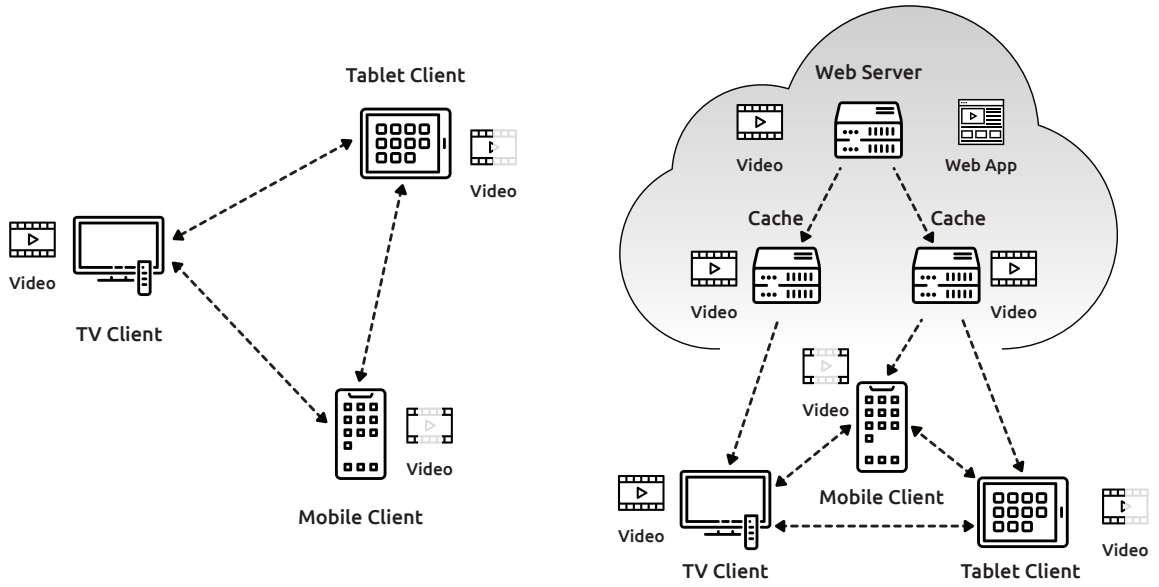


Figure 5.4: Peer-Assisted Video Delivery

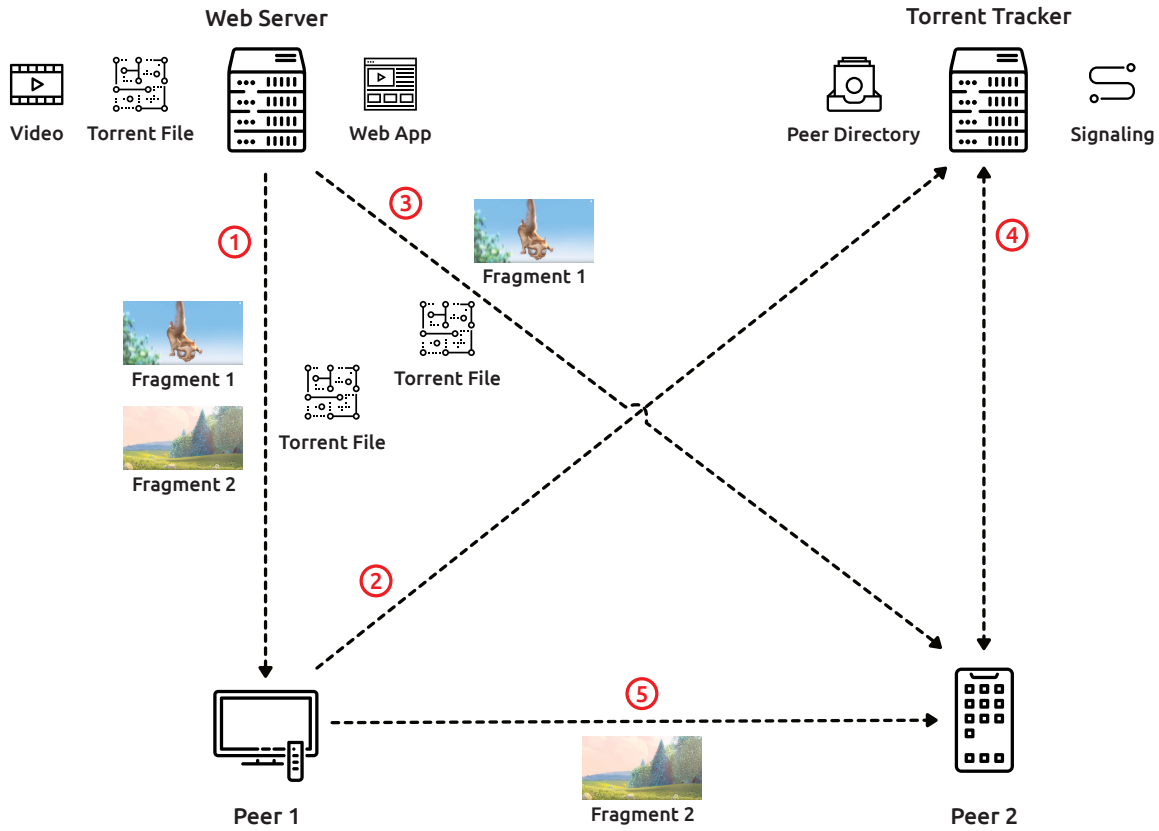


Figure 5.5: Overview of Proposed Architecture and Video Fragment Exchange

The example above showed the simplest case for retrieving video from the peer-to-peer network. We additionally include several strategies for efficient data transfer. For instance, various parameters can be set to limit the amount of uploaded data from a peer; in the example above, the second peer could have already retrieved the first fragment from the peer-to-peer network. Alternatively, the video file can also be obtained from a participating peer. We provide two strategies for downloading video fragments. The first, sequential-piece-selection, downloads fragments sequentially based on the timeline of the video. The second strategy, rarest-piece-selection, first retrieves those fragments, that are most needed throughout the entire peer-to-peer network. The concrete library methods and their parameters are explained in the next section.

5.3.2 OakStreaming Library

In this section, the implementation of the OakStreaming peer-to-peer video streaming library is presented. It was originally devised as a bachelor thesis by Bartels [Bart16]. The system consists of a WebTorrent tracker, a Web server and OakStreaming instances. The OakStreaming instances run on the devices of the viewers. The implementation is based on the WebTorrent JavaScript library. It is an adaptation of the BitTorrent protocol for the Web, using WebRTC connections for exchanging data fragments. We extended its functionality significantly by introducing additional parameters targeting video streams. The OakStreaming library has been developed as a Node.js module which is turned into a Web browser compatible version via the Browserify[8] bundler. The Node.js module only exports a single object which is the constructor to create OakStraming instances; it is global to the browser window's namespace. An OakStreaming instance provides several public methods for the library user.

The three main motivations for developing it were to reduce server load compared to state-of-the-art client-server-based video streaming systems; to avoid transfer of intellectual property to a third party; and to maintain a reasonable quality level for the viewer. Since all major browser manufacturers have implemented the W3C WebRTC or similar specifications, some commercial WebRTC-based peer-to-peer video streaming systems have been developed, while solutions by the academic community lack many desired features. The OakStreaming library extends the state-of-the-art WebTorrent library by various means:

- Configurable limit on the amount of data each peer is allowed to upload.
- Configurable parameter specifying when the client switches from sequential-piece-selection to rarest-piece-selection.
- Dynamic combination of server and peer-to-peer streaming (peer- assisted delivery)
- Possibility to easily add new client instances to an existing peer-to- peer network, without using a torrent tracker, by explicitly exchanging signaling data.

The main challenge during the implementation was that in many third-party libraries we employed, documentation was lacking. Some properties were only described implicitly in GitHub issues and

[8] http://browserify.org/

not in the official API descriptions. This is due to the fact, that the corresponding standard is still very new, and the libraries are not yet fully mature.

5.3.3 OakStreaming Evaluation

The evaluation of our OakStreaming library is divided into a technical analysis and a user study. In the technical part, we measured how the three implemented features added on top of the WebTorrent library affected the video streaming. In the user evaluation, we asked seven Web developers to use the OakStreaming API and give feedback regarding the usability of the library, its documentation and peer-to-peer video streaming in general.

Technical Evaluation

In order to test how well the implemented system is able to organize peer-to-peer streaming, several tests have been conducted with up to eight peers. The tests were run on a middle-end laptop running the Windows 10 operating system and the Chrome browser using the Blink browser engine. Each test was conducted with one seeding OakStreaming instance and 2, 4 or 8 OakStreaming instances which emulated viewers of the video. The video used was a three minute high definition (HD) video that comprised 106 Megabytes. At the start of each test, each OakStreaming instance established a WebSocket connection to a WebTorrent tracker. As implemented in the WebTorrent library, the tracker automatically initializes the WebRTC connection between OakStreaming instances. With two WebTorrent instances connected, they can exchange video fragments. We have found that there are a number of factors that cause a delay of up to a few seconds in the initialization phase of the stream over WebTorrent. The process of querying a neighbor for video fragments could take a significant amount of time, even if the peer-to-peer connection had already been established. Moreover, a peer can only receive file data from its neighbors of fragments whose size is specified during creation of the torrent. These circumstances increase the time from the moment a peer receives a chunk of video data to the moment it serves the received chunk to the viewer. The OakStreaming library uses the default hash value calculation algorithm of the WebTorrent library for the fragment. Additionally, after a WebRTC connection between two peers has been established, the WebTorrent protocol needs time to initialize the neighborhood conditions and exchange information which fragments which peer can deliver. Because of the delays, the emulated viewers were started with a random time offset. First, an interval from 0 to 10 seconds was chosen. Using this interval, the average amount of video data that the viewers delivered to each other was relatively low. Therefore, several interval sizes were tested. Besides 0 to 10, the checked interval sizes were 0 to 20, 0 to 30 and 0 to 60. When using 0 to 60 as time offset interval, most data was transferred between emulated viewers compared to the other three; it was therefore chosen as the offset value for all test runs.

As a result, our tests showed that a lower threshold for the video playback buffer before the OakStreaming client switches from sequential-piece-selection to rarest-piece-selection leads to a higher overall download time. This correlation was expected and confirms the usefulness of the parameter, as with a longer sequential setting, rare fragments are prioritized lower, possibly leading to bottlenecks with video progression. Depending on the scenario, different values may be useful.

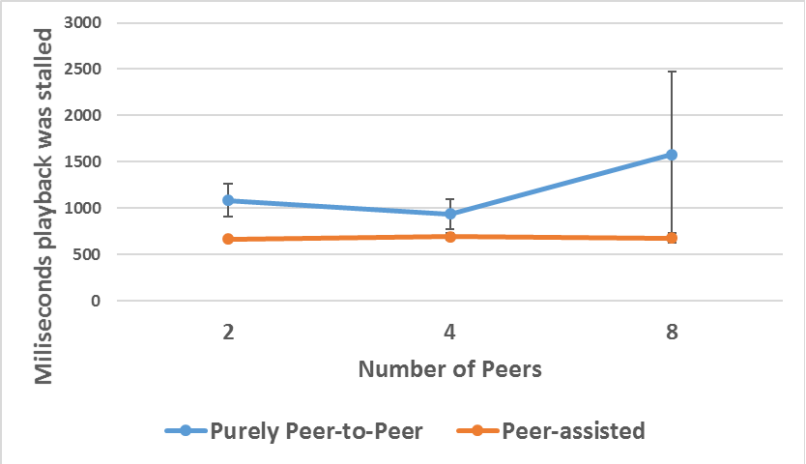


Figure 5.6: Peer-to-Peer vs. Peer-Assisted Streaming [Bart16]

We also tested pure peer-to-peer delivery versus peer-assisted delivery regarding the average time the playback was stalled. Figure 5.6 shows a comparison of pure peer-to-peer and peer-assisted streaming in regard to the total amount the video was stalled, as an average of multiple test runs. As expected, peer-assisted delivery significantly reduces stalling time; here, video playback was only interrupted during initial start-up (around 600 ms).

The results of the technical evaluation clearly indicate that peer-assisted delivery and automatic switching between sequential-piece-selection and rarest-piece-selection enhance the overall quality of service of the peer-to-peer video streaming system, when taking the whole network into consideration. When the peer-to-peer network consisted out of two peers, the average start-up time for pure peer-to-peer streaming was 1086 milliseconds. In case of peer-assisted streaming this number went down to 667 milliseconds. Measurements with four peers in a pure peer-to-peer environment resulted in an average start-up time of 936 milliseconds. In case of peer-assisted streaming this number went down to 695 milliseconds. Overall, we are convinced that for the discussed use cases, the added delay of around 400 ms is acceptable. In conclusion, we can say that the applicability of the library depends very much on the usage scenario. For a comprehensive discussion of the implications for various application scenarios, see Section 5.3.4.

Developer Evaluation

The aim of the developer evaluation was to test the usability and acceptance of the OakStreaming library by asking potential library users for their opinion about the design of the library and peer-to-peer streaming in general. The seven participants drawn out of the pool of student employees at our department had intermediate to advanced skills and experience in the areas of torrent-based peer-to-peer systems, peer-to-peer systems in general, video streaming/hosting and JavaScript. The lab experiment comprised three programming task and filling out the evaluation questionnaire. The programming tasks were designed to make the participants familiar with the API and functionality of the OakStreaming library. The questionnaire mainly aimed at collecting data about the usability

and documentation of the OakStreaming library as well as general opinions regarding peer-to-peer video streaming.

Session Setup and Programming Tasks The participants were invited in groups of two to three and had to solve the same three tasks, but they were asked to work on them individually. The setup was the same for all three programming tasks. The final Web application of each task should be tested in two to three Web browser windows. The participants could conduct these tests independently from each other on their own device.

The first programming task was to create a Web application which uploads and downloads a video to and from a Web server. The second and third programming tasks then both focused on completing the program code of a Web application which streams a video peer-to-peer. The peer-to-peer connections were established locally between the browser windows of the participants.

In task 2, the participants had to use the `streamVideo` method of the OakStreaming library to create a `StreamTicket` object from a video file. The object was then shared over the a synchronized data structure with the Yjs collaboration library [Jahn19]. The received object was then put into the `receiveStream` method of the peer instances.

Task 3 was very similar to task 2 but the participants had to use different parameters and parameter values when creating the `StreamTicket` object. In contrast to task 2, the video should be streamed in the peer-assisted mode and the participants had to set a value for the ratio of time downloaded from server versus peer-to-peer. The parameter values of task 2 and 3 were given by the task description. To solve task 3 it was necessary to read parts of the OakStreaming documentation. Finally, an evaluation form was filled out by the participants. It asked the participants about their knowledge and experience in the areas of torrent-based peer-to-peer systems, peer-to-peer systems in general, video streaming/hosting and JavaScript. Moreover, the participants were asked to rate the usefulness of several features that the OakStreaming library implemented on-top of WebTorrent. Additionally, the participants had to rate the usability of the OakStreaming API documentation. Furthermore, the participants were asked about their opinion regarding peer-to-peer video streaming in general.

Results Five of the seven participants were able to solve all three tasks. Two participants were only able to complete the tasks after short assistance. The questionnaire revealed minor issues with the arguments, like putting together URL and port properties. After the evaluation the respective API was changed to a single URL property instead, which can now handle strings in several formats (e.g. http://example.com:42, http://example.com, example.com:42, etc.). Overall, all features of the OakStreaming library were considered easily understandable.

We were also interested in general opinions of our developers on peer-to-peer video streaming. Most of the participants rarely publish or share, but often consume Web videos. They saw many important advantages of peer-to-peer video streaming like benefits for small content providers with successful videos and the breaking of the monopoly of large content providers in terms of intellectual property rights. Additional remarks of the respondents covered legal issues if possibly illegal videos are streamed between peers.

Table 5.2: Use Case Recommendations for the OakStreaming Library

Use Case	Examples	Applicability
Live broadcast/multi-cast	Sports, Live TV	Moderate applicability due to WebTorrent concept
Time-limited asyn-chronous broadcast (up to 1 day)	Short videos with temporal high demand, social media sharing, TikTok, Instagram Stories	Fully applicable
On-demand with temporal bursts	Educational videos, MOOCs	Applicable, depending on mobile band-width
On-demand	YouTube, Netflix, Amazon Prime	Limited applicability, best for highly-demanded videos (e.g. for releases of new episodes).
Video conferencing	Skype, Webex	Not applicable due to WebTorrent concept

5.3.4 Applicability of OakStreaming

The results are helpful to set further goals regarding research in the area of Web-based peer-to-peer video streaming. Limitations of our work include further evaluation in larger developer and user groups. Privacy aspects in terms of sharing peer information were out of scope for our research, but we are closely following discussions around security issues raised by the introduction of WebRTC in the browser [PIP*18, ReMa17].

We conclude the discussion of our library with some recommendations for its employability in different use cases. Generally, our library is best for videos that are highly demanded within a certain period of time. Because the code is no longer executed after leaving a Web page, the client can no longer upload fragments. While it might be possible to work around this by deploying JavaScript Service Workers[9], other conflicting effects, such as the increasing storage space requirements of large video files or limited mobile data plans, play a detrimental role. Table 5.2 shows typical use cases of video streaming on the Web. We mention examples as well as an applicability estimation of our library.

5.4 Widget-Based Distributed User Interfaces

With the proliferation of new device types and form factors (e.g., foldable smartphones), each individual's "device fleet" is extended. We are now used to having at least one smartphone, a laptop, a TV and further devices like tablet computers and smart speakers. Additional devices such as smart boards are used at our workplaces. While we tend to use the most suitable devices for each application, the hand-off from one device to another is often a hassle. This raises the question of task continuity across device boundaries. For instance, a conversation within an instant messenger

[9]The Service Worker Web standard describes a technology to introduce a client-side proxy based on Javascript.

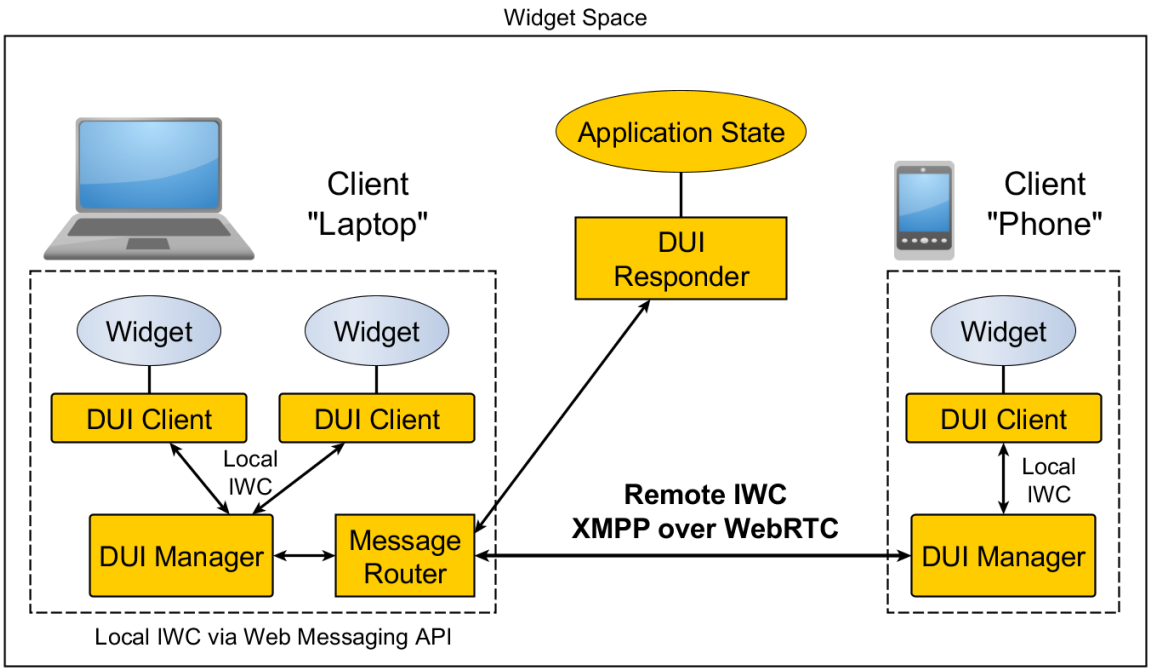


Figure 5.7: Direwolf Distributed User Interface Architecture

on the smartphone, while on the go, may be continued on a laptop computer, once seated at the office desk. Then again, a video started (and possibly cached) at the smartphone may be viewed at a larger TV screen once we arrived at the living room. Another special form of this continuity is the possibility of executing individual tasks simultaneously on several devices. An example is location-based tagging of pictures. For a map, the two-finger panning and tapping is considered much more convenient than zooming and moving with a computer mouse or a touchpad. In this context, existing research shows how Web-based user interfaces can be distributed over multiple devices [SaSi19]. While *responsive Web design* adapts a Web page to different screen sizes, it only targets the UI-level, not multi-device distribution of Web pages. Within the ROLE framework, a widget-based Web application developed in the ROLE FP7 framework at our chair, the first version of the Direwolf framework allowed to distribute certain parts of the user interface to different devices [KRN*14]. The technology used behind was *inter-widget communication* (IWC) [Renz12]. Renzel et al. distinguish two kinds of IWC, *local* and *remote*. Local communication is performed over the HTML5 Messaging API. The remote counterpart, i.e. from browser instance to browser instance, is done using an XMPP connection to a federated network of servers. However, due to the indirection over the server, the Direwolf 1.0 framework is not suitable for certain application cases with strict real-time demands like gaming. Multi-device collaboration has certain latency and security requirements. There is a trade-off between different aspects; for example, if latency is to be reduced, security is often at risk.

To solve the latency issues of the remote inter-widget communication in the first version of Direwolf, in Direwolf 2.0 we introduced direct peer-to-peer connections with WebRTC. Figure 5.7 shows the architecture of our prototype. The message router that handles the remote IWC is now connected to other devices through a WebRTC data channel. To avoid a star topology, where each

devices is connected to all others in the network, we added a relay, that forwards messages between peers. It is located on a single device, that is chosen based on heuristics such as the available bandwidth and other metrics. For instance, the availability of the battery level API in connection with lower screen width indicates a mobile device. In the implementation phase we became aware of a lack of a standard regarding the negotiation of a WebRTC data channel between two XMPP clients. We therefore worked together with the XMPP standardization board and authored an extension protocol that is described in the next section.

5.4.1 XMPP Extension for WebRTC Data Channel Negotation

While the XMPP protocol already offers various means to establish direct peer-to-peer connections between any two peers, the use cases were restricted to UDP streams with the purpose of streaming audio and video data. The WebRTC standard itself transports TCP-like communication over a UDP connection. The standard and interface is called `RTCDataChannel`, and it can be best characterized as "TCP over UDP", i.e., a reliable connection over an unreliable communication channel. To enable these functionalities, we had to extend XMPP with an extension protocol (XEP), that was thereafter successfully integrated in the standards stack [Bave14b]. In the XEP, we present a new `<sctmap>` element which takes care of signaling the necessary Stream Control Transmission Protocol (SCTP) information. To comply with the WebRTC stack, a `<fingerprint>` element is obligatory, as described by Hancke [Hanc13]. The signaling of the actual data channel is performed in-band within the SCTP stream. Due to our use case, we only use in-band signaling for reliable data channels. However, support for both *partial reliability* and *out-of-band signaling* might be added in the future on top of our fundamental XEP.

We also developed a JavaScript library extension to boost the uptake of the new standard. In order to use the popular library `strophe` and its `strophe.jingle` extension in our prototype, we had to add a method to map the corresponding SDP lines to the jingle content element and vice versa; we also had to add methods to setup the channels. Adding data channels to strophe enables a whole new world of near real-time collaborative applications within the browser, far outreaching the research goal of secure distributed user interfaces with low latency.

5.4.2 Distributed User Interface Evaluation

To evaluate the latency reductions of the Direwolf distributed user interface framework, we took real-world measurements. For this, we selected a popular jump-and-run JavaScript game and distributed its user interface across two devices; the input was performed on one computer, while the actual gameplay actions where inputted on another. This reflects real requirements where game controllers are different pieces of hardware — in our case, it was on an entirely different device.

We performed both a comparative technical evaluation as well as a user study to prove our conceptions, of which we present the technical part in the following. For the technical part, we measured the round-trip-times of messages from a widget to an instance running on a remote device. In order to match request and response, we included both a unique ID and a timestamp in the message. All tests were performed on a Sunday morning in the network of our chair, at a time with relatively low network activity to avoid any congestion issues. We performed three tests. Figure 5.8 shows

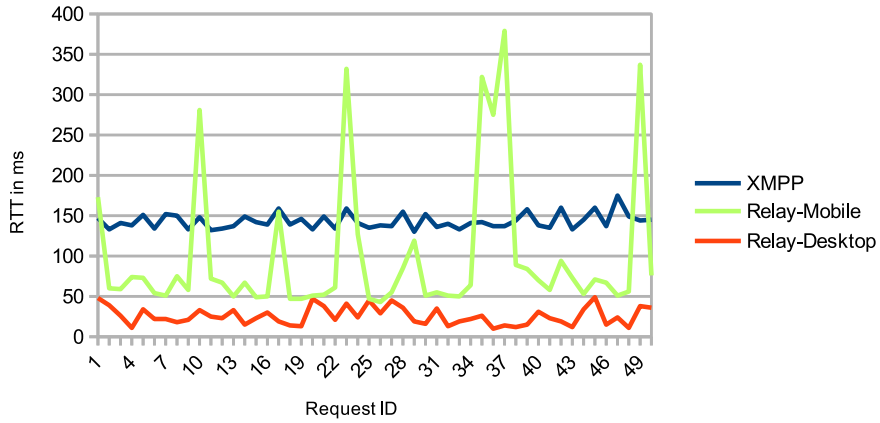


Figure 5.8: Comparison of DUI Latency with Various Protocols and Settings [KBKl14]

the results of the test series which included 50 runs. First we measured the round-trip times on the obsolete Direwolf framework that uses the XMPP server for message routing; the average delay was around 143 ms, with a median of 141 ms. The next iterations regarding the latency aspects were then using the modified version of the framework. While the test situation was the same as the previous one, this time a direct connection was established between the two devices, bypassing the XMPP server. With WebRTC activated and a relay on a powerful Desktop computer, we achieved a significant reduction of round-trip time to 25.8 ms, with a median of 23 ms. Even when the relay was hosted on the smartphone, the average round-trip time went down to 99.2 ms, with the median at 65.5 ms. In this test, several high peaks are present, however, the average is still below the non-relayed test case, proving our assumption that Desktop devices should host the relay, while even low-powered devices make a viable relay host. On a desktop machine, similar spikes can be seen on a lower scale. We were able to lead these back to connection refreshes on the server. On older, mobile hardware the effects are amplified by comparably low processing power available. Nevertheless, we were able to prove the point that all kinds of devices can benefit from our findings.

5.5 Linking Physical and Virtual Worlds

As last example of how technologies at the edge influence the development of community information systems, we depict the Internet of Things (IoT) as inherently edge-based information technology. It is characterized by a myriad of devices connected with several distinct protocols and wireless standards. Thus after the server-based deployment, and the peer-to-peer Web framework, we now arrive at a completely new architectural component, and show how we integrate these new device types into our general framework. The goal therein is to still follow the DevOpsUse life cycle, that is, supporting developers, designers and end users within their communities in setting up and using these devices. This includes usage analytics of to what extent and for which use cases the devices were employed. Again, we leverage Web standards for this task. Making smartphones and wearables aware of the physical presence of artifacts around them is a crucial requirement for any system interacting with these devices. This is particularly valid for learning system that depend on body-worn sensors and in- and output hardware.

In various domains of computer science like software engineering, the detection of capabilities or services from the service user about the service advertisements is called *service discovery*. In our setting, service discovery therefore describes an automated identification of the availability of physical artifacts in the smart space around the mobile and wearable device. To identify physical objects, each object's identifier or at least the object's type has to be advertised. Therefore, every entity (i.e., physical artifact) in our infrastructure must be uniquely identifiable. We achieve this by allocating every participant in the socio-technical network a Universally Unique Identifier (UUID). The canonical format of a UUID uses hexadecimal text with inserted hyphen characters (e.g. `656f1378-b6a8-49b4-8802-01677294c438`). When a user walks around in the proximity of such an identifiable IoT device, the mobile device need to sense the identifier by technical means in order to associate the linked information. In the following, we therefore discuss technologies that allow physical artifacts to expose a unique identifier. QR Codes encode certain chunks of information visually, similar to barcodes that are found on almost every product in supermarkets for many years. While barcodes encode a product's item number in vertical bars, QR codes consist of up to hundreds of squares whose arrangement is able to encode as much information as around 4000 alphanumerical characters. Over recent years, QR codes have gained huge momentum in street and magazine advertisements where the advertised product's website is often encoded in a QR code. Barcode scanners capable of decoding QR codes are now built into every camera app. Additionally, open source QR code readers are available and embeddable as a library within custom apps. The main advantage of QR codes is their cheap creation and customizability by simply generating codes through free online services and then printing them out on any printer. It is also the most interoperable technology out of those presented here, as except requiring a built-in camera, no additional hardware is necessary on mobile devices to use QR codes.

Near Field Communication (NFC) is a contactless technology for transferring small chunks of information between devices equipped with NFC hardware. The principle is established in public transportation ticketing systems and credit cards; the microchip that is usually embedded in the cards has to be held close to the reader for a short amount of time so that the data on the NFC chip can be read out and processed in another step. NFC typically works over a range of a few millimeters; it has an active and a passive mode. The active mode allows two NFC devices to actively exchange information over a wireless link; the speed is usually as low as around 100-400 kilobits per second, which is sufficiently fast for the minimal amount of information that NFC devices

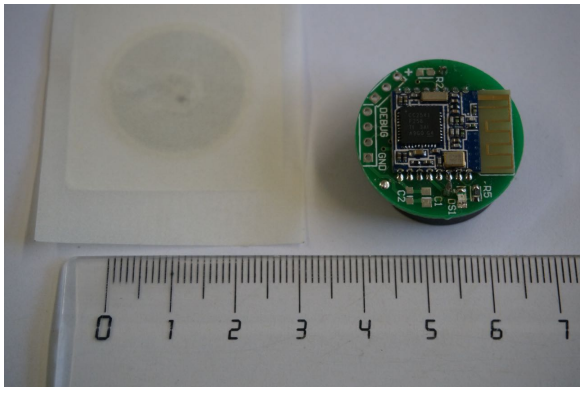


Figure 5.9: Size Comparison (in cm) of NFC (left) and BLE Chips (right)

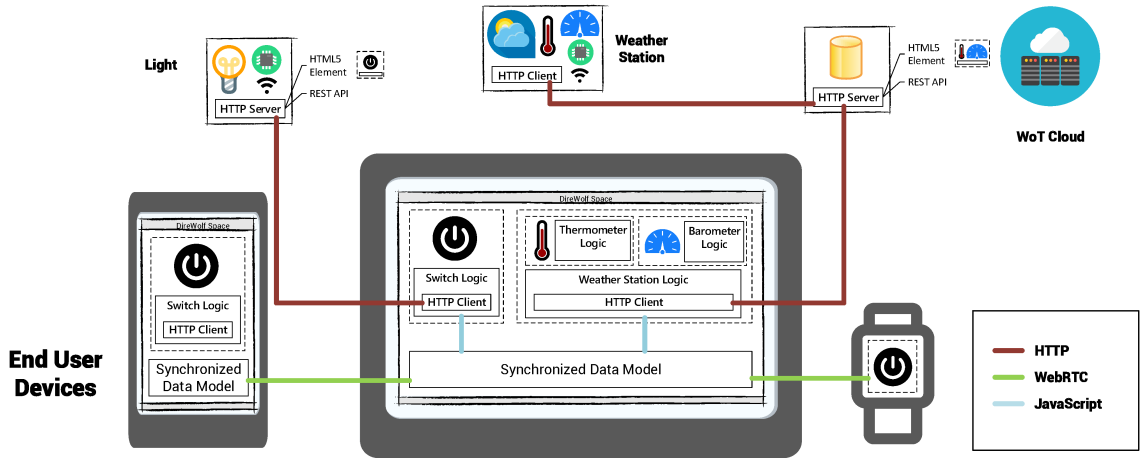


Figure 5.10: Architecture of Web Components for Internet of Things Devices

exchange. In the passive mode, the accessing device is sending out wireless waves on a specific frequency. Via electromagnetic induction, the field in the sending circuit on the NFC chip is activated and enough energy is harvested to answer the request with the data that was previously saved on the chip. These chips are usually called NFC tags. They are available in different shapes such as the already mentioned credit cards, or simple stickers as shown on the left in Figure 5.9. NFC technology is available on state-of-the-art Android smartphones and many tablets. Though NFC is also available on the iOS platform, only the newest iOS 13 API allows further access beyond mobile payment applications.

Bluetooth Low Energy (BLE) is a sub-specification of the latest version of the Bluetooth standard stack for resource-efficient data transfer that is similar to NFC. The circuit for a BLE chip fits on a thumbnail and can be powered by a coin-sized battery for around 1-2 years. To advertise its presence, it periodically broadcasts messages with a configurable signal strength. After calibration, i.e. measuring the signal strength at distance intervals of 1 meter, a mobile device may measure its distance to the BLE chip; typically this can range from a few centimeters to around 70 meters. The broadcast data packets include a few bytes of data that can be predefined similarly to those of an NFC chip. The iBeacon specification by Apple defines the broadcast data to include a UUID as well as a major and a minor ID. The UUID is typically the identifier of a specific deployment; the minor and major identifiers mark iBeacons within the UUID. E.g., a retail chain may define a UUID for all its stores, a major ID for a specific store and a minor ID for a particular iBeacon within the store. While iBeacons are natively accessible through the iOS development framework, libraries with similar functionality are available for Android. Figure 5.9 (on the right) shows a typical BLE chip. In early 2015, Google announced another protocol for BLE chips called Eddystone that directly competes with Apple's stack. The open source protocol specification defines specific message frames that Eddystone-enabled devices know. In addition to a unique identifier similar to that in the iBeacon specification, Eddystone is also able to send a so called Eddystone-URL for sending full Internet addresses. Besides, Eddystone beacons send out a namespace.

We evaluated thes four technologies in a comparative study, combining ideas of end user develop-

ment and the Internet of Things on the Web [KoKl16b]. Figure 5.10 shows a conceptual overview, how smart devices can be embedded into Web applications. The use case is a presentation room, where we want to be able to control lighting from a smartphone. Instead of downloading an app on an app store, and then configure the app to connect to just this smart socket, we instead provide a Web application that is available via a public URL from the meeting room. To leverage the smart socket's capabilities instantaneously without having to configure it with each new device, we equipped it with a QR code. The QR code encodes again a URL to the Web component that is itself controling the hardware. For this reason, we implemented a Web-based QR reader[10], which we developed using the WebRTC Media Source Extensions API, to be able to access the device's camera. As Web application, we extended our Direwolf framework for multi-device widget-based Web applications. As discussed in Section 5.4, Direwolf gives us the conceptual notion of sharing Web interfaces by synchronizing Web applications across multiple devices using widgets. Once coupled with the Direwolf platform, the user interface elements are aligned in flexible grids according to the concepts of responsive Web design [Marc11]. All imported elements get access to a shared data model for cross-device synchronization of their state. An optional master flag on the device the element was imported on makes it the primary responsible entity to access the device and saving parameters in the shared model. This way, we avoid redundant requests to constrained devices. Rather, sensor and other values are only accessed once and distributed over the synchronization channel. In the case of the master disconnecting, the functionality is migrated to a new device.

As technical challenge of the prototype, we identified the missing dependency management of JavaScript on the client. In the current version, all imported elements need to reference the same version of third-party libraries to avoid undesired behavior. *Import maps* that are currently being standardized in JavaScript will presumably solve this issue and allow to exactly specify the needed version of a library. Apart from this minor implementation issue, we are confident, that an easier coupling to heterogeneous Internet of Things devices like the one discussed contributes to a smoother and less error-prone integration and thereby acceptance of new devices and device types. In the next section, we conclude this chapter with an outlook on further innovative measures for seamless adoption of new functionalities, services and apps on the Web.

5.6 Advanced Deployment Patterns on the Edge

This chapter has analyzed and discussed: first, the provisioning of container-based applications in REST-based microservice architectures, second, low-latency exchange of peer-to-peer multimedia and data streams, and third, the integration of Internet of Things devices on webpages. In the following discourse, we present two recent technological developments that are capable of combining these three aspects into a single, coherent application environment. The goal is to show, how the experiences and results achieved with the DevOpsUse life cycle model are also applicable to changing conditions, use cases and technologies. First, serverless computing is introduced as advancement of the microservices model. Second, progressive Web applications as offline-capable Web applications on mobile devices are discussed. Both are contributing to the ongoing diffusion of the traditional client-server model on the Web.

[10] https://github.com/istvank/qr-reader

5.6.1 Serverless Computing

As discussed earlier, virtual machines are abstractions of hardware on top of an operating system. Within virtual machines, containers can be executed that further abstract away the virtual machine and result in a sandbox environment for each application or service run within. The next evolutionary step that is currently being introduced in industrial practices is *serverless computing* [BCC*17]. This step was heralded in late 2014 with the introduction of *Amazon Lambda*. Serverless computing, often called *Functions as a Service (FaaS)*, considers underlying aspects such as server, networking, operating systems, data and virtualization levels such as virtual machines and containers as given. As such, the term 'serverless' does not refer to a peer-to-peer architecture where there are no servers; instead, developers do not have to care about servers or runtime environments. In this model, heavily lightweight, particular functions are running in their own sandbox. This leads to a number of implications for the software architecture. First, the functions have to be stateless, as they possibly only live as long as a call is received and answered. This goes hand in hand with the statelessness of the REST-based paradigm. Second, the data storage level has to be externalized to dedicated data storage services. Serverless Computing allows for a highly fine-granular scalability, as only those functions that are in high demand need to be replicated, which may happen on demand. Current implementations rely on a reactive architecture where events, like requests, trigger the start of a function. From an economical point of view, this allows a pay-per-use model, where only the net runtime is charged. Individual functions can be maintained and updated easily, as the development overhead declines while the comprehensibility is raised with smaller software packages. Examples of real-world implementations of the Serverless Computing paradigm are besides the above-mentioned Amazon Lambda, Google Cloud Functions and the open source IBM OpenWhisk system. Examples of real-world implementations of the Serverless Computing paradigm include the open source IBM OpenWhisk, in addition to Amazon Lambda and Google Cloud Functions mentioned above. Use cases are for example the backends of voice-activated personal assistants, that are only accessed a few times per day. As such, they are a perfect fit for the community-driven development model represented by the DevOpsUse model. As single functions are easier to understand in comparison to a large-scale monolithic Web application, the onboarding of new developers and the integration of new community members within open source initiatives become much easier. Instead of having to work through extensive documentation about the whole system functionality, and setting up a heavy-weight build system for it, they can directly work on a particular component, presuming they are familiar with the programming language and model used.

The serverless computing paradigm can directly be applied within the presented Layers Box approach, as Docker containers offering runtime environments for the functions are available from open source repositories. Due to their scalability, they enable the deployment of services capable of handing the large number of devices in the Internet of Things. For instance, the integration of a wearable device for heart rate measurement through a QR code with the help of the Direwolf framework could trigger the start of a corresponding serverless function that is able to trigger health-related services.

5.6.2 Progressive Web Applications

Progressive Web Applications (PWAs) are Web applications that behave like native applications and therefore exhibit some particular characteristics in both development and usability. They are a recent phenomenon driven by new Web standards that augment some of the capabilities previously not possible in Web browsers. From the user perspective, they are accessible via a standard URL entered in the address bar of common browsers. PWA is not a competing technology to responsive applications, instead they complement each other; among other effects, responsiveness is achieved by automatically adapting the user interface to different screen sizes, while PWAs handle the complete life cycle of a Web application. They ship with a manifest file linked from the HTML head that describes some of the properties, like an app icon, a splash screen image and a link to a *Service Worker*. The Service Worker is a managed cache developed in JavaScript, that is responsible for downloading resources like static HTML pages, JavaScript modules as well as style files. The worker intercepts calls from the application and either forwards the request to a Web server, or serves with a local version of the requested resource, saved in a local cache. The cached resource is not necessarily a static HTML or image file, but also dynamic API calls can be served by the cache. As the cache is highly application specific, the Service Worker implements an interface in JavaScript. Similar to HTTP Web caches, the worker may base its decision on the "max-age" HTTP attribute from the Web server. Often, the worker regularly probes the server whether a new version is available of a given resource. A common pattern is also to serve dynamic API requests immediately, and regularly querying the API server for the updated data that is then served on consecutive calls. As a consequence, only light-weight adaptations of the content of the Web application is performed on the client. One of the challenges of serving and updating PWAs is therefore the update policy, as the application is only refreshed on subsequent calls. Therefore, PWAs often ask the user to refresh the page manually. A Service Worker is also an endpoint for push notifications originating from the server. They make it possible for applications to notify the user when the HTML5 application is currently not loaded in a browser tab. Besides their use case in instant messaging, they are often used to keep up long-term user engagement by reminding users of the app, after not being in use for a certain period of time. Requirements Bazaar (cf. Section 3.2), for instance, is implemented as a PWA.

Over the last years, many approaches to make single-page Web applications available on mobile and desktop environments have been introduced. To the mobile end, these technologies are often related to cross-platform development environments like Cordoba [Apac19]. On desktop operating systems, a different approach is targeted. A common representative is Electron [Elec19], a node.js framework that ships Web applications together with an executable portable browser engine. An alternative is Lorca[11] that does not ship the browser engine as executable, but relies on an installed version.

The motivation that all these approaches have in common, is to reduce development overhead across platforms, and to reuse developer knowledge of JavaScript. As PWAs can be installed on the home screen, they reduce the overhead of users of having to start a Web browser and using the address bar to enter a URL, or even going through a search enginge to find a specific Web application [RoLe04]. Recently, the platform-independence reached the level that PWAs have found their way into the app store ecosystems of Microsoft and Google.

[11] https://github.com/zserge/lorca

5.7 Conclusion

In this chapter, we presented the applicability of the DevOpsUse life cycle model "on the edge". We call 'edge' everything that takes place at the periphery of the Internet, for example at organizational and administrative boundaries. We started with the presentation of the Layers Box, a community-driven hosting infrastructure of learning services on the premises of SMEs and networks of SMEs. It brings the discussed advantages of microservices closer to the edge by using a state-of-the-art containerization approach based on the Docker technology. As we were able to show, the use of the authentication and authorization solutions OpenID Connect and OAuth2 ensure a stabilization of the infrastructure regarding the essential user identification functionality. We discussed a user-facing application of low-latency peer-to-peer data streams across the edge, which is browser-to-browser video streaming. To this end, we presented OakStreaming, a library as developer support for implementing highly scalable video streaming applications in Web browsers. It significantly extends WebTorrent with properties to manage the data flow from a device, taking into consideration limiting factors such as network bandwidth and cost-benefit aspects of cellular data connections. With the integration of WebRTC into our widget-based distributed user interface framework, we then showed how we shaped the standardization of peer-to-peer protocols that allow two end user devices to communicate directly. Our primary use case was reducing the latency of user input across heterogeneous device types. We measured a significant reduction of the round-trip time from around 150 ms to 25 ms. As last prototype, we showed how we can integrate inherently edge-placed Internet of things devices in existing Web applications with the use of out-of-the-box Web standards and technologies. We essentially evaluated and validated the approach in an exhibition context. Finally, we placed the brand-new industrial practice of serverless computing and progressive Web applications into our research context and argued how DevOpsUse provides support for it.

What is now missing is an integrated analysis perspective that gives communities the power in their hands to show how effective their use of services and applications is. In particular, it should be able to effectively map the scenarios and architecture components discussed here. Thereby, a standardized deployment structure as presented in this chapter significantly helps towards a standardized monitoring solution, both on backend and frontend. The OakStreaming framework includes a monitoring component to measure the benefits of peer-to-peer architecture in the concrete use case. In the next chapter, we show how these and other log files can be used in a community-driven visual analytics framework.

Chapter 6

Visual Analytics for Community Self-Awareness

The ongoing digital transformation of workplaces, social interactions and entire value chains leads to a rapid increase in available data. Besides the opportunities regarding sense-making of the data, its amount leads to a number of hurdles. *Big Data* as research area deals with these problems that arise when tens of thousands to millions of data points are involved. One of Big Data's best-known models divides the area into the "4 V": *Volume* stands for the amount of data, *Variety* for its diversity, *Velocity* for the speed at which (new) data is created, and finally *Veracity* refers to meaningfulness and trustworthiness of data. Volume has an impact on various dimensions within information systems. First, it is a major challenge for systems to handle it. Since the amount of records produced can quickly exceed any inherent storage capacity, they are distributed within networks. These network infrastructures must be designed in order to be able to carry the records to data storage nodes, for short-term or long-term memory. These nodes, in turn, must have the capacity to handle this data. The next step is then sense-making from data to use it for process improvements, for example. This can happen in real time, or at some point in the future. In addition to the above-mentioned challenges, further ones include privacy, the format, semantics and the interpretation of data [LaJa12].

At the intersection of the Internet of Things, Web services and communication between individuals and devices, the aforementioned challenges are amplified. In the previous chapters, we have shown how the DevOpsUse life cycle integrates user requirement creation and information system development. The last chapter has then introduced some means to deal with the *volume* of data emerging from developed applications, by introducing new paradigms like edge computing and Web-based peer-to-peer communication streams between actors. However, we have not yet touched upon the *variety* of heterogeneous data in particular. The integration of different data sources and types make it particularly difficult to mine for regularities. To this end, computers have outstanding capabilities to record and transform data quantities in the shortest possible time. However, only the analytical abilities of humans make it possible to grasp connections at lightning speed. Precisely this combination of human and computer is the subject of the research area *Visual Analytics*. In this chapter, we therefore look at how communities of practice can prepare their data in a way that benefits them. Following the design science research methodology, we created a Web-based tool and ran through

several iterations. First, a general solution was developed. The visual analytics tool accesses the logging capabilities of an XMPP server to visualize data flows and interactions between clients. We then introduce *Social Web Environment for Visual Analytics* (SWEVA), a Web-based tool for visual analytics. Its technological foundation builds on the previous prototype and therefore shows the possibilities of the Web in terms of code reusability. It enables a model-driven visual flow design, which is then executed in real-time. The tool was developed with a special focus on usability, so that end users can find their way quickly, and work with experts at the same time and communicate using built-in communication tools. Following the spirit of continuous innovation, we are interested in how we can design a system in such a way that it is also prepared for future applications. For this, the visual flow is able to include various forms of data acquisition, processing and analysis. Before the tools are presented in detail, we show related work concerning visual analytics in the next section.

6.1 Visual Analytics

The term *visualization* generally refers to the visual representation of information to enable humans to quickly grasp and understand data. Visualizations build on the cognitive ability of humans to see quantitative and qualitative differences encoded in colors, shapes or structures. Typical visualizations are for example diagrams that any spreadsheet application can output from tabular data, like area charts, steam graphs and pie charts, just to name three. For example, a bar chart can easily visualize the number of requirements within each project of Requirements Bazaar. However, normally these diagrams offer no to little interaction possibilities, and lack the ability to do exploratory analysis with their fixed data representation. Visual analytics (VA) is the "science of analytical reasoning facilitated by interactive human-machine interfaces" [KAF*08]. It is a multidisciplinary approach that includes data science, data management, data analysis, human-computer interaction and decision support, among many others. The visual analytics process is depicted in Figure 6.1. It starts with data gathering and its optional transformation by removing outliers, scaling or other corrective measures. As a next step, a model is mined upon the data, where parameters are refined. At the same time, visualizations are built using the data, with graphical representations of the numbers. At the visualization stage, user interaction allows changing parts of the visualization in order to highlight certain facts, or to change other parameters. Ultimately, the process leads to knowledge gain. Based on the new knowledge, a feedback loop back to the data gathering part allows to adjust the input parameters by calibrating the selection of data sources.

Visual analytics facilitates the visual exploration of large data collections. In particular, visual analytics focusses on drawing conclusions out of complex data sets. Most systems that visualize data follow Shneiderman's mantra "overview first, filter and zoom, details on demand" [Shne96]. Visual analytics extends this mantra to "analyze first, show the important, zoom, filter and analyze further, details on demand" [KAF*08].

Up until some years ago, interactive visualizations were only feasible on software running on native platforms, with software such as RapidMiner and Knime. With the advent of Ajax (cf. Section 2.6) and other means of user interaction on the Web, most notably CSS3 and WebGL, new possibilities emerged in the browser to create appealing visualizations. Lightweight architectures running on microservices allow the provisioning of massive data on demand.

6.2 Related Work

In this section, we are focussing on related work in the area of Web-based visual analytics systems. We are particularly interested in solutions that make it easy for general audiences to create visual analytics pipelines following the visual analytics process model described above. This section is divided into three parts. First, we present all-purpose open source and commercial tools available for visualizing large data sets, coming from a data science background. We then show work that can be used to analyze Internet of Things systems. Lastly, we present related work that has been used in the area of learning analytics.

KNIME is an open source product for interactive analysis of data sets [BCD*09]. Originally developed for the pharmaceutical domain, it is now also used for customer relationship management and general business intelligence purposes. The application is JAVA-based and is developed as an Eclipse extension. It features a client-side user interface as well as a REST-based service backend that can be invoked by modules available on the frontend. A large variety of nodes and ready workflows are available to be integrated into custom graph-based analytics pipelines. *RapidMiner* is another software from the category of Desktop applications for large-scale data mining and machine learning. Its focus lies on designing workflows from data source to analytics, enabling the creation of predictive models. Similarly, the product *Tableau* offers a drag-and-drop interface to create custom analytics pipelines. They can then be connected in dashboards. One of its distinguishing features is geolocation-based analysis. The Gartner report on data science platforms presents further commercial and open source software for creating analytical models [IKBL19].

Besides the general-purpose analytics tools, there is commercial software for analyzing heterogeneous Internet of Things data. In particular, here we take a look at Web-based dashboards to make

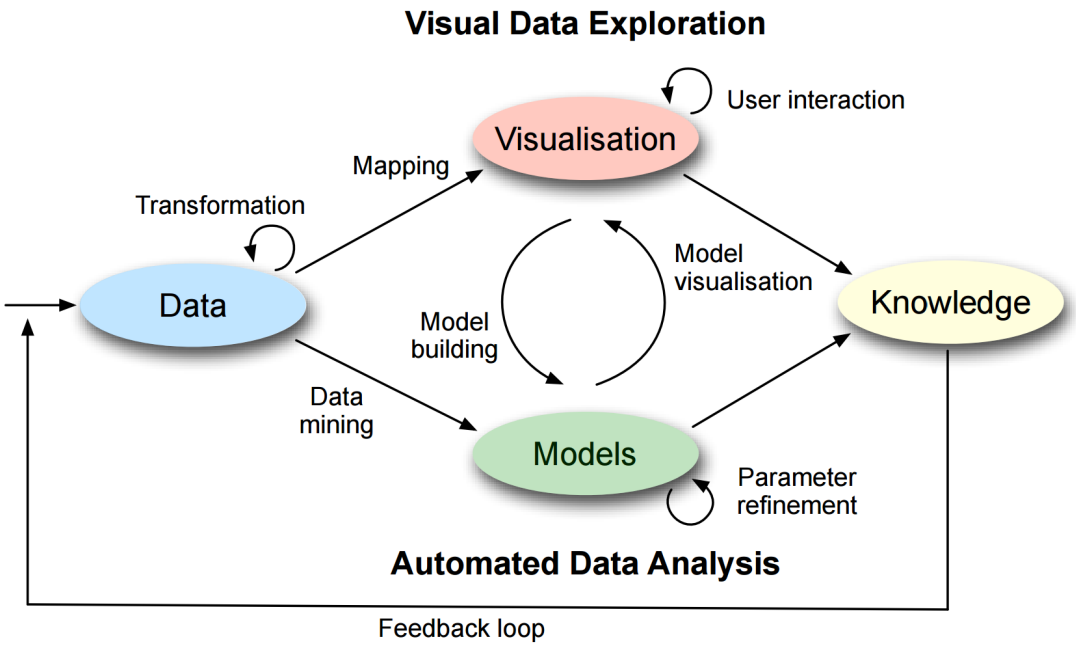


Figure 6.1: Visual Analytics Process [KAF*08]

device-specific data available for evaluation on the Web. *IBM Watson IoT Platform* is a commercial visual analytics platform for the Internet of Things [IBM17]. The Web application offers IoT analytics features for device management and analytics applications. Devices can be configured to send data into the cloud using the MQTT protocol. Applications can then interact with the data. Finally, collected data can be visualized in a configurable dashboard that provides location, live property values as well as alerts caused by user-defined rules. Bosch provides their own IoT solution called the *Bosch IoT Suite* [Bosc17]. Using the provided development toolbox, users can create their own IoT applications. The platform offers an IoT Hub component that transports IoT data from sensors to the applications developed by the customer. *Pheme* is a cloud-based service that repurposes Web analytics services for IoT data collection and visualization [MCJ*15]. The term *Web analytics* usually refers to the collection and evaluation of data that users produce when visiting websites. In their work, Mikusz et al. map typical Web analytics properties to IoT events. Pheme consists of four different modules: import, preprocessing, visualization and reporting.

Navigator for OSS Evolution (NOSE) is an analytics tool for community-oriented statistics of open source software communities [HLKl14]. It covers data about the in- and outflow of members in project mailing lists, the number of code commits, the size of the core and a social network graph. Additionally, some data mining algorithms are run on top of the data, like sentiment analysis. It mainly focuses on the visualization and offers little interaction possibilities, thus it cannot be considered a fully-fledged visual analytics tool in the narrow sense. NOSE supports iterative refinement, however it is bound to individual tools and therefore lacks extensibility. As possible future work, the authors mention adding more data sources.

The MobSOS Query Visualization Service is an application developed to visualize MobSOS usage and survey data on the Web to analyze and evaluate service success models [KRLJ16, DEKl12c]. It allows analysts the exploration of data stored in an SQL database, by exposing query results over a REST-based interface and showing pie-, bar- and linecharts in the browser, among other visualizations. On the one hand, the SQL queries allow a high flexibility of the data input. On the other hand, the requirement of understanding (complex) SQL queries in order to grasp what kind of data it requests hampers the adoption of the application by all community members, e.g. end users or designers. User interface widgets with predefined queries can be generated, shared and embedded into other websites. However, they are mostly static as far as the interaction possibilities are concerned. The selection of visualization possibilities is rather generic and does not focus on specific domains. The displayed data is directly taken from the database query result without additional processing like filtering or outlier detection. Therefore, the MobSOS Query Visualization Service is quite limited in regard to the visual analytics process described above.

Dynamic Visual Topic Analytics (D-Vita) is an application for creating dynamic topic models for academic publications and visualizing them interactively [DGKl13]. Topic modeling is a statistical means to discover the 'topics' of a given set of documents. In D-Vita, topic models are created by visually arranging tools in a toolchain using a simple editor and parameterizing values. The toolchain starts with crawling the data corpus for the desired information, then processes and analyzes it. Finally, interactive visualizations are generated. They show, how topic distributions develop in academic publications over time. Even though the process is very flexible and extensible, it relies on tools running locally inside a workflow engine. The tool parameters are predefined for the given domain. Visualizations are limited to time rivers and pie charts, as D-VITA focuses on topic models. Therefore, it also can not be considered a generally applicable visual analytics tool.

6.3 Visual Analytics of IoT Systems

Dealing with data from Internet of Things system brings manifold challenges. As mentioned in the introduction of this chapter, the number for example of sensors available poses new challenges like the volume of data. For the communication, current implementations of sensor networks struggle with a myriad of standard on different network layers. Often, vendor-specific transmission channels and interfaces limit access to the data. For instance, protocols for message-oriented middleware in the IoT include *Message Queuing Telemetry Transport* (MQTT), *Advanced Message Queuing Protocol* (AMQP), *Extensible Messaging and Presence Protocol* (XMPP), *Constrained Application Protocol* (CoAP), among many others. On the lower protocol layers, we find *Bluetooth* and *ZigBee* in many of today's rolled out systems. However, we can observe a convergence towards open Web protocols to make the data available. As a consequence, although the actual machine-to-machine communication takes place via proprietary protocols, most commercial standard solutions are equipped with a gateway that translates device-specific communication channels into the open HTTP web standard so that they can be called by apps on smartphones. This is the main idea behind our framework for visual analytics of IoT systems.

In a first step, we created a framework for near real-time monitoring for an XMPP network. It provides users with the ability to monitor and analyze traffic within a federated XMPP network in near real-time. In this context, near real-time refers to the characteristic of the server, that communication is not permanently stored for later analysis, but instead is being processed immediately, typically within a time frame of a couple of seconds. The main goal is to allow managers of the network to get a quick overview of the state through several visualizations. Another requirement of the system was to develop a minimally invasive method of accessing the raw data through state-of-the-art XMPP server implementations.

6.3.1 XMPP Analytics Architecture

Figure 6.2 shows a conceptual overview of the developed XMPP analytics system. To achieve the goal, we leverage the existing structure of XMPP networks, and deliver log messages using existing connections between servers and clients. There are two variants of the logging plugin (yellow background in the figure). One is installed on XMPP clients (in this case the *Logging Client* on top left). The other is handling XMPP servers (the *Logging Server* on top center). The *Analytics Clients* on the right can either be directly connected to an XMPP server, or to other clients via a collaborative space in a ROLE space. The ROLE space is also where the user interface widgets are served from. XMPP clients who are not equipped with a logging plugin can also be observed if their communication partner is logging itself. Therefore, the system is able to log messages sent between two XMPP clients, if at least one of the clients is logging.

The server component handles the communication between logging clients through a new XMPP extension protocol. The log itself describes characteristics, e.g. how many clients are connected and how many messages are sent between which clients. Servers additionally report statistics about their current memory load and CPU usage. We decided to add a plugin to clients as well, in order to be able to log peer-to-peer messages directly sent between clients without an intermediary server. The logging XMPP servers may also store log data in a database which in turn is accessed by

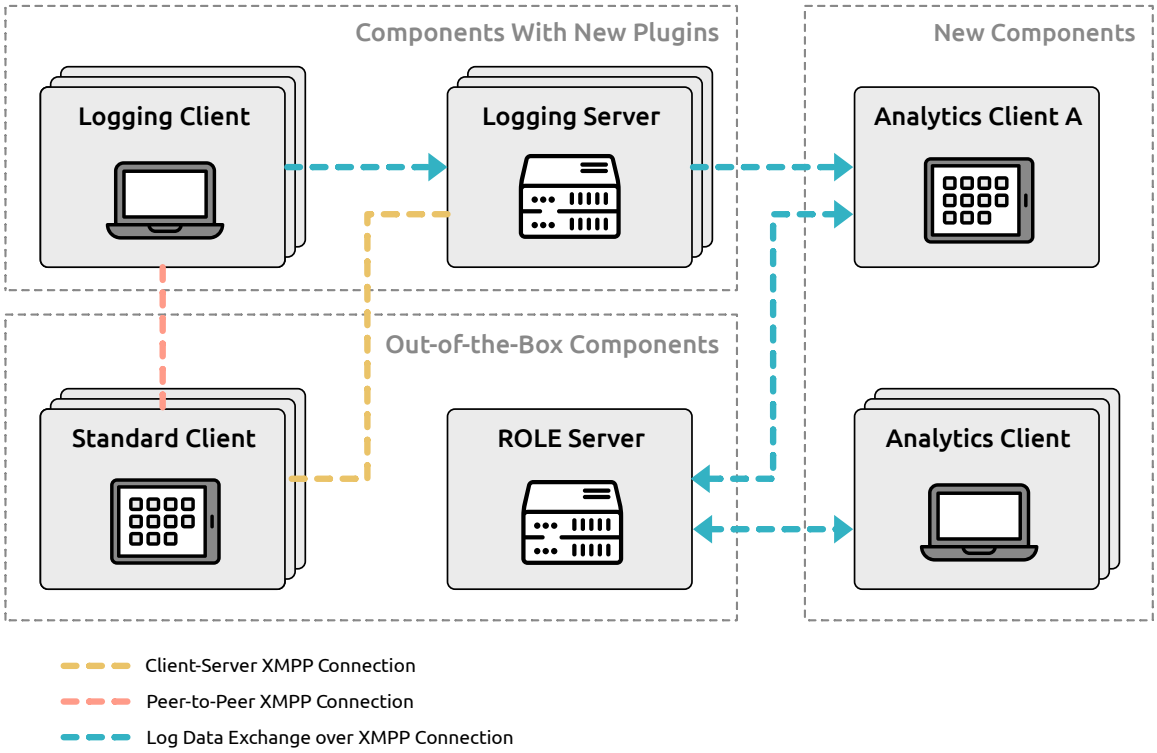


Figure 6.2: XMPP Network with Logging Plugins and Analytics Clients (adapted from [Guth14])

the MobSOS query visualization service, however in the first iteration we focus on near real-time aspects of the system.

Figure 6.3 shows a screenshot of the Analytics Client implementation. The application runs within a dedicated analytics ROLE space. On the left, we see the configuration widget for selecting the logging server. In the middle, there are statistics about the current memory and processor load of the server. On the right, we see a graph visualization of all the connected clients. The thicker a node, the more messages were sent from or to it. Interaction possibilities in this prototype are limited; they include selecting the number of nodes by a factor determined by its importance in the network, which in turn is calculated by the number of messages sent or received.

6.3.2 Evaluation of the IoT Analytics Prototype

To validate the feasibility and federation capabilities of the solution described above, we performed preliminary technical and usability evaluations of our system. For reproducibility reasons, we used an IoT dataset from hurricane Katrina [PHSh10], one of the most severe natural disasters in the history of the United States. It was developed within *ACDSense*, an inter-organizational sensor network scenario spanning our three universities: RWTH Aachen University, TU Dresden, and BTU Cottbus-Senftenberg [SGR*14]. The dataset contains multiple thousand measurements of several weather stations. In our scenario, the data was replayed through an XMPP server, and our visual analytics frontend was connected to the XMPP server. For up to 30 nodes, the near real-time

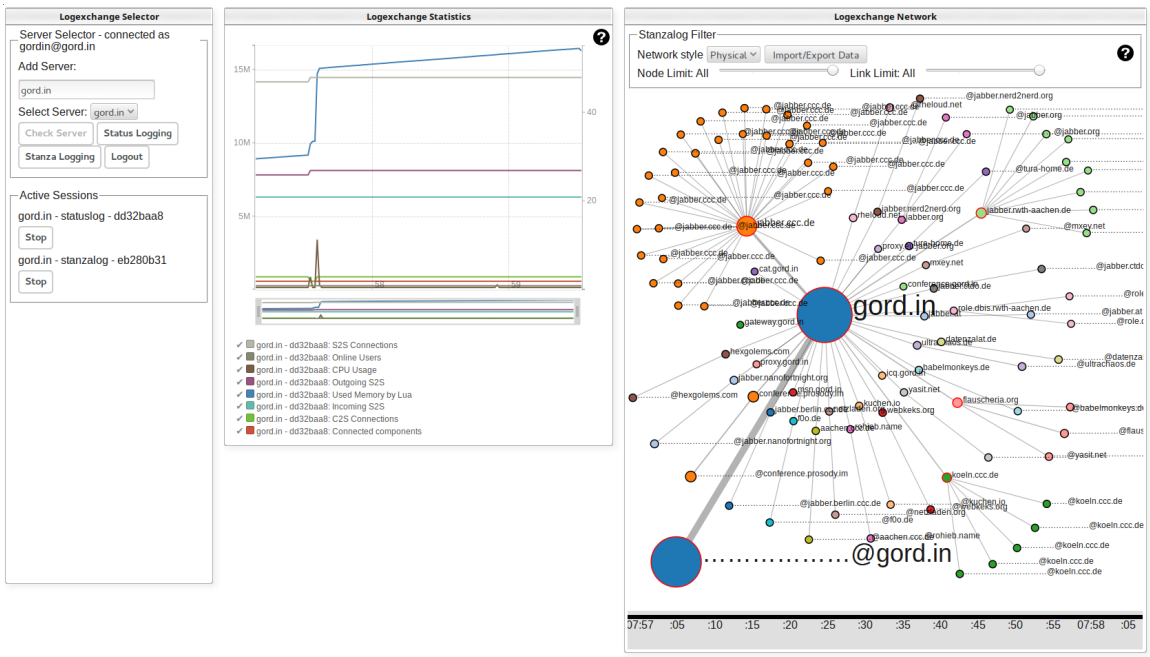


Figure 6.3: Screenshot of Browser-Based XMPP Analytics Application [Guth14]

graph widget rendered the graph at a speed of around 55 frames per second. When adding more than 30 nodes, the visualization began to slow down. For around 70 clients, the frame rate dropped to around 30 frames per second. When monitoring a network with 110 clients, we still measured 20 frames per second.

We additionally invited 12 volunteers out of our pool of bachelor and master students of computer science and performed a usability study. In total, we held six evaluation sessions with two participants in each. The participants were asked to collaboratively generate near real-time visualizations using the IoT dataset described above. We provided two laptops running on Windows 10, with recent Chrome browsers. A third laptop hosted the frontend and backend services as well as the evaluation network simulation. After the modeling of the data pipeline, the participants were asked to identify certain nodes in the analytics. For that, they had to interact with the visualization. Finally, the users had to fill in a survey. Although most participants knew about the Internet of Things paradigm, only few were familiar with the details of IoT protocols and visual analytics. All users agreed that for the given analytics tasks, extracting the information via the graphs was efficient and easily comprehensible. Furthermore, the availability of near real-time visualizations was helpful in understanding the inner working of the IoT network. Minor usability issues detected in our tests like finding the right buttons could be solved by changing the icon and offering tooltips. Overall, the preliminary evaluation showed the usefulness particularly in scenarios where a large set of data is available.

Table 6.1: Comparison of Visual Analytics Tools

System Property	Framework						
	NOSE Dashboard [HLK114]	MobSOS Query Visualization	D-VITA [DGK113]	IBM Watson Analytics [RVP*14]	GitHub Visualization Tool	Black Duck Open Hub	**SWEVA**
Usable by both end users and developers	●	○	◐	○	●	◐	●
Interactive visual analytics	○	○	●	●	○	●	●
Near Real-Time collaboration	○	○	○	○	○	○	●
Continuous refinement of processing models	○	◐	●	○	○	○	●
Flexible integration of existing tools	○	○	◐	○	○	○	●
Sharing of visualizations and results	●	●	●	●	●	●	●

● = provides property; ◐ = partially provides property; ○ = does not provide property

6.4 SWEVA: A Social Web Environment for Visual Analytics

The related work discussed in Section 6.2 shows diverse use cases. In general, however, they demonstrate the uptake and acceptance of visual analytics tools by communities of practice. What becomes evident in the research survey is the lack of tools that are generally applicable, relying on heterogeneous data sources, community-aware and embeddable into third-party websites. With respect to the applicability to learning analytics, we miss approaches integrating social network analysis methods like (overlapping) community detection and expert identification. This complex interplay of technology appropriation, social factors and community learning analytics is coming with a set of heterogeneous requirements that are not trivial to fulfill. In the following, we present our conceptual architecture and implementation to fill these gaps. We start with the challenging area of integrating heterogeneous data sources of the Internet of Things in a Web-based tool. We then extend the conceptual scope to general visual analytics of arbitrary data sources. Finally, SWEVA is presented as collaborative Web tool for visual workflow modeling. The system was developed in a master thesis by Ruppert [Rupp16].

In the previous section, we presented the XMPP logging system that is able to create simple near real-time visualizations of an XMPP network. Besides the limited interactivity, the prototype specifically focuses on XMPP networks making it not useful for other protocols. Even though XMPP has been successfully used and implemented in various use cases spanning from world-wide instant messaging systems to Internet of Things networks, it only represents one of the available data exchange standards. Also, there are many more types of interactions in socio-technical system development.

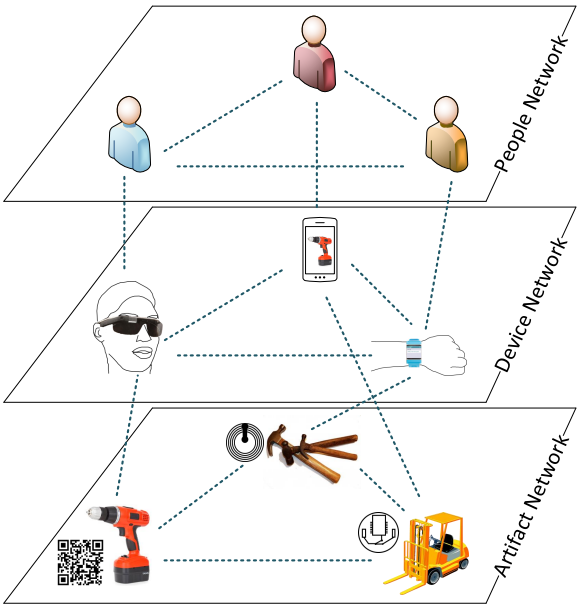


Figure 6.4: Actor-Network of Interacting with Physical Artifacts

Figure 6.4 takes into account other dimensions and therefore additionally represents socio-technical interaction patterns. It deals with the use case of learning at the workplace, here on the construction site in particular. There, working with physical artifacts allows analysis of various interactions: (a) device-to-device for the ambient intelligence; (b) device-to-user with the wearable technologies; (c) user-to-user for community interactions. Thus, we are interested in a system that is able to visualize data coming from and flowing between all these dimensions. A Web-based solution is ideally suited for this requirement, as Web browsers are universally available on different device types. The Social Web Environment for Visual Analytics (SWEVA) is a conceptual approach and prototypical implementation of a service able to retrieve, process, and visualize data from heterogeneous data sources such as machine data or wearable sensors. It is extensible on various levels to handle a wide variety of current and future protocols for data exchange between the Internet of Things. Within the framework, advanced social network analysis processing can be accessed. Due to its Web-based nature, users can open the application in any browser to take part in their communities' analytical undertakings.

The main building block and reasoning behind our approach is to leverage open Web technologies over the whole chain from accessing device data to analyzing it. Our system collects data from heterogeneous sources through Web technologies and makes it available via a browser-based platform that visualizes the data in near real-time. The complete pipeline and its layers are shown in Figure 6.5 from top to bottom in chronological order; the image does not represent the iterative refinement process that SWEVA enables. The platform is extensible in terms of more data sources and visualization options.

As shown in Figure 6.6, the pipeline starts at the data source level. Data sources can be anything from real Industry 4.0 machines, body-worn wearable sensors or input captured on smartphones. Besides, data can also be retrieved from any website, such as open data provided by governmental

and non-governmental organizations. The actual data format like JSON or XML is not yet important in this step. The next level is the data aggregation tier. Here, data that is typically not available over the Internet is made available on repositories. For instance, sensors that are using heterogeneous exchange protocols may upload their recordings to a common IoT database. This step also comprises 3rd party services, such as sentiment detection of discussion forums or map providers. After the first steps that make the data available on the Web, the collaborative model editor comes into play. The editor is explained in detail in the next section. For running the modeled data processing pipeline, we developed the *Core Framework* which is explained in Section 6.4.2. Finally, the processed data is loaded into the collaborative data visualization engine that is described in Section 6.4.3.

Figure 6.7 maps the parts of the visual analytics process as shown before with the responsibilities of the SWEVA framework which are presented in the following. The *Collaborative Modeling Tool* and the Core Framework are responsible for the data gathering, transformation and mining. Models are built within the modeling tool, before they are taken up by the Core Framework. The *Collaborative Visualization Tool* is responsible for the visual data exploration, in particular visualization and knowledge gathering. Through collaboration within the community, models are built and refined. Similarly, interaction is performed collaboratively by users. Together, the system parts comprise the SWEVA framework.

Figure 6.8 shows a conceptual overview of the system. On the left side, multiple Internet of Things sensors deployed in Industry 4.0 machines and wearable devices provide input to an asynchronous message queue. In the center, we see the real-time message queue, which routes messages to both a database for long-term memory of historical data, and to devices interested in showing the data in visual analytics charts On the right, the output visual analytics chart is shown that can be modified by user input through input text boxes, sliders and checkboxes.

6.4.1 Modeling Data Processing Pipelines

The Collaborative Modeling Tool is used for the creation and refinement of data processing models. Communities are enabled to collaboratively create their own visual analytics tools based on their

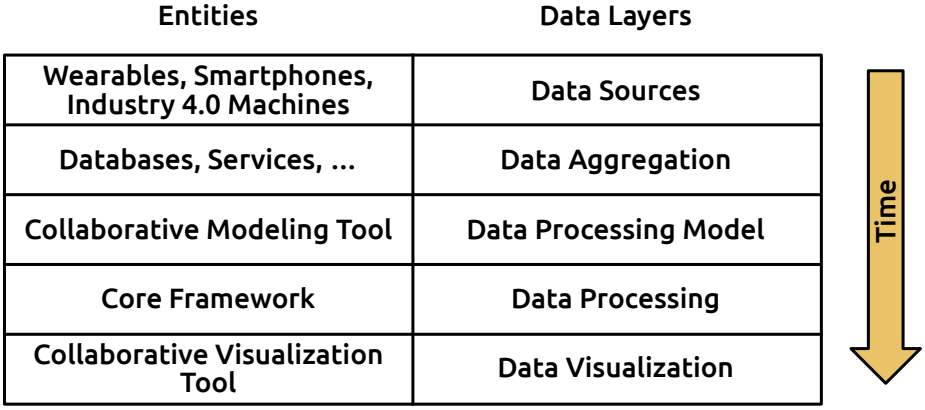


Figure 6.5: Layered Abstraction of Visualization Creation (adapted from [Rupp16])

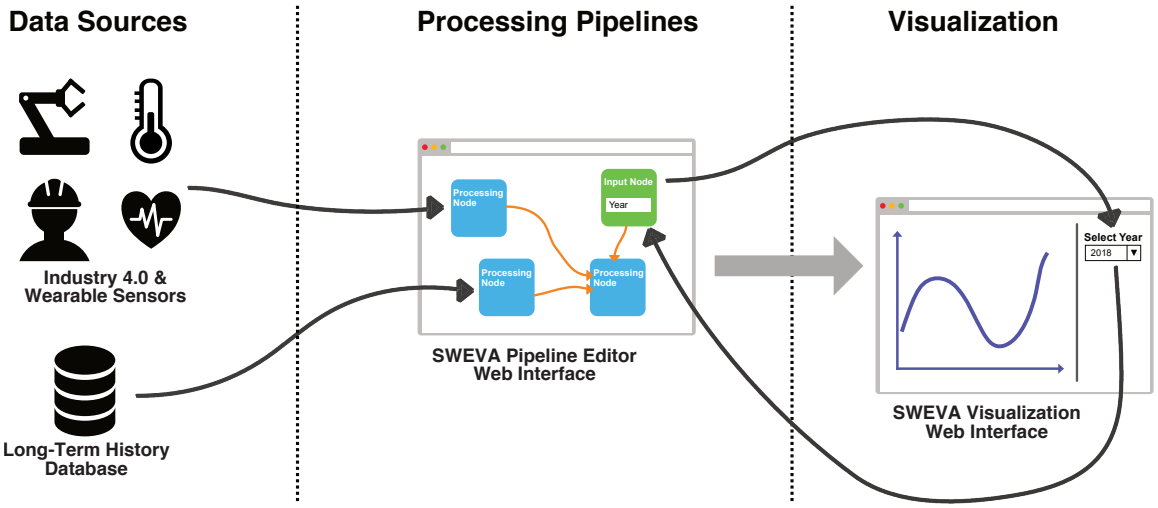


Figure 6.6: Conceptual Schema of Processing Pipeline

current needs. The intent of the collaboration is to allow even end users without development experience to actively contribute their ideas for visualizations, by being able to describe their wishes to (remote) collaborators. Processing models are visualized in an easy-to-understand way, hiding unnecessary details until they are required, following the visual analytics mantra described above. These models define the processing steps from raw data to highly interactive visualizations. They are represented by a graph consisting of nodes, and edges connecting the nodes. We define the

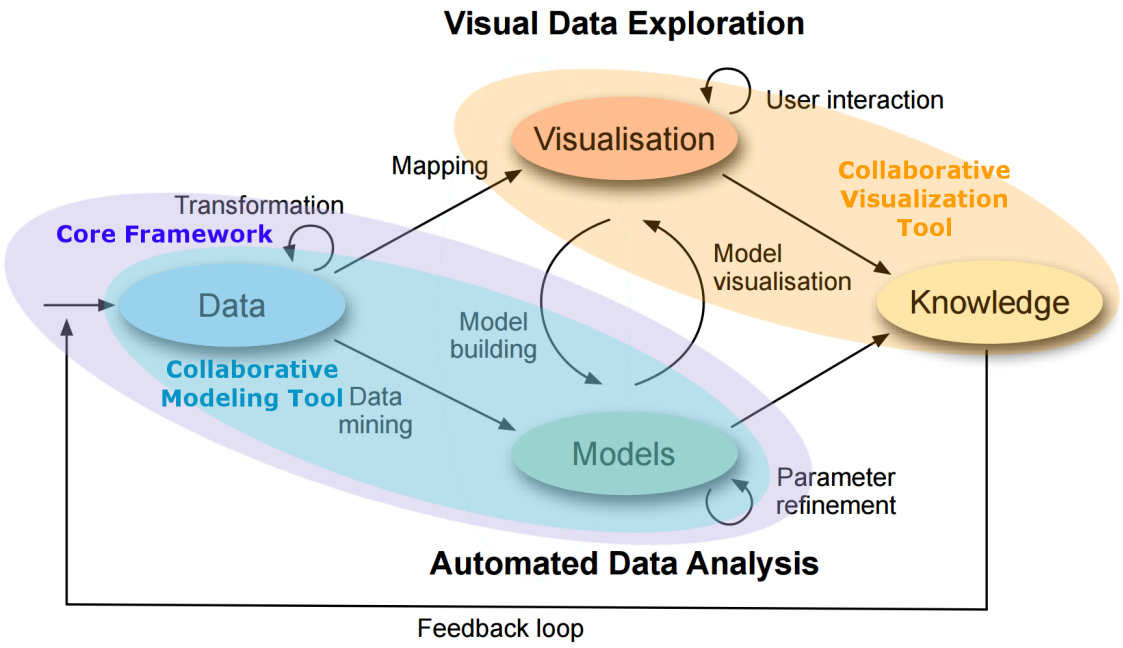


Figure 6.7: Visual Analytics with SWEVA [Rupp16], based on [KKEM10]

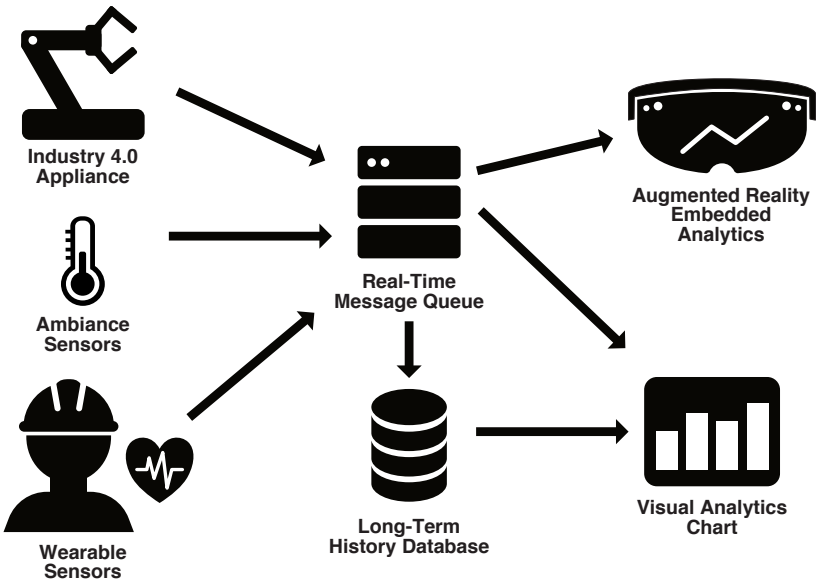


Figure 6.8: Conceptual View of SWEVA

processing model as a directed acyclic graph (DAG), where the nodes represent service calls and computations, and the edges the flow of data between them. If an edge is present from node A to node B, it means that data is processed first at A and B is fed with the output of A. We decided for a DAG for a number of reasons. First, as DAGs are cycle-free, we do not allow any loops in the flow. While this limits the expressiveness by forfeiting Turing completeness, we see as benefit, that we can decide the halting problem under the assumption that all service calls and local computations inside the nodes halt. Additionally, there are no infinite loops within the processing model itself. Second, the DAGs lead to at least one topological ordering, which means for a graph $G = (V, E)$ with vertices V and edges E: $\forall(u, v) \in E$ with $u, v \in V$ holds $u < v$. A topological ordering can be found in $O(|V| + |E|)$ for example by Kahn's algorithm [Kahn62]. As the edges describe dependencies between service calls, a topological ordering shows an order in which they can be run without conflicts. On the practical side, this leads to an error confinement to individual nodes, not the entire processing mode. An advantage is that the tool is able to highlight the broken node to enable quick troubleshooting. Third, the absence of loops can also be justified by an increase in performance, as many iterations over remote services can lead to a communication overhead.

There are two types of nodes available in the collaborative model editor; first, data processing nodes and second, input nodes. A data processing node itself stands for an arrangement of 6 layers:

1. **Data Parameters:** Every node can receive input from none, one or multiple other nodes.
2. **User Parameters:** User-defined parameters are coming from input nodes. They represent the interactivity in the visual analytics process.
3. **Request Creation:** In this layer, data and user parameters are merged and a request to a service is prepared.

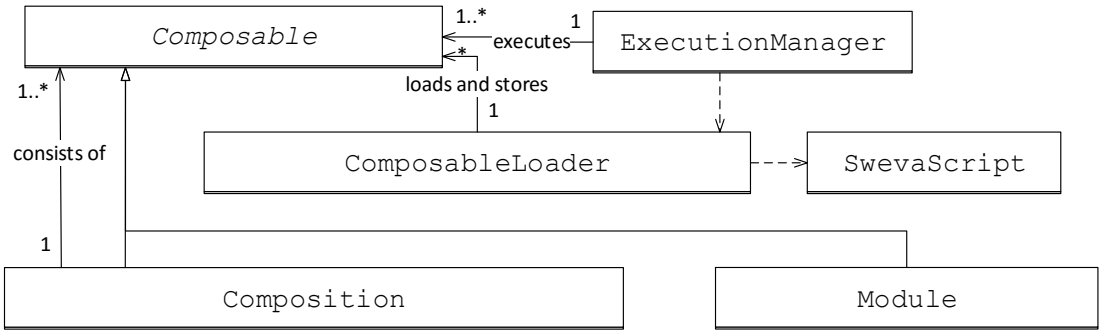


Figure 6.9: UML Diagram of the SWEVA Core Framework [Rupp16]

4. **Request Sending**: Here, the request to the server is executed, for instance with an HTTPs fetch to a service.

5. **Response Transformation:** Depending on the output of the service, the response may be transformed in another representation. For instance, a transformation from XML to JSON formats can be achieved here.

6. **Results:** The output of a node can be connected to other nodes as input. The last node, depending on the topological ordering of the graph, outputs the overall data that is used as input for the visualization.

Before sending a request or receiving a response, custom user-definable scripts can be run to enable encapsulation of the service call, independent of its concrete data format requirements. Alternatively, the layers 3-5 can be replaced by a local computation script without calling any services. Examples for this functionality are simple rounding or summation operations. Multiple modules can be grouped into compositions. These compositions may be represented by a node in the graph, so that processing nodes and compositions can be used interchangeably.

Finally, the tool outputs the model and makes it available to the Core Framework. The model can also be created in a text editor, as it is written in the human-readable JSON format.

6.4.2 Running Processing Pipelines on the Edge

The Core Framework is responsible for collecting data input and to call each execution in the given order as defined in the acyclic graph outputted by the model editor. When a model is executed for the first time, the default values of the user input nodes are collected and then inputted into the processing nodes which are consecutively run based on the total ordering. Multiple modules within a model can be executed concurrently.

SWEVA is an edge-oriented visual analytics framework. For this reason, by default, the processing pipelines are run on the accessing client. This decentralized structure has no single point of failure and is thus better scalable compared to a server-centric application, as the users provide the

necessary hardware power themselves. However, in cases, where users are accessing SWEVA with a resource-constrained mobile device like an augmented reality headset, there may not be enough processing power on the client to run complex processing pipelines. For this reason, we developed the *Execution Service*. It is a server-side execution service with its feature being on pair with the client-side execution engine. A sessionless REST-based approach reduces session management overhead by offering an interface where Composables are uploaded. As result, the API delivers the calculated data.

Figure 6.9 shows the UML class diagram of the SWEVA process graph. It shows, that multiple Modules can be grouped into Compositions. As they have the same interface as Modules, they can receive parameters and output data in the same way, making them interchangeably usable in the graph. The overarching class is a Composable. It allows the combination of simpler elements into more complex ones. An example for a combination would be the following composition of three modules for sentiment analysis of a text in a foreign language. The first node could be a Composable itself, applying text mining algorithms for detecting the language. The second node would then translate the language using a Web service call. Finally, the third Module would call a language-specific service for performing the actual sentiment analysis.

As an alternative to the custom graph-based processing model, we also investigated the usage of a Business Process Execution Language (BPEL) engine for service orchestration [OASI07]. BPEL is an XML-based standard for the orchestration of Web services. It uses Web Services Description Language (WSDL) interface descriptions for the creation and interpretation of service messages. Although BPEL is widely used in the professional realm, we found its target too specific for our use case. For instance, it heavily relies on XML as data exchange format, while today's Web services are often REST-based. Additionally, it focuses on server-side execution of the process models. This comes in impractical for small computations as they are possible in SWEVA modules. Even though with Oracle's BPEL extension bpelx:exec Java code can be written, it is not compatible with a client-side JavaScript rendering. Because of these shortcomings, we decided for the custom approach described above. As last step, the Core Framework makes the final calculation available to the visualization.

6.4.3 Component-Based Interactive Visualizations

The final component in the SWEVA framework is the interactive visualization tool. Besides visualization and interaction with the gained visual analytics model, it is responsible for the knowledge gain by converting raw data into a humanly comprehensible format. The created diagrams consist of various shapes and colors. To be as general as possible, it must support theoretically infinite visualization techniques. Since the tool cannot provide this from the ground up, it meets this demand with a modular architecture. Visualizations are loaded on demand, as soon as the configuration, supplied together with the processing model, is loaded. Hereby, we leverage the Web as distribution platform.

Besides the graphical output, the visualization tool also provides a user input form, as defined by the processing model. Depending on whether the input node is configured as text input or as dropdown, appropriate user interface widgets are rendered next to the visualization. Any changes here are propagated to the Core Framework, which then reprocesses the graph with the updated

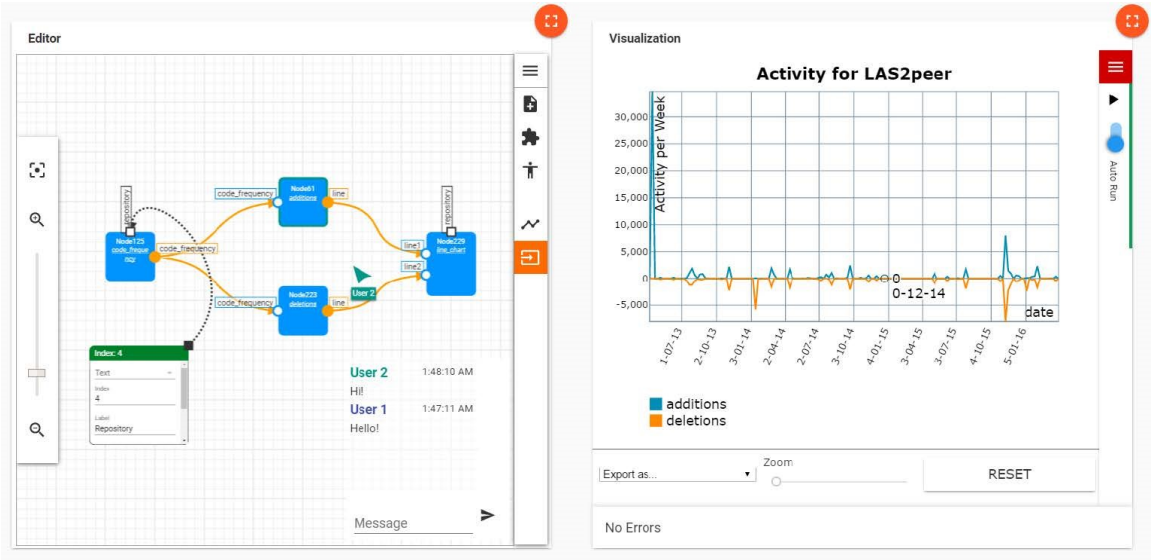


Figure 6.10: Screenshot of the SWEVA Collaborative Visual Analytics Tool

inputs and delivers the result to the visualization tool, which in turn renders it.

In the following, we list some recommendations for developers of new visualization widgets:

(a) **Information overview:** Following the visual analytics process of Figure 6.1, visualizations should start with an overview, and allow "zooming in" for further details.

(b) **Predictions and highlights:** The human eye should be lead to interesting aspects of the data.

(c) **Search and filter:** There should be the possibility to restrict the visualization to certain, user-selected aspects.

(d) **Export:** To enable the use of the gained visualization outside the SWEVA application, e.g. for reports, there should be an export possibility.

(e) **Reusability:** We encourage general visualization modules which are configurable to adapt them from different contexts.

In the next section, we explain the prototypical implementation of the whole SWEVA Web-based system for visual analytics.

6.4.4 Web-Based Implementation

In this section, we discuss the implementation of the SWEVA framework. It is divided into a front- and backend, both are developed using component-based software engineering. On the user-facing side, we developed a single-page Web application with state-of-the-art Web components. Figure 6.10 shows a screenshot of the whole Web application. The particular instance portrayed in the figure displays the processing model for retrieving development data from an open source

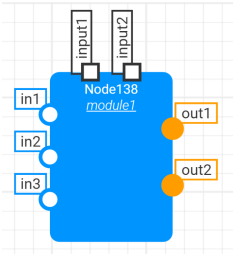


Figure 6.11: SWEVA Processing Node

repository on GitHub. Specifically, the model first retrieves raw data in JSON format from a public Web service. The dataset contains dates as index and the number of lines added or removed as values. This dataset is then separated into additions and deletions via two processing nodes. Finally, the data is fed into a line chart.

The collaborative model editor consists of a model viewer and editor. It is a tool for creating and editing visualization pipeline models that define the processing steps from raw data to highly interactive visualizations. The editor was developed using the open source *jsPlumb Toolkit*[1]. The jsPlumb Toolkit enables developing graph-based modeling applications for Web browsers. On the left side, we added zoom controls to be able to look into details of the model. On the right side, a toolbar gives access to various model-related functionalities, like adding input nodes or data processing nodes. To simplify reuse, we added a repository of predefined data processing nodes which can be browsed through in the interface. The visualization types, e.g. line chart, stacked area chart or bar chart, can be selected in the toolbar as well. Finally, the update model button generates the data visualization pipeline as JSON document and hands it over to the Core Framework.

Figure 6.11 shows a graphical representation of a processing node as implemented in the Web frontend. On the top, the connection points for user inputs can be seen. On the left, the circles represent input links, while the output can be shown on the right. They represent the screws to be adjusted during the visualization. An example is filtering which data is shown in a visualization. The data pipeline remains open for continuous refinement during the visual analytics process. Created models can be grouped into compositions and exported for later use. That way, recurring, complex data transformations only need to be modeled once; later, they can be imported into other visualization pipelines.

The Core Framework is responsible for running the data retrieval and trans- formation pipeline. It is developed in JavaScript and can thus run in browser or server environments, the latter using a NodeJS instance. We also developed an execution service to be able to run it within the Java-based peer-to-peer microservice framework las2peer [KRLJ16]. It is applicable for long-running data retrieval and transformation operations, that would require too much processing power on the client. After the Core Framework has generated an output JSON file, it is handed over to the visualization tool.

The visualization tool consists of a viewer where various charts can be loaded and displayed. It inherently supports zooming and panning operations; besides, the diagrams it loads may offer further functionalities like expanding or collapsing a tree structure or limit the displayed key space

[1] https://jsplumbtoolkit.com/

to a certain time period. Similar to the model editor, the right side features a toolbar. It contains the controls defined as user inputs. Currently, we support text, number, numerical slider, toggle, dropdown and fixed value inputs. The inputs are automatically validated according to their type. For instance, user inputs to numerical fields are checked whether they represent numbers; if not, they are not processed. In case of such a validation error or other malfunctions during running the pipeline, an error is displayed in a logging pane at the bottom of the screen.

As stated above, both frontend parts are developed using Web components. One of the main advantages are their reusability. After importing and declaring them in HTML, they can be used as normal elements on the page. For example, we are able to import visualizations into third-party WordPress pages by a simple HTML `import` statement and the use of the custom tag `<sweva-visualization-container>`. By using the Polymer library from Google for user interface widgets, we were able to get accessibility out-of-the-box. Also, the user interface adheres to the cross-browser Material Design guidelines [Goog18]. This enabled us to focus on functionalities, rather than browser and platform quirks.

A particular focus of the framework lies on its collaboration capability. This allows different community members to work simultaneously on models and visualizations to support each other. Concrete awareness features include a mouse shadow and a chat. The mouse shadow displays the current mouse pointer position on all connected instances of SWEVA. We color-coded the shadow by user. In both our modeling and visualization frontend components, we added a simple textual chat to further support cooperation between remote users. Technically, the synchronization is achieved via the CRDT JavaScript library Yjs [Jahn19].

6.4.5 SWEVA Evaluation

To evaluate the SWEVA prototype, we performed a technical evaluation as well as a moderated user session with volunteers for assessing the usability and acceptance of our tools. Additionally, a set of example visualizations was created, visualizing the collaboration of software development communities.

For the usability evaluation, we invited 14 software developers out of the student pool at our university. Most of them had software development experience, but we also had users without any programming knowledge. They were divided into four one-hour meetings, three of which were attended by three people and the last by five. In order not to make the findings dependent on special previous skills of the participants, we opted for an evaluation scenario that is neither in software development nor in learning. Instead, the participants were asked to create their own dataset by rating well-known movies; we are confident that the abstraction given by the movies is reasonable for our evaluation goals and the limitations of the study. In principle, these movies could be replaced by any other item, like software bug reports or learning content. To create the dataset, we developed a simple frontend that presented the participants with a title, a poster and a short plot summary for each movie. The users could then individually and per-movie choose among four different rating methods from a dropdown menu: a slider with values between 0 and 10, a binary rating of *good* and *bad*, a textual input field and and a random rating between 0 and 10 (the computer would rate then). After the rating procedure, the second part of the evaluation asked the participants to visualize the rating results as bar chart with the help of SWEVA. The first subtask produced a very simple bar

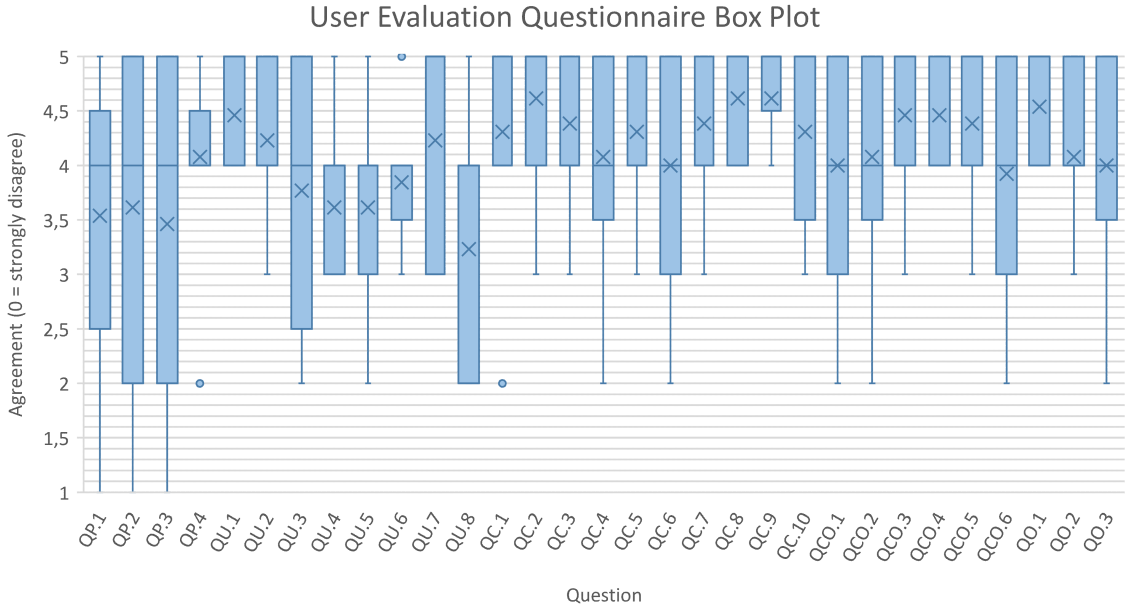


Figure 6.12: SWEVA User Study Results [Rupp16]

chart showing the relative distribution of the four rating methods. Then, the users extended the data processing model with additional information and statistics. Afterwards, they were asked to openly discuss what rating method they would prefer when building a custom movie-rating Web page. The third part involved a data processing model that visualized the ratings as a graph. Each movie and participant was represented as a node, connected to each participant that rated that movie. The usability study participants then had to manually adjust the SWEVA processing model code to add a slider controlling the minimum rating required to connect a movie with a voter.

After the session, the participants were asked to fill out a questionnaire consisting of 31 five-level Likert-scale questions and two open questions. The results are shown in Figure 6.12 with their corresponding question identifiers that are discussed in the following. The first part of the questionnaire (QP.1 - QP.4) asked about demographic data. Three participants considered themselves as non-developers (QP.1 rated lower than 3). Almost all participants with development experience considered themselves as familiar with Web development (QP.2 rated 3 and higher). Eight participants considered themselves as members of software development communities (QP.3 rated 3 and higher). Almost all participants were open to collaborative problem solving (QP.4 rated 3 and higher). The second part (QU.1-QU.8) asked about tool usability following principles described by the norm EN ISO 9241-1102. With QU.1 averaging at 4.5 most participants agree that the tools offer the necessary functionality to complete their tasks and can be used efficiently (QU.2 averaging at 4.2). The provided help and instructions from the tools, however, was perceived as lacking. The same is valid for the ability to learn quickly, as the overall ease asked through QU.3-QU.6 had average ratings of 3.6 to 3.8. Users recommended more tooltips, guides and demo videos. Customizability was perceived as sufficient (QU.7 at 4.2 in average), while the error feedback should be improved (QU.8 average at 3.2). The third part of the questionnaire (QC.1-QC.10) asked about the general SWEVA platform. With one exception, all participants understood that the graph mod-

eled the visualization (QC.1 rated 3 and higher). They also agreed that visual modeling has benefits compared to a purely textual representation (QC.2 at average 4.6). The possibility to reuse nodes of a graph in other graphs was understood (QC.3 at average 4.4), and the data flow was perceived as easy to read (QC.4-QC.5 at average 4.1-4.3). Based on these ratings we are confident that the directed acyclic graph is a valid modeling approach to control the data flow. The concept of continuous refinement did not appear to be entirely clear to our participants (QC.6 at average 4.0). With QC.7 at average 4.4, the idea to use visualization as decision support instrument was well accepted. The same is true for visualizing community data for self-reflection (QC.8 at average 4.6) and the statement, that a better understanding of community needs can help community processes (QC.9 at average 4.6). Most participants are interested in knowing more about the communities they are involved in (QC.10 at average 4.3). The concept of improving community self-reflection via visual analytics seems to be very well accepted. The collaboration was evaluated with the fourth part of the questionnaire. The question about the helpfulness of the collaborative features had mixed results (QCO.1 at average 4.0). While it received many ratings of 5, four ratings were 3 or below. During the session we observed a non-constructive collaboration with some participants who accidentally deleted nodes that others were working on. We expect this to be improved by adding awareness features with a change history and undo support. The question about whether participants felt well-involved in the collaboration process (QCO.2), non-developers gave lower average ratings of 3.3 compared to developers at average 4.3. We also observed how experienced developers often took the initiative and continued working on tasks faster than non-developers. However, all participants agree, that the combination of developers and non-developers works well with SWEVA (QCO.3). We suppose that non-developers expect a training effect after some time. In line with that, all participants see how the collaboration encourages the sharing of common experiences and knowledge (QCO.4 at average 4.5). Most participants were able to learn something new from their collaborators (QCO.5 at average 4.4); all non-developers gave the maximum rating of 5 here. Shortage of collaboration awareness is also highlighted by results to QCO.6 at 3.9 that asked about whether users were clear what other participants were doing. Finally, the fifth and last part of the questionnaire asked about the general opinion about the tools. All participants found it useful (QO.1 was rated 4 or better) and can imagine using it in the future (QO.2 rated 3 or higher with one exception). The last questions offered a free text input field for suggesting possible use cases. Participants see various application areas of SWEVA for data visualization. The feedback was mostly very positive and offered helpful suggestions on improving the platform. Many even see themselves as being part of a community helping to improve SWEVA by adding new processing nodes and visualization types. The collaboration aspect worked very well and was accepted. However, it offers still room for improvement largely due to missing awareness features beyond the chat. Overall, the usability could leverage from more tooltips and by providing learning material in form of video tutorials. The modeling using a directed acyclic graph was applicable to community self-reflection. Further research studies are needed in particular application areas, though.

For the technical evaluation, we benchmarked example visualizations by measuring the time required and memory consumed to process the data and generate the visualization. All measurements were taken on a client machine with an Intel i5-3210M CPU with 2.50 GHz, running a Google Chrome version 49. To measure the speed, the data visualization request was run three times with disabled caching; in the following we list the average values. Regarding the heap memory consumption, we subtracted the memory consumption of an empty HTML page (1.7 MB) from the

memory required of an HTML page with our tools embedded. The first example was a visualization of Requirements Bazaar projects and contributors. It required 1.11 seconds to process and render the visualization. The overall time required for the Core Framework to complete its work was 0.87 s, out of which 0.68 s were spent waiting for the server to respond 15 REST API requests. It consumed 5.4 MB of heap and downloads 1.8 MB of resources. The second example visualized MobSOS usage statistics of all registered services. It required 3.04 s to calculate and render a line chart. The time spend by the Core Framework was 2.86 s, out of which 2.74 s were spent waiting for 8 server responses. The visualization had a total heap memory consumption of 5.3 MB while downloading 1.8 MB of resources. We can confirm, that waiting for server responses took most of the calculated time and is especially significant with many requests. As a suggestion, developers should use only few requests and utilize batch processing, if the RESTful API offers it. Alternatively, GraphQL databases with deep requests could help here to limit the number of round-trips (cf. Section 4.3). The Core Framework performs very well with short processing times, even for complex graphs. Memory consumption seems reasonable, but there is still room for improvement, as we did not focus on optimization techniques in favor of easier debugging.

6.5 Visual Community Learning Analytics

In the previous sections we argued that SWEVA can be employed in all kinds of use cases. Following the DevOpsUse life cycle, we first showed how visual analytics is used in Internet of Things networks. SWEVA was originally developed to follow the development activities in an open source network. Now, we want to discuss visual community learning analytics as a special case to answer questions regarding community-oriented informal learning settings. It is an example of how we are able to retarget our monitoring instrument based on its inherent modularity and extensibility through component-based software engineering. In particular, we present three use cases dealing with learning analytics, derived from European research projects over the last years that were introduced in Section 2.5.

Learning analytics aims to collect, manage, analyze and exploit data from learners and instructors to facilitate the actual learning process [Klam13]. However, learning analytics in formal learning contexts like a university setting is often restricted to collect and analyze data from students following curricula through a learning management system like Moodle. In informal learning, however, a deep understanding of learners and entities interacting with each other is needed. There is a lack of institutional roles which induces the negotiation of roles in communities based on reputation and expertise [Klam13]. In this open space without institutional setting, the context of a community using an information system gives a common social structure for the stakeholders. Community learning analytics takes a step further and strives to provide a deep understanding of interactions between learners and other entities in learning processes. It comprises the identification, analysis, visualization and support of informal community-regulated learning processes. It is concerned with identifying expertise within a community and in comparison with other communities. A learning system able to discover the experts within a community is empowered to transfer knowledge from experienced users to new staff. The access to expert identification [WoBl11] and expert recommender algorithms [DHKo08], along with the visualizations of their outputs, is therefore beneficial for our system. (Overlapping) community detection algorithms in turn are key to expert identifi-

cation systems, as they are able to distinguish between the core and the periphery of a community [Shah18].

6.5.1 Wearable-Enhanced Learning Analytics

Together with the shift from formal to informal life-long learning scenarios, we see an increased involvement of physical activities performed with the body, in particular with regard to collaboration with networked devices. It comes with an integration of declarative and procedural knowledge [Ullm04]. Wearable sensors and computers are the primary subject to be analyzed in these informal learning situations. A wearable computer is defined as an 'ìntelligent assistant that augments memory, intellect, creativity, communication, and physical senses and abilities" [Star01a, Star01] with components "small and light enough to be worn on a user's body for convenient operation" [GoMe03]. The term 'smartphone' usually represents a different form factor, yet the principles are considered to be the same [Blak15]. As always present body-worn devices, wearables are ideally suited to informal learning in the workplace. But learning with the help of wearables also introduces some challenges. First of all, continuous notifications may disrupt the user's workflow [IqBa08]. Besides learning-related notifications, workers may also be interrupted by non-work related messages. Overall, permanently available wearable devices have the potential to offer more integrated learning experiences as opposed to purely classroom-based scenarios. The workplace setting allows context-dependent solutions that embed directly into the work practices. However, the introduction of multiple new devices for learning in the workplace leads to a complex interplay of devices, services and people. It is particularly difficult to find out if and when learning happens, and whether it is successful.

As such, we consider learning with physical artifacts using wearable technologies as an ideal use case for community learning analytics. Conceptually, we regard all participants of our sociotechnical system as actors in terms of the Actor-Network Theory (ANT) [LaHa99]. According to the Actor–Network Theory, both human and non- human actors are part of the society. Actors always have to be understood within a network of other agents or resources that define the identity and properties of the actor. Moreover, complex networks themselves are considered actors in another, higher-level network.

It may leverage the actual context of the user, spanning from detecting the actual physical location and even tool the user is employing, up to measuring body parameters like the current heart rate. Industry 4.0 appliances and wearables offer an enormous amount of data and are therefore usable by a huge variety of learning services. Examples of such data are inventory trackers and alerts on incorrect operation. In comparison, body-worn wearable sensors may capture the heart rate, arm movements or track the eye gaze. The more data is available, the more complex it is to analyze and reason upon it. One of the challenges in reasoning and researching on this data lies in the high degree of context sensitivity and interdependence of data coming from machine and human data sources. For example, a higher stress level identified through a significantly increased heart rate may be the cause or effect of machine malfunction.

6.5.2 Informal Learning Scenarios

In [KoKl16] we present three testbeds from technology-enhanced learning, representing real-life scenarios for learning with physical artifacts, where the SWEVA community learning analytics approach is applicable. They originate from the third-party funded project Learning Layers, that was dealing with informal learning at the workplace for the construction and healthcare sectors. The scenarios aim to enable different types of digital devices to digitally enhance physical artifacts like tools with additional information.

Digital Toolbelt Scenario: At construction sites, workers have to deal not only with a large variety of building materials, but also with a huge quantity of different tools. As an additional burden for fast adaption of construction techniques, these materials and tools are rapidly changing. Thus in this scenario, workers carry around a mobile device in their work belt and wear smart watches on their wrists and possibly even wear a protective helmet with integrated augmented reality glasses. At the start of the day, the workers open a digital toolbox app for getting a list of the day's tasks. Depending on the specific task that the worker has to fulfill, the app presents a list of tools like drills and screws that need to be collected first. For every tool, comments from other workers or annotated videos are available as manuals that can be displayed on a mobile device or on the visor of a smart helmet. Additionally, new videos may be recorded and uploaded to a repository to be discussed by co-workers. Finally, the apps for wearables may track the current context to prompt construction workers in case there are subtle improvements possible to their work progress.

Healthcare Scenario: Staff in a hospital carries around tablets and wears smart watches and smart glasses. When entering a patient's room, the doctors and nurses are reminded on their wearables that healthcare data has been loaded onto the tablet. The wearable may already have presented the most important actions like medications due on its display. During these actions, further data may be shown on the smart glass as an augmented reality overlay, e.g. during operations. Finally, critical alerts on a patient's status may be automatically received on the smart glass at any time, just like pagers notify doctors about certain events.

Figure 6.13: Exhibition Scenario with Physical Artifacts

Private Home Builder Exhibition: In the scenario, visitors enter a space and then use their mobile and wearable devices for interacting with the exhibition items. The exhibition concept can be seen in Figure 6.13. For every exhibition item on display, more digital material is available on a Web-based backend. Notifications on the wearables guide the visitors to further material that can be accessed with the help of mobile devices. Such digital information include a textual description of the object, PDF documents, and multimedia information such as audio tracks and videos. Users may discuss the exhibits by adding comments to the digital versions of the items and even comment on remarks made by previous visitors, enabling a discussion. Besides simply viewing the items, visitors are able to bookmark links to physical artifacts in a personal library within the system. To take home the virtual impressions and collections after visiting the museum, users may email their library to an email address after the visit, with links to any discussion threads for further comments. A public display may temporarily be used to see the content on a larger screen; the material may also be printed out on a stationary printer in the museum. Both emails and printouts include Web links to the system, so that the discussion may be even continued later on.

In all three scenarios, the users leave traces behind of their usage of physical artifacts. For instance, wearable sensors may capture a worker fulfilling a certain task by operating a machine. Environmental data together with task-specific identifiers like production rejects may then be combined with the sensor data in order to detect experienced users. Later, the recordings may be replayed to novel users through augmented reality devices. The propositions of visual analytics in these scenarios are manifold. On the one hand, the effectiveness of the measured knowledge transfer may be visualized on a dashboard. On the other hand, visual cues gained through analytics can themselves be embedded right into the performance augmentation process. For instance, augmented reality devices may integrate security alerts coming from self-managed machines in the field of vision. In general, for technology enhanced learning researchers, the broad availability of IoT technologies in Industry 4.0 and wearable contexts opens new doors to explore the possibilities and limitations of applying wearables for various workplace learning settings.

6.6 Towards Immersive Community Analytics

Billinghurst et al. define collaborative immersive analytics as: "*The shared use of immersive interaction and display technologies by more than one person for supporting collaborative analytical reasoning and decision making.*" [BCBM18]. An overview of immersive community learning analytics with wearables is given in Figure 6.14. On the left, we see an industrial workplace equipped with sensors that are constantly measuring human movements, air temperature and other environmental factors. The human worker is wearing a number of body sensors, including a gesture-recognition device, a mobile phone in the pockets, and an augmented reality headset. Both, workplace settings and necessary activities are formalized in a standardized description language. Together with the sensor data, the latter are fed into a human activity recognition machine learning algorithm. The classified output, i.e. what activity is carried out and connected qualities, are fed into a visual analytics Web cockpit. Its precise definition of how to process and visualize data are defined in a community-aware, collaborative manner. With regard to communities, there are further aspects, e.g. data sharing, privacy, and legislative challenges. Finally, the results are fed back to the worker into the augmented reality headset or other actuators at the workplace.

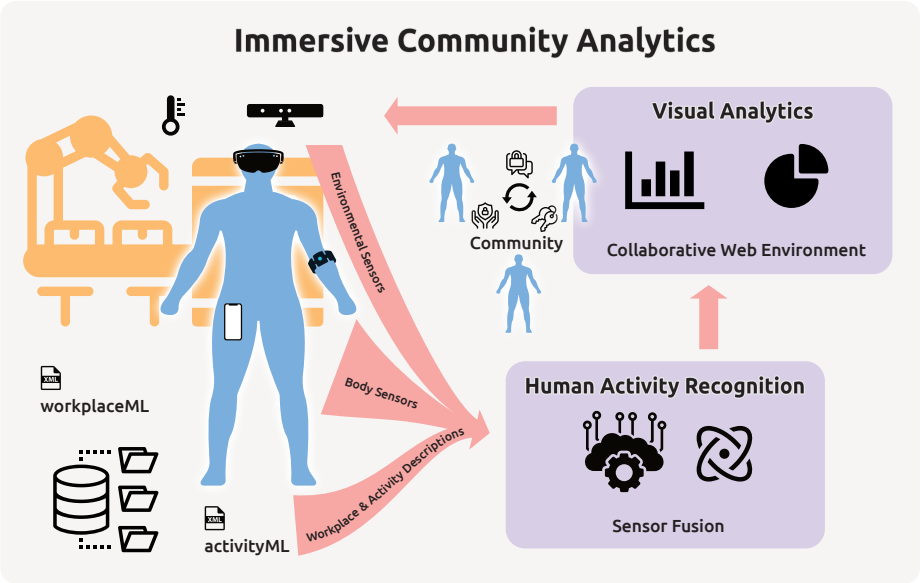


Figure 6.14: Immersive Community Analytics Scenario

In an interdisciplinary project together with the textile manufacturing institute (ITA) of RWTH Aachen University, we researched, how sensors at high-tech manufacturing settings can ensure human well-being in hazardous situations. As the use case, we chose a workplace where lightweight structures needed for example for airplane manufacturing are glued together out of composite material layers. In this setting, human workers are still superior, because of the huge variety of characteristics and material settings that need to be taken care of when working with composites. In particular, we were interested in visualizing robot-human interaction within the Web browser on a connected workstation. The goal was to identify situations within the interaction between a robot and a human, that resulted in an emergency shutdown of the robot. This was the case, when the human moved too closely towards the robot. These situations are a real threat in all industries, where human-robot collaboration is happening; therefore there exists a long rat's tail of standards and other legislative regulations. The original sensor data is coming from a Microsoft Kinect device that detects human movements. We then fuse it together with machine data from the industrial robot. A dedicated system is responsible for the calculation of the possible encounters. It causes the machine to stop in real-time, every time there is a risk of collision. SWEVA is responsible for the 3D visualization. In the data processing pipeline, we connected the output of the Kinect sensor and the outcome of the collision detection algorithm.

Figure 6.15 shows the resulting visualization of a human-robot interaction in SWEVA. Here, we see a robot (shown in blue) working together with a human (in red) for layering textile fibre composites. The goal of this visualization is to show possibly harmful situations where a robot arm is colliding with a human. The 3D output is accessible from every Web browser, as it is relying on state-of-the-art Web 3D technologies. While most of the work was spent developing and fine-tuning the collision detection algorithm, we were able to reuse SWEVA thanks to open Web standards and Internet of Things algorithms in the background. We are thus confident that the prototypes can be reused in a wide variety of industrial settings.

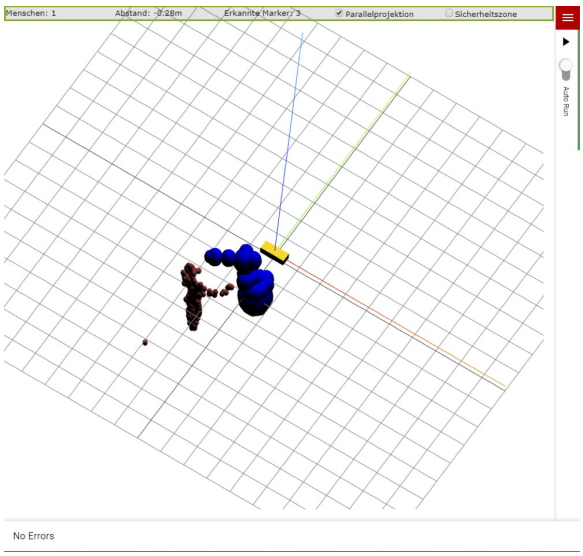


Figure 6.15: SWEVA Visualization of Human-Robot Interaction at a Production Site [Kraus18]

6.7 Conclusion

In this chapter, we introduced the analytical perspective on the DevOpsUse life cycle that is based on a collaborative visual analytics approach. Visual analytics combines the capabilities of computers to process big data, and the ability of humans to quickly detect relationships in graphical representations of data. We showed that we have mastered analytics for three technological generations. We started with a prototype dealing with Internet of Things sensors. It is capable of analyzing peer-to-peer message exchange in an XMPP IoT network. Then, SWEVA was introduced as Web-based community-aware system for collaboratively creating data processing pipelines that are used to generate dynamic, flexible visualizations. SWEVA works on arbitrary data sources; it handles the data flow from interface to visualization. It is universally applicable, easy-to-use and it runs on all Web-enabled platforms, even on constrained devices, since the processing can be offloaded to more powerful nodes running microservices. Our conceptual findings of componentized visual analytics pipelines can also be applied in various disciplines, while the technical framework can be embedded into any Web-based platform leading to ubiquitous visual wearable-enhanced learning analytics. Ultimately, SWEVA helps to significantly increase the relevance and applicability of learning services working with Industry 4.0 and wearable sensors.

We have not discussed security and privacy aspects of SWEVA, but we have followed appropriate regulations in the Learning Layers projects. This is especially important in the healthcare sector, where national and international privacy regulations prohibit the wide exchange of data. But even in the construction sector, our commercial partners in the Layers project were concerned about intellectual property rights and related issues such as what information to share and what to keep private. Taking these aspects into consideration is therefore of high importance. In particular with wearable computing, data ownership questions play a role [Gaff15]. For instance, heart rate monitors collect very personal data that could lead to negative consequences if leaked to health insurance companies (or loan providers etc.). On an organizational level, internal data as consequence of recording

traces for analytics within a company could find its way to competitors. Privacy concerns relate to personal details like a user's current location. A cloud service where the location data of multiple wearable app users is collected could be hacked and thus reveal patterns of the user's absence from home, enabling opportunities for burglars. Intellectual property protection is especially hard as wearables could enable hidden capturing of sensitive data. Gaff suggests to clarify such issues on a case-by-case basis prior to using wearables, to prevent disputes. As many different kind of devices are being released currently, lawmakers cannot fully grasp beforehand what laws are applicable to which technology.

This chapter concluded with a discussion on immersive learning analytics in Section 6.6. We discussed, how SWEVA has already been successfully applied in the realm of human-robot collaboration in the textile composite industry with a real-time 3D visualization of human-robot collisions. With the topic of visual analytics this thesis arrived at the last last part of the DevOpsUse life cycle. Self-awareness and reflection is an essential prerequisite for communities of practice in their pursuit of getting better. All artifacts supporting our methodology were presented in a community-oriented manner. In the next chapter, we shift our angle and look at DevOpsUse from an evolutionary perspective.

Chapter 7

Managing Evolution With DevOpsUse Infrastructure: A Longitudinal Analysis

The previous four chapters discussed the agile DevOpsUse community-driven information systems development methodology, looked at all its aspects and presented prototypes as design artifacts to deal with the particular challenges. The sequence of these contents reflected the loop around the DevOpsUse life cycle. In this chapter we now shift the dimension and look at our methodology from a different perspective. Mentioned aspects include applicability, scalability, sustainability and involvement. First, we apply DevOpsUse on three generations of technology that were introduced in Section 2.6: near real-time peer-to-peer Web architectures, edge computing, and the Internet of Things. We then demonstrate our methodology's capabilities through longitudinal studies in several large-scale international digitalization projects, and present key findings in terms of best practices. Finally, we show how the framework and its open source tools were validated in entrepreneurial and medical teaching courses.

All technological evolution is intertwined to some extent, but each has its own characteristics, making it special to deal with it in a common development strategy. Moreover, the societal impact of each one of them is undisputed, as the following examples show:

(a) Peer-to-peer architectures diffused into mainstream over the last years. Video conferencing in the browser with WebRTC is a practical aspect; Bitcoin and other distributed ledger systems however are potentially challenging our international financial system.

(b) Short distance communication like NFC is today built into every single credit card and most smartphones. The latest iteration of the Bluetooth wireless standard (BLE) meanwhile advanced entire business models; for example, the cashier-driven ordering process at the world's largest fast food chain was rapidly complemented by self-service ordering and table markers with high-tech indoor positioning based on Bluetooth beacons.

(c) Recently, our university changed from a closed source to an open source learning management system Moodle. Finally, in 2018, the European Commission's open source strategy was supplemented by sections on licensing and community building [Euro18].

While we have anticipated some of these developments early on [DKK*13, KoKl15, RKKJ17], technology continues to evolve. This chapter therefore shows that with the agile methodology we developed within the scope of this thesis, we can not only target communities in specific, but also handle all kinds of technological changes in general. The key outcome is that information system development is best dealt with in a societal context, explicitly integrating all community members while keeping their agency and strengthening their involvement.

This chapter is organized as follows. Section 7.1 first looks at DevOpsUse from the perspective of technological evolution. Section 7.2 gives recommendations and best practices for large-scale collaborative development projects. Section 7.3 discusses how to sustain DevOpsUse practices in teaching. Finally, Section 7.4 concludes this chapter.

7.1 Dealing With Technology Disruptions

The instantiation of the DevOpsUse life cycle in terms of tool support starts with continuous innovation and Requirements Bazaar that can be used in different usage contexts in order to capture and discuss the ideas of various community members. Then, end users can be integrated into model-driven development practices with the Direwolf Interaction Flow Modeler, leveraging their extensive domain knowledge. Here, we found the use of synchronous collaborative technologies to be the missing link between developers and end users. The created applications are then deployed with the help of component-based software architectures within the Layers Box. This has not only the advantage of being scalable through replicating available resources, but it also makes the applications and services modular. Contemporaneously we thereby moved the services closer to the *edge* of the cloud. Peer-to-peer technologies play an important role here, in particular the OakStreaming library for shifting resource-intensive video streaming from the cloud to the edge. Finally, the visual analytics perspective enables communities to analyze their usage of the developed software. With SWEVA, we empower communities to improve their agency by building their own analytics dashboards.

So far, we have implicitly shown that DevOpsUse can not only master professional communities of various disciplines, but also their specific needs in terms of technologies. For instance, construction workers in the Learning Layers project inherently rely on different hardware and software technological toolsets compared to medical students in a higher education scenario. In this section, we discuss how DevOpsUse was targeting technological leaps over the course of this dissertation. To achieve this, we take the above mentioned developed design science artifacts and highlight various aspects that make them applicable in additional use cases.

Within near real-time peer-to-peer computing on the Web, the standardization of WebRTC and its operationalization in terms of libraries and tutorials was a key enabling factor for a number of innovative use cases. Ultimately, standardization has led to its breakthrough and adoption. The most prominent application is Web conferencing. In the simplest case, two users are connected over the browser-to-browser link and exchange video and audio streams. In more complicated cases, multiple users are connected over browser-to-browser connections in a star topology. However, WebRTC allows for further use cases. In our Direwolf prototype, we leveraged the WebRTC connection for distributed user interfaces. More importantly, we were able to allow low-latency gaming over the

WebRTC data channel connection. The data channel emulates a reliable transfer protocol on an unreliable UDP connection.

For edge computing, a notable enabler was the architectural style of microservices and their deployment in container-based settings. Besides easier replicability for scaling up, microservices come with a number of advantages for the development process compared to monolithic applications. First of all, developers can be integrated faster as the hurdle to deal with a single functionality is lower. Second, they can be run and tested independently even on development workstations. Third, interfaces are contracts that can be used for testing as well as for code generation. The Layers Box discussed in Section 5.2 is an edge-oriented runtime environment for custom learning services, managed by a community of practice. We showed how it is able to run manifold software services from single sign-on providers and databases to custom-made video annotation software. In Section 5.6.1 we showed how serverless computing is the next evolutionary step in the long history of componentization and modularization. Corresponding frameworks running within Docker containers can be embedded within our Layers Box implementation.

Internet of Things is a paradigm that diffuses the borders of the Internet including the edge and integrates everyday devices that were not thinkable as computing devices before. These include smart devices for home automation but also large-scale production hardware in Industry 4.0 settings. In the ACDSense scenario discussed in Section 6.3.2, we evaluated XMPP over a federated scenario across three universities in Germany.

In this section, we showed, that DevOpsUse as development methodology could master three major technological leaps over the last years. We are confident, that it we will be able to do so in the future. To highlight the societal dimension, in the next section we analyze evidence for DevOpsUse processes within a large-scale European research project.

7.2 Long-Term User Involvement in Large-Scale Digitalization Projects

In research projects, many stakeholders get together to work on societal research problems. Most of these problems to be answered require complex information systems that need to be developed. We call these systems that manage societal processes societal software: it is software that pays as much attention to the process of creating the software product as to the software product itself [RKKJ17]. They are developed within unknown territory, as their explicit goal is to research innovative solutions. However, often there are no experienced developers in these research groups. Instead, student workers and PhD researchers without appropriate training fill in the void with prototypes that are often not market ready in the end. Development thereby is largely a social process, as consortia are required by funding agencies to compress all kinds of expertise within: application partners, developers, and researchers. It is precisely in this area of tension that it is interesting to see whether the DevOpsUse methodology is applicable, and how it performs. There are various challenges inherent to software produced by research projects. First of all, often funding agencies endorse the use and creation of open source software. Open source software is therefore suitable to address the quest for sustainability, by opening up the code and allowing outsiders to participate in the creation of code. While this way, external innovation may flow in, there is a threat concerning the outflow of brain

power. Sometimes, partners turn away during the project runtime, at other times consortia split up, either openly as a consequence to unresolved disputes, or privately, to follow hidden agendas. We therefore feel the urge to introduce both technical as well as social instruments that manage the development. The role of end users, on the other hand, is often well-defined in the research proposals. Face-to-face workshops and focus group interviews are part of the requirements engineering processes. The problems that arise are excessive traveling costs of multiple project partners. Further costs may be incurred as some end-user groups do not participate in centrally located focus groups without monetary incentives.

To analyze the social processes of information system development in a research project, we looked at the traces left behind the Learning Layers FP7 project that we participated in. Following a participatory design objective, all project members were assigned to four different co-design teams. The co-design teams acted as collaborative instrument, each targeting on a specific use case from the main application contexts. Two of the teams were particularly focusing on the health care domain (PANDORA and Bits and Pieces), and two on the building and construction domain (CAPTUS and Sharing Turbine) [LCD*14]:

- **CAPTUS:** The sharing of work practices in the construction domain is limited by the varying attitudes towards technology and availability of skilled workforce for specific methods. This prototype therefore allowed recording short amounts of video, to annotate and share them. This is to allow pointing out objects and actions in the scene and enhance it with specific knowledge.
- **Sharing Turbine:** The idea was to produce a mobile app for construction apprentices that resembles a boundary object (the so-called "white folder") between formal training and informal learning at the workplace as part of dual system training. The white folder previously was a physical folder where apprentices collected their training tasks and documented their achievements.
- **Bits and Pieces:** As informal learning is episodic in nature and thus distributed over time, this application aimed to support healthcare professionals in recording bits and pieces of information (Web links, media, pictures, voice recordings, etc.) to capture informal learning experiences. The tool provides cues (time, place, topic or person) connected to the piece of information to allow users to make sense of them at a later stage.
- **PANDORA:** It addressed the opinion and information formation concerning clinical guidelines. The adoption of these clinical guidelines, previously developed and disseminated in a top-down way, needs guidance and translation into general practice work practices. The application therefore supports critical commenting on fixed documents, enabling a discussion from bottom up.

Figure 7.1 shows the results of our case study. The four design teams and their traces are color-coded and placed around the DevOpsUse life cycle model. We gathered these data points through numerous artifacts left behind the design teams in our collaborative design space. This includes various pages created and updated in the project wiki, text documents shared in the collaborative cloud space, as well as photos, videos and audio recordings distributed within the project. Additionally,

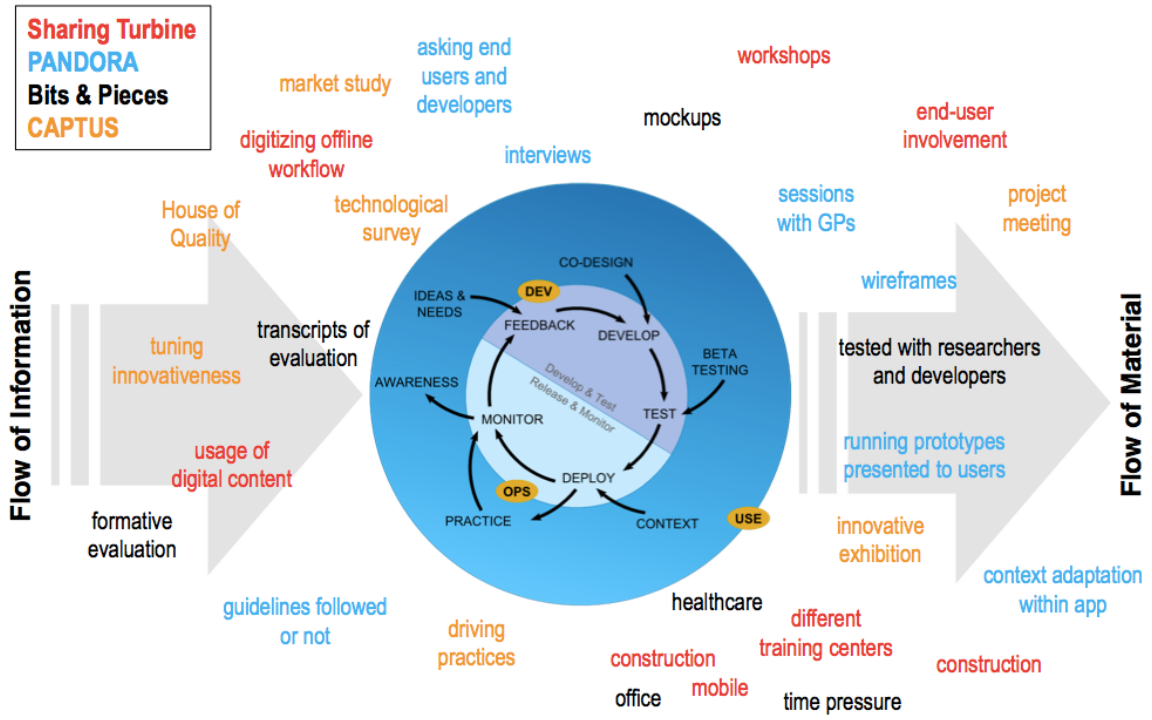


Figure 7.1: DevOpsUse Case Study With Four Co-Design Teams

we collected initial requirements based on artifacts from the research project contract (description of work), outputs from a design conference held in the first year of the project, as well as interviews with stakeholders and co-design data in an open library of the project[1]. Lastly, the Requirements Bazaar and the House of Qualities produced contained many traceable objects of interest. We then attributed the data to the initial design teams and grouped it into the two sectors construction and healthcare. In another step, we assigned the artifacts to the in- and output processes of the outer circle of the DevOpsUse life cycle model.

From a birds-eye perspective, we see a flow of information coming in from the left. This represents the specific knowledge brought in by, in our case, construction and healthcare professionals. They were harvested in various co-design sessions by the four design teams. To the right, we see an outflow of material. These are the systems, frameworks and specific tools and applications developed by the project team. Although the flow of information and material is flowing in one direction, the general model is cascaded, thus new specific knowledge and information that is generated through the created material is again flowing in on the left, further enriching the information systems development life cycle.

In the following, we look at the various design teams as listed above, and discuss their traces clockwise from the top left, just as we discussed the DevOpsUse life cycle in the previous chapters. Sharing Turbine's main starting point was to digitize the offline workflow of maintaining the white folder. Through end-user involvement in group workshops held at construction training facilities

[1] http://odl.learning-layers.eu

in north Germany, the team collected necessary requirements to develop the app. Later, the usage of digital content was analyzed. The PANDORA team conducted interviews with healthcare professionals and general practitioners. Through wireframes and running prototypes, the end users were shown the app. The usage was analyzed by checking whether the commented guidelines were followed or not. The general idea of Bits and Pieces, on the other hand, was adequately tested with researchers and developers themselves, since small traces of information could be collected well in their working practice. The findings of the formative evaluation was then fed back into development. CAPTUS, that developed the Ach so! application [VPBL15] to record short videos that could then be annotated (cf. Section 2.5), started with a market study on similar tools and a technological survey to capture frameworks and libraries that supported parts of the functionality. These were fed into a House of Quality (cf. Section 3.2.1). The prototypes were then tested in a project meeting and a construction exhibition that showcased new materials and tools. Before the application was rolled out, small tweaks were added to raise the perceived innovation factor; e.g., videos automatically stopped at the annotations for a few seconds to give the users time to read them.

Overall, the case study has helped greatly in identifying and demonstrating these traces of end user integration within a large-scale research and development project. We identified Requirements Bazaar as boundary object for all the co-design activities and thus see evidence in its applicability and usability. Ultimately, clusters of artifacts enabled us to detect overlaps in requirements and contexts between different design teams and apps/services. For instance, we have found that the context of a requirement alone widely influences many technological decisions at a very early stage. In the healthcare sector, we saw strong demand for private cloud services, while in the construction industry many requirements targeted mobile computers.

7.2.1 Project-Specific Challenges

As described above, interdisciplinary research projects face a multitude of challenges concerning their software development. Various stakeholders with diverse interests are required to produce software within a short runtime. In international projects, members are often distributed over at least three different countries, as required by funding agencies. Other challenges include the understaffed developer community with inexperienced developers, that often belong to various (overlapping) subcommunities, like developers and PhD researchers. While researchers strive for scientific success in terms of reputation and impact, application partners want to explore new business models, among other things. There are also different exploitation strategies. Small companies often depend on produced intellectual property, having invested relatively large amounts of financial resources and man power. It is therefore essential to become aware of the dichotomies and to respond to their disparate motivational factors and intentions. Even within industrial partners, there may be experienced companies and small startups.

In Table 7.1, presented at the International Conference on Software Engineering 2017 [RKKJ17], we show an overview of common software engineering challenges. Additionally to the evidence-based case study in the previous section, we collected several best practices within our research projects over the last years. Mainly, these are experiences from the ROLE and Learning Layers projects. The negotation of the main outputs early on is key to success. However, it must be acknowledged and reflected that they may change over time with shifting requirements and priorities

Table 7.1: Common Software Engineering Challenges in Research Projects [RKKJ17]

Early Decisions	Architectural decisions with considerable scope need to be made early in the project, as it is well known that early mistakes in a project are the costliest [West02]. These decisions should also include reflections on work package and task structure.
Short Cycles	In the time frame of an R&D project, development cycles must be kept short, and initial architectures and prototypes must be provided early in the project to allow for frequent refinement loops driven by research and end-user involvement.
Sustained Impact	Deployed solutions have to scale both horizontally and vertically, as project outcomes should be exploited and sustained beyond the project's funding period. Typically, exploitability and impact of project results are key performance indicators for funding agencies, although market-readiness of ICT research project outcomes is often limited [Euro09]).
Distributed Community	The initial community is a union of individuals affiliated to different partners. It needs to act as seed for a growing and flourishing wider community involving external members in a limited amount of time.
Support Infrastructure	The development process and environment, including procedures and tools for source code management, issue tracking, software branding, and similar, need to be defined, implemented and maintained. Eventual licensing and maintenance costs and efforts may not be underestimated.
Licensing	An OSS strategy usually helps sustain and transfer project results into practice and is thus interesting for funding agencies [Euro18]. By definition, any OSS must ship with an OSS license. In large project consortia, agreement on one license for all developed components is hard to reach. The use of third-party OSS and differing exploitation goals among partners impose the use of different, however compatible licenses.
Stakeholder Engagement	Societal software projects inherently require instruments to achieve wide coverage of stakeholder engagement. Such instruments thus need to be as inclusive and supportive as possible even for non-technical audiences.
Baseline	In research projects, the state of the art builds the baseline for all activities. "Quick & dirty" approaches ignoring existing knowledge may thwart the cutting-edge research ambitions of the project. However, software engineering processes need to be flexible to understand and adapt to emergent change, resulting from the complex and never ending interplay of people influencing technology and vice versa, as we find it in socio-technical systems and as it is described by ICT-related adaptations of structuration theory [Orli10].
Unknown Territory	Research outcomes inherently remain unspecified beforehand, while the research methodology leading to those expected outcomes must be well-defined. This poses a concrete challenge, as software engineering activities need to synchronize with primary research activities.
Success Awareness	Any software artifact developed within research projects requires rigorous evaluation with respect to success as complex construct combining quality and impact as they are perceived by different relevant stakeholder groups. With respect to sustainability and given that success itself is a highly dynamic and context-dependent construct, success awareness must be conceptualized as result of ongoing long-term developmental evaluation even beyond project lifetime [BCD*15, Renz16].
Decision Making	Although research institutions collaborate in projects with legally binding contracts and agreed work plans, all of them have different motivations, internal agendas and working processes to be aligned with the overall project agenda. This diversity often poses an obstacle to efficient decision making.

of stakeholders. Integration should be a convergent force. We explicitly borrowed the term integration from software engineering, where it regularly and automatically binds together different strands coming from various software components, and tests their combination. There are three perspectives to integration. First, the integration with application partners through requirements engineering. Second, server-side integration with unified services. Third, client-side integration through end-user tools. In the next sections, we present our set of DevOpsUse instruments and tools in action within our research projects. These include a developer community, a governing body, as well as a pre-configured development infrastructure that especially helps speeding up practices in the initial phase.

7.2.2 Developer Community

Inherently, social aspects play a major role in community-driven information systems development. Therefore, we have initiated the explicit formation of various subcommunities within the research projects. One of them was the developer community, also called the *developer task force* for acknowledging the importance of developers. Because part-time development in small distributed teams hampers the overall development progress [Kuns07], we initiated a project-wide developer community, with the possibility of smaller task-specific groups for immediate communication. To include younger developers from the start, several instruments were set up. In the later WEKIT project, a hackathon was organized right at the start of the project. This has resulted in early onboarding of younger developers in social Web 2.0 tools such as Facebook. As a side effect, opening up earlier gives credibility to participants in terms of a higher weighting of their own ideas. In addition, it played an important role that the developers got to know each other personally. This has removed the barrier of communication via online tools.

Tool support for the developer task force, besides the development-oriented toolset discussed in Section 7.2.5, consisted of a video-based online conferencing tool and a Wiki for short- and long-term documentation. As a blended approach, the developer community worked together with a governing body that is described in the next section.

7.2.3 Governing Body

While the developer community handles day-to-day tasks on a regular basis, its issues sometimes touch upon territories that have a wider, often strategic impact on the project. For instance, the decision to hosted cloud solutions versus setting up an own operation infrastructure at local application partners, can have a significant financial dimension that needs thorough consideration from all stakeholders. In order to relieve developers from such decisions, while acknowledging their importance, we set up another social instrument as governing body, the *architecture board*. It came together in regular intervals, often within general project face-to-face meetings, but also via online conferencing software. The board was mainly informed by the developer community, but it also took into consideration the voice of application partners and the project agreement. Binding decisions were taken by means of votes, one per board member. Thereby, each work package had one vote. We also added voting power to the co-design teams responsible for certain cross-functional work streams within the project.

With regard to the governance structure, it should be noted that boards like the governing body play the role of a loose group which nevertheless meets regularly. This multi-stakeholder flat hierarchy therefore stands in contrast to firmly anchored top-down approaches, as is often the case, for example, in companies where strategic decisions are mainly made within the management circle. Following an agile methodology, these boards have to be evaluated regularly. For instance, as described above, in the Layers project we gave the co-design teams described in Figure 7.1 more agency over time, as their impact concerning the developed software became stronger.

After discussing the social factors of project collaboration, we now turn towards the tool support layer and present a pre-configured infrastructure for faster initial setup in the next section.

7.2.4 Developer Hub

One of the major factors in an open source strategy is reaching out to third-party developers. While the actual source code, hosted on open repositories like GitLab or GitHub, is inherently the most important resource, as without them the software is not open source, an extensive documentation is key. This comprises various resources such as documentation, built libraries, tutorial videos or even promotional material like downloadable or t-shirts and flyers available for order. Within the Layers projects, we ran a series of webinars about development-related tasks. The topics ranged from general ones like continuous integration and container-based deployment up to highly specialized ones about particular aspects of the project-specific infrastructure and libraries. All the videos are linked on the developer hub.

An overarching site presenting libraries and other developer-oriented tools is a strategy also followed by global players such as Google and Amazon. Both websites include not only API documentation but also extensive training materials up to complete online courses that result in a certificate once completed.

7.2.5 Pre-Configured Infrastructure

After discussing the social instruments of the development life cycle, we now turn to recommendations for the technological development infrastructure. In particular, we now close the gap and show how the proposed tools reciprocally interact with state-of-the-art agile software development practices through automation. Our proposal thus includes all collaboration tools that enable (remote) developers to work together on a common software project. We see a particular need for the development infrastructure to be standardized across participating organizations right from the start, otherwise the unclear situation leads to alternative infrastructures at individual institutions, hampering the first release cycles and manifesting bad practices due to missing integration. To avoid unnecessary manual work, we stress the need for automation on all layers. Automated frontend and backend tests can help identifying bugs early on. Continuous integration processes execute build tasks for code that has been checked into the repositories right after it is available. Besides testing individual software packages, there should be test cases that validate projects that build upon each other. For instance, the API calls of a frontend application should be tested against the service that offers the API. It becomes evident, that model-driven technologies with code generation for both frontend and backend parts are highly advantageous for this matter. Automating the delivery of software helps for releasing software regularly. Automated deployment makes sure that software is kept updated on all debugging and production systems.

Recommendations change over time because of the everlasting duality between social and technical development. The influx of developers experienced with a certain tool, who are willing to organize training sessions for lesser experienced developers can lead to the adoption of the tool. Therefore, the following concrete product recommendations should be treated with caution. Rather, it should be treated as testbed that is able to adopt and absorb new tools and instruments. We start the walk around the development life cycle depicted in Figure 7.2 with the social requirements engineering. For this, we use Requirements Bazaar. For code hosting, we set up a public GitHub organization under a central project name. That not only boosted convergence but also gave a common structure

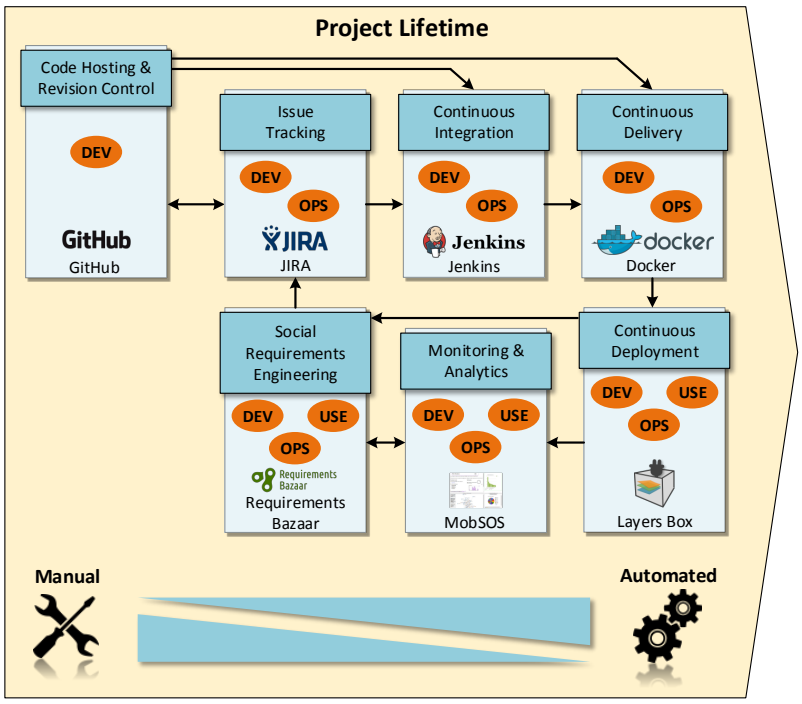


Figure 7.2: DevOpsUse Roadmap

that resonated in the developers' identification with the project. A self-hosted open source alternative to GitHub is GitLab that further integrates some project management tasks. Issue tracking can be performed either via GitHub issues, or via JIRA that offers an open source license. Continuous integration is performed using Jenkins or Travis CI. The continuous delivery happens with Docker containers, and their release on the hosted Docker Hub software container repository. For the continuous deployment, we developed the Layers Box that draws its containers from the central Docker Hub repository. It was presented extensively in Section 5.2. For integrated analytics, i.e. summative, formative and even developmental, we use the MobSOS framework, which in turn can again inform users of Requirements Bazaar.

As mentioned above, these recommendations are highly susceptible to changes in the tool environment and licenses. For instance, the GitLab tool recommended above for self-hosted code hosting, integrated various means for project management over time, like Kanban. It can even fulfill the roles of continuous integration and continuous delivery to the Docker Hub repository. Other solutions such as SourceForge turned out as a wrong choice in one of our earlier European projects, as managing various subprojects turned out to be cumbersome with the Subversion code repository system. However, we stress the importance of introducing a general software engineering methodology, but acknowledge that it will evolve in parallel with the software artifacts it produces and the community working with it.

We explicitly exclude the actual *integrated development environment* (IDE) from our tool recommendation chain, as its choice depends on the individual preference and the accumulated experience of the developer. This is supported by the current phenomenon away from extensive "all-in-one" IDEs like Eclipse and Visual Studio towards flexible, plugin-based text editors such as Atom and

Visual Code. These novel editors, which have emerged in recent years in the tradition of older editors such as Sublime Text, have in common that they are developed open source with JavaScript, and are made executable with a cross-platform framework like Electron [Elec19].

7.3 Teaching DevOpsUse

In order to sustain the DevOpsUse methodology, and to gain further experiences in adjusting the parameters and trying out new tools, we integrated it in both theoretical and applied teaching over the last years [LNKK16]. This step considers the aspects of scalability through internalization in education, and involvement through interacting with a large number of people. Within the newly developed *Social Computing* lecture that is targeting master students, we are talking about DevOpsUse in the context of community-driven development and agile methodologies. As a practical exercise, students are asked to enter requirements into Requirements Bazaar, and to rate and comment each other's requirements. By not only learning, how to build social computing applications and services, but to see the holistic approach, students are getting acquainted with how real development processes in the industry work. Additionally, by trying out Requirements Bazaar as a new tool, we enable students to think about the bigger picture, and therefore about the general approach behind the concrete tools they are using; often, the students' knowledge is restricted to the usual tool offerings provided by the university (e.g., GitLab) or indirectly nudged by the industry through discounts (e.g., *GitHub Student Developer Pack*[2]). While many of the presented tools like Requirements Bazaar, SWEVA and Pharos were themselves developed and extended by bachelor and master thesis workers and therefore been a subject of research, our students actively use our Web applications and methodologies for their theses. For instance, requirements for their prototypes are captured in Requirements Bazaar and discussed with their advisors and supervisors. Later, the LaTeX source code of the written assignment is stored within our GitLab, where a continuous integration server is compiling PDF versions automatically after every commit using a Docker-based build system. Issues and annotations about the content are collected as issues within GitLab and worked upon in Kanban-style boards.

In the annual *High-tech Entrepreneurship and New Media* lab course, multiple teams of 4-6 students cooperate with local startup companies to develop a project-oriented software. The students work together with company employees in a community of practice to mutually engage in the advancement of their software prototype, by working on an innovative new business model, that would otherwise not be feasible for our startup partners [RKJW07]. While the students are prepared for later employment, the companies get to know state-of-the-art approaches coming straight from research. Embedding it in this course therefore has the potential to radiate DevOpsUse beyond the university and carry it into startups.

To promote our usage of DevOpsUse first within the Learning Layers consortium and then with a broader circle around the Layers open source projects, several webinars have been produced and published on the Layers YouTube channel[3]. A webinar is a combination of slides and some other video content like screencasts in a short video, connected with some tasks, so that the viewers may

[2] https://education.github.com/pack

[3] https://www.youtube.com/user/learninglayerseu

Figure 7.3: Learning Layers DevOpsUse Webinar Badges

emerge in directly trying out the presented things after watching the video. The Layers DevOpsUse webinars are intended to demonstrate the tools and practices that we were using in Layers for managing the project-wide collaboration on OSS across Europe. Additionally, the series is about the design, development and operations infrastructure of the Learning Layers project. For multiple of the webinars, tasks have been prepared that the audience can fulfill to get familiar with the tools; they are listed alongside the videos. For easier recognition of what video is produced for which target audience, we created a badge system with badges for end users, researchers, developers and operators. The badges are shown in Figure 7.3. With the help of these webinars, we were able to involve non-developers in the collaborative writing process happening for the final deliverable of the project; it was performed on GitLab with the output being an automatically generated static website.[4] In the following, we summarize the content of the six Layers webinars. They are available in the Open Developer Library (http://developer.learning-layers.eu/documentation/devops-webinars).

- **Challenges:** The first webinar gives an overview about the Learning Layers project. Some challenges are mentioned like how to deal with scaling up the delivery of innovative learning solutions together with managing the user feedback to make the tools better. At the end, the DevOpsUse life cycle is presented and some pointers to the rest of the webinars are given.
- **Requirements Bazaar:** In this webinar, we demonstrate how end users and developers can get in contact with each other and use social features of the Requirements Bazaar to start the requirements negotiation process. We show how likes and comments enable a rich communication. The webinar includes an exercise on basic usage of the Requirements Bazaar that mentiones best practice guidelines.
- **JIRA Issue Tracker:** This webinar gives an overview about the most important functionalities of JIRA for Layers. It is explained how to create a new issue and how developers go through the life cycle of problem resolving.
- **GitHub and Jenkins:** This webinar presents the project's usage of GitHub and Jenkins. GitHub is shown as a modern source code repository highly used in open source development and Jenkins is demonstrated as the leading open source continuous integration server. This video is also interesting from an operations perspective regarding continuous integration and deployment.
- **Layers Box and Layers Adapter:** Large-scale deployment through Layers Box and Layers Adapter is covered in this webinar. More concretely, it is demonstrated how a Layers Box contains services for authentication and authorization using OpenID Connect. It is also

[4] http://results.learning-layers.eu/

shown how the box serves the services that are needed by the frontend Layers apps through the Layers Adapter.

- **MobSOS:** This webinar introduces MobSOS as a framework for providing methodological and technical support for communities to develop success awareness of the IT artifacts they create, use, and research. This video is dedicated to end-users, developers, operators, and researchers. After a short contextualization of MobSOS in the DevOpsUse cycle, viewers are introduced to the goals, concepts and tools of the MobSOS approach. Two guided hands-on exercises allow viewers to try out MobSOS tools interactively.

In order to familiarize computer science students with DevOpsUse, we created teaching material on state-of-the-art software development methodologies. The slideset shown each year presents theoretical background about communities of practice, but also includes hands-on tutorials about source code management with GitLab, and continuous innovation with Requirements Bazaar. The slides have since then be also integrated into the course material of the AR-FOR-EU European project for teaching augmented reality, where multiple courses have been held, e.g. in Oxford, UK in January 2019, as well as Klagenfurt, Austria, in June 2019.

7.4 Conclusion

In this chapter, we shifted perspective on the developed community-driven continuous innovation methodology for developing information systems: While in the previous chapters we followed professional communities around the software development life cycle of continuous innovation, model-driven development, deployment at the edge and finally visual analytics, we now showed how DevOpsUse can answer particular technological leaps as well. As the Web is spreading more and more and thereby taps into different types of digital technologies from wearables to Internet of Things, it becomes necessary to constantly adapt and refine our developer support toolset. For instance, we have identified several technological leaps in he course of the last years. In particular, we discussed near real-time peer-to-peer technologies, edge computing and the Internet of Things. Exemplarily, we gave pointers to how DevOpsUse was implicitly or explicitly adapting to them. Discussed aspects included *credibility* and *consolidation* by giving stakeholders agency through governing boards, *scalability* and *involvement* by including co-design practices into the infrastructure, and lastly *comprehension* through embedding DevOpsUse into teaching practices.

Then, on a more general level, we discussed how DevOpsUse is applicable in large-scale international development projects. Through empirical observation, targeted interviews and evaluation of produced design and software artifacts we have proved the success of the methodology in several European research projects we took part in over the last years. We showed, that many of the structures evolved within the cross-functional co-design teams responsible for various work-streams of these projects. Based on our experiences, we listed several recommendations of particular tool configurations and social instruments like a developer community and a governing architectural board. The toolset is not to be understood as a fixed entity that cannot be changed over time and across projects. Instead, it will evolve together with the software artifacts it helps developing. Thereby, it acts as a flexible testbed for absorbing new stimuli and forces. We acknowledge, that the social

perspective plays a major role in the success or failure of information systems development. We are convinced, that our methodology is employable for future societal challenges and technological leaps as well. The next chapter concludes this dissertation with a retrospective take on the research questions posed in the introduction.

Chapter 8

Conclusion & Future Work

This chapter concludes this doctoral thesis with an overall summary of the findings and pointers to possible future work. The contributions are divided into published research and open sourced software. They represent both designed artifacts as well as the communication about them. The results summarized in this doctoral thesis were published in four major research domains within computer science. Furthermore, we have already seen evidence of the released open source software being taken up around the world.

The future work reaches from well-known tracks in the area of computer science up to new directions with societal impact such as *citizen science*, and *grassroot* movements. Just as agile principles, originally coming from software engineering, have improved and still transform the processes of various industries, we see an enormous potential of the DevOpsUse methodology and tools to apply them beyond information systems engineering. Community-driven design combined with automated artifact generation has manifold chances to this end, but again opens up several interesting challenges.

8.1 Summary of Results and Contributions

The contributions in the following are split into published research and released software. While the research contributions in Section 8.1.1 illustrate the innovation capabilities and theoretical background that this thesis contributes, the software section demonstrates the applicability of the conceptual work. To ease the adoption for manifold use cases, we followed an open source approach where all developed artifact were made available at the open source code repository GitHub within our research group's organization.[1]

8.1.1 Research Outcomes

Despite the challenge of our interdisciplinary research work, we were able to push innovation boundaries of the areas Web and Software Engineering, Technology-Enhanced Learning and Hu-

[1] https://github.com/rwth-acis

man Computer Interaction. These achievements are discussed in the next sections. In addition to the written contributions and oral presentations given at conferences and the peer review activities for conferences and journals, we have also initiated a workshop and published a consecutive special issue of a journal in the field of co-design. These activities are the last part of the design science in information system cycle, namely *communication*.

In the following, we revisit the research questions introduced in Section 1.2, list developed artifacts and reference published work. The articles answering the questions were not necessarily published in chronological order. This is partly due to the technological background of the Web as a changing medium; some standards such as WebRTC for browser-to-browser communication were only declared stable during our work. Thus, as these developments opened up new possibilities, we integrated them in our approach. On the other hand, this is also due to the necessary adaptation of our design artifacts, following the cycles of the design science research methodology.

RQ1: Building Blocks of Information Infrastructures

- Peer-to-Peer Video Streaming [KoKl18d, KoKl19]
- Peer-to-Peer Distributed User Interfaces [KBKl14, KNKl15]
- Generation of Web Frontends for OpenAPI [KoKl18b, KoKl18d]
- Linking Physical Artifacts to the Web [KoKl15, KoKl16, SGR*14]

Regarding the first question, our main contribution is the identification of peer-to-peer architectures as a necessary constituent of a scalable and community-driven infrastructure. This includes a detailed exploration of different usage scenarios such as video streaming [KoKl18d, KoKl19] and low-latency communication within distributed user interfaces [KBKl14, KNKl15]. Service descriptions like OpenAPI and AsyncAPI are stabilizing factors; we leveraged them for the model-driven generation of user interfaces [KoKl18b, KoKl18d]. Regarding the Internet of Things we showed how physical artifacts can be linked to the Web to ease adoption [KoKl15, KoKl16, SGR*14].

RQ2: End User Tools for Developing Community-Specific IS

- Smart Ambient Learning with Physical Artifacts [KoKl15, KoKl16]
- Direwolf Interaction Flow Designer [KoKl18c]
- Co-Design of Gamified Mixed Reality Applications [KHKl18b]

Linking physical artifacts to the Web also partly answered the second research question, as it facilitates end user activities such as informal learning at the workplace [KoKl15, KoKl16]. Besides, we identified abstractions through models as boundary object between developers and end users [KoKl18c]. In our design artifacts, near real-time collaboration plays an important role. Similarly, we demonstrated how gamification and co-design enable designing community-specific applications within innovative areas such as mixed reality [KHKl18b].

RQ3: Establishing a Scalable Continuous Innovation Life Cycle

- Best Practices for Collaborative Projects [DRN*15, RKKJ17]
- Crowdsourcing Co-Design [BKKl19]
- Community Learning Analytics with IoT Devices [KoKl17, KoKl18]

Concerning the last research question about the sustainability of the innovation capability, we published multiple guidelines with best practices for the organization of large-scale digitization projects [DRN*15, RKKJ17]. For scaling user involvement, we introduced means of crowdsourcing within the realm of visually co-designing Web applications [BKKl19]. Finally, we deem community self-awareness as essential for monitoring and reflecting on the information system usage [KoKl17, KoKl18].

In the following, we continue with a discussion of particular research domains and discuss particular aspects of our publications in more detail.

Web Engineering: In the Web Engineering domain, we published several articles at the International Conference on Web Engineering and the associated Journal on Web Engineering. The research presented at that conference revolved around the topic of distributed user interfaces and related thereto experiences with peer-to-peer computing from within HTML and JavaScript Web frontends. The Direwolf framework, first published by Kovachev et al. in 2013 [KRNK13], was thereby extended in 2014 with direct peer-to-peer connections with WebRTC [KBKl14]. In the latter paper that was awarded with the Best Demo award, we showed how WebRTC can decrease browser-to-browser latency from around 150 ms to less than 25 ms depending on the specific network conditions. This enables new innovative use cases like gaming. Since then, we have seen many implementations using WebRTC published by the Web Engineering community. WebRTC was also in the focus of our article about peer-to-peer video streaming on the Web with the Oak-Streaming library [KoKl18c, KoKl19]. Here, we showed how smart usage of Web technologies can decrease the load on centralized points in the network infrastructure. Thereby, even economical and privacy aspects could be considered.

Collaborative aspects on the Web have also been a topic of our research. In a paper published at CollaborateCom 2013, we analyzed various existing developer support libraries that enabled near real-time collaboration on the Web [KGKl13]. While one of them was used for the collaborative House of Quality prototype featured in this thesis (cf. Section 3.2.1), we helped identify shortcomings that in the end lead to the development of the Yjs library at our chair. It was employed, for example, within our video annotation prototype presented at the International Conference on Web Engineering in 2015 [KNKl15].

Technology-Enhanced Learning: Most of the research presented in this thesis was funded by large-scale European projects in the area of technology-enhanced learning. Therefore, the most obvious publication opportunity, also following the research tradition of the ACIS group, were conferences and journals in this area. We coauthored several papers at the European Conference on Technology-Enhanced Learning and associated workshops. At the Joint European Summer School on Technology Enhanced Learning, we prepared and several workshops about DevOpsUse. In one of them, we presented Requirements Bazaar as well as the concept of gamification. The participating doctoral students then got screenshots of various aspects of our Web application and were

asked to think about possible means of gamification. The results were documented within Requirements Bazaar. Later, a bachelor student developed and evaluated some of the ideas in a bachelor thesis [Gott17].

The work on visual analytics of informal learning and the SWEVA prototype were featured in parts at the Immersive Learning conference series and in two journal issues [KoKl15, KoKl17]. Finally, an article was published at a special issue of the Computers in Human Behavior journal, which is an outlet at the cross-section of computer science and psychology [KoKl18].

In the Appendix B of this thesis, we included a section of all project deliverables we contributed to and where parts of this dissertation have been discussed.

Human Computer Interaction: In the human computer interaction community, we participated at a doctoral seminar at the Mensch-Computer-Interaktion conference 2016 in Aachen, Germany. We discussed the DevOpsUse approach and received feedback and the suggestion, to follow up on the research concerning involving larger crowds of end users in the information system design process. This has led to the work towards crowdsourcing for designers.

Another contribution has been on the Human Computer Interaction International (HCII) conference in 2019 [KAKo19]. We presented our work on immersive analytics as a new field within mixed reality, human activity recognition and machine learning. The work was performed in the context of the European WEKIT project and a master thesis. We gave an outlook and preliminary findings in how to embed mixed reality into a community-driven immersive analytics life cycle, thereby following the DevOpsUse model.

Software Engineering: In the field of software engineering we published at the prestigious ICSE conference, and at ICGSE, which feature a partially overlapping community. Our open source software approaches were presented at several occasions at the yearly FOSDEM open source conference, one of the world's largest gathering of open source projects. Our appearances and discussions helped directing the focus of our software projects; additionally we were able to gather useful feedback on how to extend our tools. The open source contributions are highlighted in detail in the next section.

8.1.2 Open Source Applications and Libraries

Following the presented open innovation principles discussed in Chapter 3, we pursued an open source approach that included all the prototypes developed in project, bachelor and master work. This goes in line with both the artifact creation and the communication phases of the design science in information systems guidelines [HMPR04]. Following our best practices set up in Section 7.2, the extensive open source communication approach was accompanied by several talks at large open source meetings like FOSDEM, FOSASIA, and within the XMPP community. Because of our early focus on collaboration as a service, manifested with a publication that compares various Web collaboration frameworks [KGKl13], and with the help of the Yjs library [Jahn16, Jahn19], we were able to include collaboration as a commodity functionality in most of the prototypes. Table 8.1 gives an overview of all the software developed in the course of this thesis.

Table 8.1: Open Source Software Repositories

Name	Description	Relevant Section
Requirements Bazaar	A Web-based continuous innovation platform. The hosted version has already attracted many communities from around the world.	3.2
House of Quality	An online Web application for creating HoQ diagrams. As far as we know, it is the only collaborative available.	3.2.1
Pharos	Crowdsourced co-design for involving large numbers of users in the design process.	3.3
Layers Box	Community-driven private cloud solution for self-hosting community-specific services.	5.2
OpenID Connect Web Component	An embeddable JavaScript library and HTML button that handles the authentication at any OIDC provider.	5.2.2
Direwolf	Collaborative modeling and transformation tool.	4.4
OakStreaming	WebTorrent-based peer-to-peer video streaming library.	5.3
SWEVA	A modularized Web framework for collaboratively creating and exploring visual analytics charts. The Web Components are reusable on third-party websites.	6.4
GaMR Framework	A gamified environment for mixed reality training. It is actively in use in medical education at Maastricht University.	3.4
Kanect	Machine learning human activity recognition and augmented reality player.	6.6

8.2 Future Work

While the research presented in this dissertation answered our initial research questions, the solutions and methods described open up several interesting research opportunities. Each of the core chapters gave pointers to possible directions that could be followed up in subsequent bachelor, master and PhD research or even whole project work. In the following, we discuss further possible strands of research based on the outcomes of this dissertation. An obvious task is the further communication of DevOpsUse, including its principle of end user involvement, the methodology and the tools developed. This applies to computer science education but also to other disciplines where agile approaches are applicable. Hereby, we consider the Internet of Production as particularly worthwile (cf. Section 2.7).

For the presented methodology of continuous innovation, we adopted several best practices from collaborative software development. We see a huge potential and impact in conveying further principles beyond versioning and distributed code repositories for societal use and goals. The current *citizen science* movement is likely to benefit greatly from retargeting our presented end user involvement support tools. Citizen science is defined as an open science approach, where ordinary non-expert persons observe, collect and share phenomena. The gathered findings can then be reviewed by experts, to be commented on in order to get better and more targeted reports in the future.

Out-of-the-box, Requirements Bazaar comes with tools that can support these processes. Requirements Bazaar is *themeable* in terms of colors and names and all labels are *localizable*; together the UI can be quickly adapted to new use cases.

Another way to bring DevOpsUse into a different context is automation. While there will be always parts where human input will remain significant and that thus cannot be automated, we see a huge challenge in discussing and defining the technical opportunities but also limits of automation through soft- and hardware. This is in particular important in the area of artificial intelligence; the social debate on this has already begun, in fact. With our research on crowdsourcing co-design, we have shown up in which direction this could go. We suggest to use more automation when the issue is about scaling up and getting more feedback from users. Here, further use of machine learning approaches could even answer the arisen issues with collecting mass feedback, by automatically classifying problem types from unstructured replies.

To a similar extent, peer-to-peer technologies have not yet reached their full potential for positively impacting societal structures. The extent to which regulatory and legislative forces criticize peer-to-peer approaches and even try to push them back shows what disruptive potential they can have. Technologies like blockchains and its application through *Bitcoin* threaten existing financial structures and thereby established international monetary safety mechanisms. Even on an infrastructural level, communities are currently rediscovering their fundamental power. For instance,the Long Range Wide Area Network (LoRaWAN) is a freely available wireless network protocol for the IoT, driven by the LoRa alliance [BlKu17]. While the technique is proprietary and patented, it relies on open source software. As LoRa uses license-free radio frequency bands, an ecosystem with commercial and community-driven gateways emerged over the last years, following the community principles of the older yet similar amateur radio.

The techniques of distributed computing can also be applied to the data platform. Current big data architectures aim to concentrate generated massive amounts of data at central locations. The recent concept of data mesh aims at the application of decentralization on the data platform, similar to distributed computing with microservices [Dehg19]. As data science becomes a commodity through extensive tool support, data scientists will have to take over more responsibilities in cross-functional teams together with operations experts. This will enable more innovative solutions beyond centralized data lakes. The presumed technologies like single sign-on solutions, increased automation and open interfaces have been well discussed in this thesis. We already see precursors of this movement with GraphQL built inside the Arango NoSQL database. Concerning software components and their deployment, we suggest further research into distributed deployment infrastructures. This is in particular valid for Internet of Things systems, were centralized approaches are not feasible due to the massive fringe of connected devices on the edge. Similarly, research concerning Web frontend deployments should take into consideration peer-to-peer delivery of parts of the user interface. While we have presented the Direwolf approach that goes in this direction, there are manifold opportunities in the areas of security and privacy, but also usability and acceptance.

Finally, new areas like mixed reality pose new opportunities on various levels. We discussed its potential in learning technologies and immersive analytics. We are convinced that principles at the core of DevOpsUse, like open innovation, automation, and cross-functional communities of practice contribute to unfolding the full societal potential of these and further applications.

Bibliography

[AIMo11] Atzori, Luigi, Iera, Antonio, and Morabito, Giacomo. Siot: Giving a Social Structure to the Internet of Things. *Communications Letters, IEEE*, 15(11):1193–1195, 2011. doi:10.1109/LCOMM.2011.090911.111340.

[AIMo14] Atzori, Luigi, Iera, Antonio, and Morabito, Giacomo. From "Smart Objects" to "Social Objects": The Next Evolutionary Step of the Internet of Things. *IEEE COMMUNICATIONS MAGAZINE*, 52(1):97–105, 2014. doi:10.1109/MCOM.2014.6710070.

[Akao90] Akao, Yoji. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press, Cambridge, 1990.

[Amer18] *American Heritage Dictionary of the English Language*. HOUGHTON MIFFLIN HARCOURT, 2018. ISBN 978-1328841698.

[Ande06] Anderson, Chris. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, New York, 2006. ISBN 1-4013-0237-8.

[Apac19] Apache. Apache Cordova, 2019. https://cordova.apache.org/.

[APB*99] Abrams, Marc, Phanouriou, Constantinos, Batongbacal, Alan L., Williams, Stephen M., and Shuster, Jonathan E. UIML: an appliance-independent XML user interface language. *Comput. Networks*, 31(11-16):1695–1708, 1999. doi:10.1016/S1389-1286(99)00044-4.

[Apia17] Apiary Inc. API Blueprint, 2017. https://apiblueprint.org/.

[Asyn19] AsyncAPI. AsyncAPI specification 2.0.0. https://www.asyncapi.com/docs/specifications/2.0.0/.

[Bart16] Bartels, Philipp. *WebTorrent-based Video Streaming on the Web*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, 2016.

[Bave14b] Bavendiek, Jens. XEP-0343: Use of DTLS/SCTP in Jingle ICE-UDP, 2014. http://xmpp.org/extensions/xep-0343.html.

[BBJ*18] Bergkvist, Adam, Burnett, Daniel C., Jennings, Cullen, Narayanan, Anant, Aboba, Bernard, Brandstetter, Taylor, and Bruaroey, Jan-Ivar. WebRTC 1.0: Real-time Communication Between Browsers: W3C Candidate Recommendation 27 September 2018, 2018. https://www.w3.org/TR/2016/WD-webrtc-20160128/.

[BCBM18] Billinghurst, Mark, Cordeil, Maxime, Bezerianos, Anastasia, and Margolis, Todd. Collaborative Immersive Analytics. In Marriott, Kim, Schreiber, Falk, Dwyer, Tim, Klein, Karsten, Riche, Nathalie Henry, Itoh, Takayuki, Stuerzlinger, Wolfgang, and Thomas, Bruce H. (Eds.), *Immersive Analytics*, vol. 11190, pp. 221–257. Springer International Publishing, Cham, 2018. ISBN 978-3-030-01387-5. doi: 10.1007/978-3-030-01388-2_8.

[BCC*17] Baldini, Ioana, Castro, Paul, Chang, Kerry, Cheng, Perry, Fink, Stephen, Ishakian, Vatche, Mitchell, Nick, Muthusamy, Vinod, Rabbah, Rodric, Slominski, Aleksander, and Suter, Philippe. Serverless Computing: Current Trends and Open Problems. In Chaudhary, Sanjay, Somani, Gaurav, and Buyya, Rajkumar (Eds.), *Research Advances in Cloud Computing*, pp. 1–20. Springer Singapore, Singapore, 2017. ISBN 978-981-10-5025-1.

[BCD*09] Berthold, Michael R., Cebron, Nicolas, Dill, Fabian, Gabriel, Thomas R., Kötter, Tobias, Meinl, Thorsten, Ohl, Peter, Thiel, Kilian, and Wiswedel, Bernd. KNIME - The Konstanz Information Miner. *SIGKDD Explor. Newsl.*, 11(1):26, 2009. doi: 10.1145/1656274.1656280.

[BCD*15] Becker, Christoph, Chitchyan, Ruzanna, Duboc, Leticia, Easterbrook, Steve, Penzenstadler, Birgit, Seyff, Norbert, and Venters, Colin C. Sustainability Design and Software: The Karlskrona Manifesto. In Bertolino, Antonia, Canfora, Gerardo, and Elbaum, Sebastian (Eds.), *Proceedings 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, ICSE 2015, pp. 467–476. IEEE Computer Society, Los Alamitos, CA, USA, 2015. ISBN 978-1-4799-1934-5. doi: 10.1109/ICSE.2015.179.

[BCFr17] Bernaschina, Carlo, Comai, Sara, and Fraternali, Piero. IFMLEdit.org: Model Driven Rapid Prototyping of Mobile Apps. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 207–208. 2017. doi: 10.1109/MOBILESoft.2017.15.

[BCFr18] Bernaschina, Carlo, Comai, Sara, and Fraternali, Piero. Formal semantics of OMG's Interaction Flow Modeling Language (IFML) for mobile and rich-client application model driven development. *J. Syst. Softw*, 137:239–260, 2018. doi:10.1016/j.jss.2017.11.067.

[BDIv17] Bødker, Susanne, Dindler, Christian, and Iversen, Ole Sejer. Tying Knots: Participatory Infrastructuring at Work. *Comput. Supported Coop. Work*, 26(1-2):245–273, 2017. doi:10.1007/s10606-017-9268-y.

[Beck03] Beck, Kent. *Extreme programming explained: Embrace change*. The XP series. Addison-Wesley, Boston, 10 th printing edn., 2003. ISBN 0-201-61641-6.

[Behr12] Behrendt, Malte. *Social Requirements Engineering in the ROLE Requirements Bazaar*. Master Thesis, RWTH Aachen University, Aachen, Germany, November 2012.

[BKGo11] Buhrmester, Michael, Kwang, Tracy, and Gosling, Samuel D. Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on psychological science : a journal of the Association for Psychological Science*, 6(1):3–5, 2011. doi:10.1177/1745691610393980.

[BKKl19] Bonilla Oliva, Delcy Carolina, Koren, István, and Klamma, Ralf. Infrastructuring for Crowdsourced Co-Design. *Interaction Design and Architecture (IxD&A)*, 2019.

[Blak15] Blake, M. Brian. Worrying about Wearables. *IEEE Internet Computing*, 19(5):4–5, 2015. doi:10.1109/MIC.2015.101.

[BlKu17] Blenn, Norbert and Kuipers, Fernando. LoRaWAN in the Wild: Measurements from The Things Network, 09062017. http://arxiv.org/pdf/1706.03086v1.

[BMVa19] Brito, Gleison, Mombach, Thais, and Valente, Marco Tulio. Migrating to GraphQL: A Practical Assessment. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 140–150. IEEE, 24022019 - 27022019. ISBN 978-1-7281-0591-8. doi:10.1109/SANER.2019.8667986.

[Bodk15b] Bødker, Susanne. Using IT to 'Do Good' in Communities? *The Journal of Community Informatics*, 11(2), 2015.

[Boni18] Bonilla Oliva, Delcy Carolina. *Infrastructuring for Crowdsourced Co-Design*. Master thesis, RWTH Aachen University, Aachen, Germany, 2018.

[Bosc17] Bosch. Bosch's IoT platform, 2017. https://www.bosch-si.com/de/iot-plattform/bosch-iot-suite/homepage-bosch-iot-suite.html.

[BoVa96] Bodart, François and Vanderdonckt, Jean. Widget Standardisation Through Abstract Interaction Objects. In *In Advances in Applied Ergonomics*, pp. 300–305. USA Publishing, 1996.

[BPLe14] Bauters, Merja, Purma, Jukka, and Leinonen, Teemu. In-Time On-Place Learning. In Sánchez, Inmaculada Arnedillo and Isaías, Pedro (Eds.), *Proceedings of the 10th International Conference on Mobile Learning*. IADIS Press, 2014.

[BPVa08] Buyya, Rajkumar, Pathan, Mukaddim, and Vakali, Athena. *Content Delivery Networks*, *Lecture notes in electrical engineering*, vol. 9. Springer-Verlag, Berlin, 2008. ISBN 978-3-540-77886-8.

[BrFr14] Brambilla, Marco and Fraternali, Piero. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. The MK/OMG press. Morgan Kaufmann, 2014. ISBN 978-0128001080.

[BrMa08] Bradwell, Peter and Marr, Sarah. Making the Most of Collaboration: An International Survey of Public Service Co-Design, 2008. https://lx.iriss.org.uk/sites/default/files/resources/Making%20the%20most.pdf.

[Broo96] Brooke, John. SUS: A Quick and Dirty Usability Scale. In Jordan, Patrick W., Thomas, Bruce, Weerdmeester, Bernard A., and McClelland, Ian Lyall (Eds.), *Usability Evaluation in Industry*, pp. 189–194. Taylor & Francis, 1996.

[Brya17] Bryant, Mike. GraphQL for archival metadata: An overview of the EHRI GraphQL API. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 2225–2230. IEEE, 12/11/2017 - 12/14/2017. ISBN 978-1-5386-2715-0. doi:10.1109/BigData.2017.8258173.

[CaGa14] Carretero, Jesús and García, J. Daniel. The Internet of Things: connecting the world. *Personal and Ubiquitous Computing*, 18(2):445–447, 2014. doi:10.1007/s00779-013-0665-z.

[Cass19] Cass, Stephen. Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectrum*, 56(5):16–17, 2019. doi:10.1109/MSPEC.2019.8701189.

[CCRN00] Carroll, John M., Chin, George, Rosson, Mary Beth, and Neale, Dennis C. The Development of Cooperation. In Boyarski, Daniel and Kellogg, Wendy A. (Eds.), *Proceedings of the conference on Designing interactive systems processes, practices, methods, and techniques - DIS '00*, pp. 239–251. ACM Press, New York, New York, USA, 2000. ISBN 1581132190. doi:10.1145/347642.347731.

[CCT*03] Calvary, Gaëlle, Coutaz, Joëlle, Thevenin, David, Limbourg, Quentin, Bouillon, Laurent, and Vanderdonckt, Jean. A Unifying Reference Framework for multi-target user interfaces. *Interacting with computers*, 15(3):289–308, 2003. doi:10.1016/S0953-5438(03)00010-9.

[CFF*03] Costabile, Maria Francesca, Fogjli, Daniela, Fresta, Giuseppe, Mussio, Piero, and Piccinno, Antonio. Building Environments for End-User Development and Tailoring. In *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003*, pp. 31–38. IEEE, Oct 28-31, 2003. ISBN 0-7803-8225-0. doi:10.1109/HCC.2003.1260199.

[Ches03] Chesbrough, Henry William. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press, Boston, MA, USA, 2003. ISBN 1578518377.

[Cisc17] Cisco Systems. The Zettabyte Era: Trends and Analysis: June 2017, 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf.

[CJAS06] Chinnici, Roberto, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 26072007. http://www.w3.org/TR/2007/REC-wsdl20-20070626.

[DaDi13] Le Dantec, Christopher A. and DiSalvo, Carl. Infrastructuring and the Formation of Publics in Participatory Design. *Social Studies of Science*, 43(2):241–264, 2013. doi:10.1177/0306312712471581.

[DaMa14] Daniel, Florian and Matera, Maristella. *Mashups: Concepts, Models and Architectures*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-55048-5. doi:10.1007/978-3-642-55049-2.

[DCJM19] Dizdarević, Jasenka, Carpio, Francisco, Jukan, Admela, and Masip-Bruin, Xavi. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Comput. Surv.*, 51(6):1–29, 2019. doi:10.1145/3292674.

[DDKN11] Deterding, Sebastian, Dixon, Dan, Khaled, Rilla, and Nacke, Lennart. From Game Design Elements to Gamefulness: Defining "Gamification". In Lugmayr, Artur, Franssila, Heljä, Safran, Christian, and Hammouda, Imed (Eds.), *the 15th International Academic MindTrek Conference*, p. 9. 2011. doi:10.1145/2181037.2181040.

[Dehg19] Dehghani, Zhamak. How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh, 2019. https://martinfowler.com/articles/data-monolith-to-mesh.html.

[DEKl12c] Derntl, Michael, Erdtmann, Stephan, and Klamma, Ralf. An Embeddable Dashboard for Widget-Based Visual Analytics on Scientific Communities. In *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '12, p. Article No. 23. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1242-4. doi:10.1145/2362456.2362486.

[DFHo16] Dalpiaz, Fabiano, Franch, Xavier, and Horkoff, Jennifer. iStar 2.0 Language Guide, 2016. http://arxiv.org/pdf/1605.07767v3.

[DGKl13] Derntl, Michael, Günnemann, Nikou, and Klamma, Ralf. A Dynamic Topic Model of Learning Analytics Research. In *LAK (Data Challenge)*. 2013.

[DHK*14] Derntl, Michael, Hannemann, Anna, Klamma, Ralf, Koren, István, Nicolaescu, Petru, Renzel, Dominik, Kravčík, Miloš, Shahriari, Mohsen, Purma, Jukka, Bachl, Martin, Bellamy, Edward, Elferink, Raymond, Tomberg, Vladimir, Theiler, Dieter, and Santos, Patricia. Customizable Architecture for Flexible Small-Scale Deployment: D6.2: Learning Layers Project Deliverable, 2014.

[DHKo08] Drachsler, Hendrik, Hummel, Hans G. K., and Koper, Rob. Personal recommender systems for learners in lifelong learning networks: the requirements, techniques and model. *International Journal of Learning Technology*, 3(4):404–423, 2008.

[DKK*13] Derntl, Michael, Klamma, Ralf, Koren, István, Kravčík, Miloš, Nicolaescu, Petru, Renzel, Dominik, Ngua, Kiarii, Purma, Jukka, Attwell, Graham, Gray, Owen, Ley, Tobias, Tomberg, Vladimir, Henry, Christina, Whitehead, Chris, Theiler, Dieter, Trattner, Christoph, Maier, Ronald K., Manhart, Markus, Schett, Maria, and Thalmann, Stefan. Initial Architecture for Fast Small-Scale Deployment: D6.1: Learning Layers Project Deliverable, 2013.

[DMGl07] Duvall, Paul M., Matyas, Steve, and Glover, Andrew. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007.

[DRHa11] Doan, AnHai, Ramakrishnan, Raghu, and Halevy, Alon Y. Crowdsourcing Systems on the World-Wide Web. *Communications of the ACM*, 54(4):86, 2011. doi:10.1145/1924421.1924442.

[DRN*15] Derntl, Michael, Renzel, Dominik, Nicolaescu, Petru, Koren, István, and Klamma, Ralf. Distributed Software Engineering in Collaborative Research Projects. In *Proceedings 2015 IEEE 10th International Conference on Global Software Engineering (ICGSE 2015)*, pp. 105–109. IEEE Computer Society, Los Alamitos, CA, USA, 2015. ISBN 978-1-4799-8409-1. doi:10.1109/ICGSE.2015.12.

[DuRa13] Dunne, Anthony and Raby, Fiona. *Speculative Everything: Design, Fiction, and Social Dreaming*. The MIT Press, Cambridge, Massachusetts and London, England, 2013. ISBN 978-0262019842.

[ECCa17] Ed-douibi, Hamza, Cánovas Izquierdo, Javier Luis, and Cabot, Jordi. Example-Driven Web API Specification Discovery. In Anjorin, Anthony and Espinoza, Huáscar (Eds.), *Modelling Foundations and Applications*, *Lecture Notes in Computer Science*, vol. 10376, pp. 267–284. Springer International Publishing, Cham, 2017. ISBN 978-3-319-61481-6. doi:10.1007/978-3-319-61482-3_16.

[ECCa18] Ed-douibi, Hamza, Cánovas Izquierdo, Javier Luis, and Cabot, Jordi. OpenAPItoUML: A Tool to Generate UML Models from OpenAPI Definitions. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 487–491. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi:10.1007/978-3-319-91662-0_41.

[Ecma15] Ecma International. ECMAScript® 2015 Language Specification, 2015. http://www.ecma-international.org/ecma-262/6.0/index.html.

[EGHS16] Ebert, Christof, Gallardo, Gorka, Hernantes, Josune, and Serrano, Nicolas. DevOps. *IEEE Softw*, 33(3):94–100, 2016. doi:10.1109/MS.2016.68.

[Ehn08] Ehn, Pelle. Participation in Design Things. In *Participatory Design Conference 2008*, pp. 92–101. 2008.

[Elec19] Electron. Electron, 2019. https://electronjs.org/.

[Enge05] Engeström, Yrjö. *Developmental Work Research: Expanding Activity Theory in Practice*, *International Cultural-historical Human Sciences*, vol. 12. Lehmanns Media, Berlin, 2005. ISBN 9783865410696.

[ERAn09] Eisenhauer, Markus, Rosengren, Peter, and Antolin, Pablo. A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems. In *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pp. 1–3. 2009. doi:10.1109/SAHCNW.2009.5172913.

[Euro09] European Commission. FP6 IST Impact Analysis Study: Final Report, 2009. http://cordis.europa.eu/fp7/ict/impact/documents/wing-pilot-fp6-final-report-18-12-09.pdf.

[Euro18] European Commission. Open source software strategy, 2018. https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en.

[Face18] Facebook Inc. GraphQL Specification: June 2018 Edition, 2018. https://graphql.github.io/graphql-spec/June2018.

[Fiel00] Fielding, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation, University of California, Irvine, Irvine, CA, USA, 2000.

[FMJo19] Ferrer, Ana Juan, Marquès, Joan Manuel, and Jorba, Josep. Towards the Decentralised Cloud. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019. doi:10.1145/3243929.

[FNC*16] de Freitas, Adrian A., Nebeling, Michael, Chen, Xian Anthony, Yang, Junrui, Ranithangam, Akshaye Shreenithi Kirupa Karthikeyan, and Dey, Anind K. Snap-To-It: A User-Inspired Platform for Opportunistic Device Interactions. In *Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems (CHI'16)*. 2016. ISBN 978-1-4503-3362-7/16/05. doi:10.1145/2858036.2858177.

[FoHi01] Fowler, Martin and Highsmith, Jim. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[FoLe14] Fowler, Martin and Lewis, James. Microservices, 2014. http://martinfowler.com/articles/microservices.html.

[Gaff15] Gaff, Brian M. Legal Issues with Wearable Technology. *IEEE Computer*, 48(9):10–12, 2015. doi:10.1109/MC.2015.280.

[GBMP13] Gubbi, Jayavardhana, Buyya, Rajkumar, Marusic, Slaven, and Palaniswami, Marimuthu. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GDAd15] Groen, Eduard C., Doerr, Joerg, and Adam, Sebastian. Towards Crowd-Based Requirements Engineering A Research Preview. In Fricker, Samuel A. and Schneider, Kurt (Eds.), *Requirements Engineering: Foundation for Software Quality*, *Lecture Notes in Computer Science*, vol. 9013, pp. 247–253. Springer International Publishing, Cham, 2015. ISBN 978-3-319-16100-6. doi:10.1007/978-3-319-16101-3_16.

[GoMe03] Gorlenko, Lada and Merrick, Roland. No wires attached: Usability challenges in the connected mobile world. *IBM Systems Journal*, 42(4):639–651, 2003. doi:10.1147/sj.424.0639.

[Goog17b] Google. Gallery: Material Design, 2017. https://material.io/gallery/.

[Goog18] Google. Material Design Guidelines, 2018. https://material.io/guidelines/.

[Gott17] Gottschlich, Dirk Manuel. *Gamifizierung in der Softwareentwicklung zur Erhöhung der Nutzerbeteiligung*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, 2017.

[Grap19] GraphQL Foundation. GraphQL Best Practices, 2019. https://graphql.org/learn/best-practices.

[GrVa17] Grisot, Miria and Vassilakopoulou, Polyxeni. Re-Infrastructuring for eHealth: Dealing with Turns in Infrastructure Development. *Comput. Supported Coop. Work*, 26(1-2):7–31, 2017. doi:10.1007/s10606-017-9264-2.

[GSA*17] Groen, Eduard C., Seyff, Norbert, Ali, Raian, Dalpiaz, Fabiano, Doerr, Joerg, Guzman, Emitza, Hosseini, Mahmood, Marco, Jordi, Oriol, Marc, Perini, Anna, and Stade, Melanie. The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software*, 34(2):44–52, 2017. doi:10.1109/MS.2017.33.

[GTMW11] Guinard, Dominique, Trifa, Vlad, Mattern, Friedemann, and Wilde, Erik. From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices. In Uckelmann, Dieter, Harrison, Mark, and Michahelles, Florian (Eds.), *Architecting the Internet of Things*, pp. 97–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-19157-2_5.

[Guth14] Guth, Andreas. *Near Real-time Visual Community Analytics for XMPP-based Networks*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, December 2014.

[HaCl88] Hauser, John R. and Clausing, Don. The House of Quality. *Harvard Business Review*, 66(3):63–73, 1988.

[Hadl09] Hadley, Marc. Web Application Description Language, 2009. https://www.w3.org/Submission/wadl/.

[HaLu01] Hanseth, Ole and Lundberg, Nina. Designing Work Oriented Infrastructures. *Computer Supported Cooperative Work (CSCW)*, 10(3-4):347–372, 2001. doi:10.1023/A:1012727708439.

[Hanc13] Hancke, Philipp. XEP-0320: Use of DTLS-SRTP in Jingle Sessions, 2013. http://xmpp.org/extensions/xep-0343.html.

[Hann14b] Hannemann, Anna. *Requirements Management in Community-Oriented Software Development*. Ph.D. thesis, RWTH Aachen University, Aachen, 2014.

[HaPe18] Hartig, Olaf and Pérez, Jorge. Semantics and Complexity of GraphQL. In *The Web Conference 2018*, pp. 1155–1164. Association for Computing Machinery and International World Wide Web Conferences Steering Committee, New York and Geneva, op 2018. ISBN 978-1-4503-5639-8. doi:10.1145/3178876.3186014.

[HeYe07] He, Jiang and Yen, I-Ling. Adaptive User Interface Generation for Web Services. In *IEEE International Conference on e-Business Engineering (ICEBE'07)*, pp. 536–539. 2007. doi:10.1109/ICEBE.2007.82.

[Hipp86] Hippel, Eric von. Lead Users: A Source of Novel Product Concepts. *Management Science*, 32(7):791–805, 1986. doi:10.1287/mnsc.32.7.791.

[HKKH18] Hensen, Benedikt, Koren, István, Klamma, Ralf, and Herrler, Andreas. An Augmented Reality Framework for Gamified Learning. In Hancke, Gerhard, Spaniol, Marc, Osathanunkul, Kitisak, Unankard, Sayan, and Klamma, Ralf (Eds.), *Advances in Web-Based Learning – ICWL 2018*, vol. 11007, pp. 67–76. Springer International Publishing, Cham, 2018. ISBN 978-3-319-96564-2. doi:10.1007/978-3-319-96565-9_7.

[HLG*14] Heintz, Matthias M., Law, Effie Lai-Chong, Govaerts, Sten, Holzer, Adrian, and Gillet, Denis. Pdot. In Jones, Matt, Palanque, Philippe, Schmidt, Albrecht, and Grossman, Tovi (Eds.), *the extended abstracts of the 32nd annual ACM conference*, pp. 2581–2586. 2014. doi:10.1145/2559206.2581266.

[HLKl14] Hannemann, Anna, Liiva, Kristjan, and Klamma, Ralf. Navigation Support in Evolving Open-Source Communities by a Web-Based Dashboard. In Luis Corral, Alberti Sillitti, Giancarlo Succi, Jelena Vlasenko, and Anthony I. Wasserman (Eds.), *Open Source Software: Mobile Open Source Technologies*, *IFIP Advances in Information and Communication Technology*, vol. 427, pp. 11–20. Springer, Berlin Heidelberg, 2014. ISBN 978-3-642-55128-4. doi:10.1007/978-3-642-55128-4_2.

[HMPR04] Hevner, Alan R., March, Salvatore T., Park, Jinsoo, and Ram, Sudha. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[Howe06] Howe, Jeff. The Rise of Crowdsourcing. *Wired magazine*, 6(14), 2006.

[HPR*13] Ha, Kiryong, Pillai, Padmanabhan, Richter, Wolfgang, Abe, Yoshihisa, and Satyanarayanan, Mahadev. Just-in-Time Provisioning for Cyber Foraging. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 153–166. 2013. doi:10.1145/2462456.2464451.

[HuFa11] Humble, Jez and Farley, David. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, Upper Saddle River, NJ, 2. print edn., 2011. ISBN 978-0-321-60191-9.

[IBM17] IBM. Watson IoT Platform, 2017. https://www.ibm.com/internet-of-things/platform/watson-iot-platform/.

[IFTT18] IFTTT Inc. IFTTT, 2018. https://ifttt.com/.

[IKBL19] Idoine, Carlie, Krensky, Peter, Brethenoux, Erick, and Linden, Alexander. Magic Quadrant for Data Science and Machine Learning Platforms, 28012019. https://www.gartner.com/doc/reprints?id=1-65WC0O1&ct=190128&st=sb.

[InVi17] InVision. InVision: Digital Product Design, Workflow and Collaboration, 2017. https://www.invisionapp.com/.

[IqBa08] Iqbal, Shamsi T. and Bailey, Brian P. Effects of Intelligent Notification Management on Users and Their Tasks. In Czerwinski, Mary, Lund, Arnie, and Tan, Desney (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, p. 93. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-011-1. doi:10.1145/1357054.1357070.

[IvDi14] Iversen, Ole Sejer and Dindler, Christian. Sustaining Participatory Design Initiatives. *CoDesign*, 10(3-4):153–170, 2014. doi:10.1080/15710882.2014.963124.

[Jahn16] Jahns, Kevin. *Near Real-time Peer-to-peer Group Editing on Tree-like Data Structures*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, 2014.

[Jahn19] Jahns, Kevin. Yjs shared editing framework, 2019. https://yjs.dev.

[JJMy09] Jeusfeld, Manfred A., Jarke, Matthias, and Mylopoulos, John (Eds.). *Metamodeling for Method Engineering*. MIT Press, 2009. ISBN 9780262101080.

[JSFo18] JS Foundation. Node-RED, 2018. https://nodered.org/.

[KAF*08] Keim, Daniel A., Andrienko, Gennady, Fekete, Jean-Daniel, Görg, Carsten, Kohlhammer, Jörn, and Melançon, Guy. Visual Analytics: Definition, Process, and Challenges. In Kerren, Andreas, Stasko, John, Fekete, Jean-Daniel, and North, Chris (Eds.), *Information Visualization*, *LNCS*, vol. 4950, pp. 154–175. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-70955-8. doi:10.1007/978-3-540-70956-5_7.

[Kahn62] Kahn, Arthur B. Topological Sorting of Large Networks. *Communications of the ACM*, 5(11):558–562, 1962. doi:10.1145/368996.369025.

[KAKo19] Klamma, Ralf, Ali, Rizwan, and Koren, István. Immersive Community Analytics for Wearable Enhanced Learning. In Zaphiris, Panayiotis and Ioannou, Andri (Eds.), *Learning and Collaboration Technologies. Ubiquitous and Virtual Environments for Learning and Collaboration*, *Lecture Notes in Computer Science*, vol. 11591, pp. 162–174. Springer International Publishing, Cham, 2019. ISBN 978-3-030-21816-4. doi:10.1007/978-3-030-21817-1_13.

[KaSy04] Karasti, Helena and Syrjänen, Anna-Liisa. Artful Infrastructuring in Two Cases of Community PD. In Clement, Andrew and van den Besselaar, Peter (Eds.), *the eighth conference*, p. 20. 2004. doi:10.1145/1011870.1011874.

[KBKl14] Koren, István, Bavendiek, Jens, and Klamma, Ralf. DireWolf Goes Pack Hunting: A Peer-to-Peer Approach for Secure Low Latency Widget Distribution Using WebRTC. In Casteleyn, Sven, Rossi, Gustavo, and Winckler, Marco (Eds.), *Web Engineering*, LNCS, pp. 507–510. Springer International Publishing, Cham, Switzerland, 2014. ISBN 978-3-319-08244-8. doi:10.1007/978-3-319-08245-5_38.

[KDBu15] Khodadadi, Farzad, Dastjerdi, Amir Vahid, and Buyya, Rajkumar. Simurgh: A framework for effective discovery, programming, and integration of services exposed in IoT. In *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, pp. 1–6. 2015. doi:10.1109/RIOT.2015.7104910.

[KGKl13] Koren, István, Guth, Andreas, and Klamma, Ralf. Shared Editing on the Web: A Classification of Developer Support Libraries. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom 2013)*, pp. 468–477. 2013. doi:10.4108/icst.collaboratecom.2013.254097.

[KHKl18b] Koren, István, Hensen, Benedikt, and Klamma, Ralf. Co-Design of Gamified Mixed Reality Applications. In *Proceedings of the IEEE ISMAR 2018 Workshop on Creativity in Design with & for Mixed Reality*, pp. 315–317. 2018. doi: 10.1109/ISMAR-Adjunct.2018.00094.

[KJHR11] Klamma, Ralf, Jarke, Matthias, Hannemann, Anna, and Renzel, Dominik. Der Bazar der Anforderungen - Open Innovation in emergenten Communities. *Informatik-Spektrum*, 34(2):178–191, 2011. doi:10.1007/s00287-010-0516-5.

[KKEM10] Keim, Daniel A., Kohlhammer, Jörn, Ellis, Geoffrey, and Mansmann, Florian. *Mastering the Information Age: Solving Problems with Visual Analytics*. Florian Mansmann and Eurographics Association, Goslar, 2010. ISBN 3905673770.

[KKJa20] Koren, István, Klamma, Ralf, and Jarke, Matthias. Direwolf Model Academy: An Extensible Collaborative Modeling Framework on the Web. In Michael, Judith and Bork, Dominik (Eds.), *Modellierung 2020 Short, Workshop and Tools & Demo Papers*, pp. 213–216. 2020.

[Klam10c] Klamma, Ralf. *Social Software and Community Information Systems*. Habilitation, RWTH Aachen University, Aachen, Germany, 2010.

[Klam13] Klamma, Ralf. Community Learning Analytics – Challenges and Opportunities. In Wang, Jhing-Fa and Lau, Rynson W. H. (Eds.), *Advances in Web-Based Learning: ICWL 2013*, *Lecture Notes in Computer Science*, vol. 8167, pp. 284–293. Springer, Berlin, 2013. ISBN 978-3-642-41175-5. doi:10.1007/978-3-642-41175-5_29.

[KLHo07] Kooken, Jose, Ley, Tobias, and Hoog de, Robert. How Do People Learn at the Workplace? Investigating Four Workplace Learning Assumptions. In Duval, Erik, Klamma, Ralf, and Wolpers, Martin (Eds.), *Creating New Learning Experiences on a Global Scale*, *LNCS*, vol. 4753, pp. 158–171. Springer-Verlag, Berlin and Heidelberg, 2007. ISBN 978-3-540-75194-6. doi:10.1007/978-3-540-75195-3_12.

[KMMa02] Kristensson, Per, Magnusson, Peter R., and Matthing, Jonas. Users as a Hidden Resource for Creativity: Findings from an Experimental Study on User Involvement. *Creativity and Innovation Management*, 11(1):55–61, 2002. doi:10.1111/1467-8691.00236.

[KNKl15] Koren, István, Nicolaescu, Petru, and Klamma, Ralf. Collaborative Drawing Annotations on Web Videos. In Frasincar, Flavius, Houben, Geert-Jan, and Schwabe, Daniel (Eds.), *Engineering the Web in the Big Data Era*, *Lecture Notes in Computer Science*, vol. 9114, pp. 671–674. Springer, 2015. doi:10.1007/978-3-319-19890-3_54.

[KoKl15] Koren, István and Klamma, Ralf. Smart Ambient Learning with Physical Artifacts Using Wearable Technologies. In *Workshop Proceedings of the 11th International Conference on Intelligent Environments, Prague, Czech Republic, July 15-17, 2015*, pp. 325–332. 2015. doi:10.3233/978-1-61499-530-2-325.

[KoKl16] Koren, István and Klamma, Ralf. Smart Ambient Learning with Physical Artifacts Using Wearable Technologies. *EAI Endorsed Transactions on Future Intelligent Educational Environments*, 2(6), 2016. doi:10.4108/eai.27-6-2016.151526.

[KoKl16b] Koren, István and Klamma, Ralf. The Direwolf Inside You: End User Development for Heterogeneous Web of Things Appliances. In Bozzon, Alessandro, Cudre-Maroux, Philippe, and Pautasso, Cesare (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 9671, pp. 484–491. Springer International Publishing, Cham, 2016. ISBN 978-3-319-38790-1. doi:10.1007/978-3-319-38791-8_35.

[KoKl17] Koren, István and Klamma, Ralf. Community Learning Analytics with Industry 4.0 and Wearable Sensor Data. In Beck, Dennis, Allison, Colin, Morgado, Leonel, Pirker, Johanna, Khosmood, Foaad, Richter, Jonathon, and Gütl, Christian (Eds.), *Immersive Learning Research Network*, pp. 142–151. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60632-3. doi:10.1007/978-3-319-60633-0_12.

[KoKl18] Koren, István and Klamma, Ralf. Enabling visual community learning analytics with Internet of Things devices. *Computers in Human Behavior*, 89:385–394, 2018. doi:10.1016/j.chb.2018.07.036.

[KoKl18b] Koren, István and Klamma, Ralf. Generation of Web Frontends from API Documentation with Direwolf Interaction Flow Designer. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 492–495. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi:10.1007/978-3-319-91662-0_42.

[KoKl18c] Koren, István and Klamma, Ralf. Peer-to-Peer Video Streaming in HTML5 with WebTorrent. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 404–419. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi:10.1007/978-3-319-91662-0_33.

[KoKl18d] Koren, István and Klamma, Ralf. The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. In Champin, Pierre-Antoine, Gandon, Fabien, Gandon, Fabien, Lalmas, Mounia, and Ipeirotis, Panagiotis G. (Eds.), *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*, pp. 781–787. ACM Press, New York, New York, USA, 2018. ISBN 9781450356404. doi:10.1145/3184558.3188740.

[KoKl19] Koren, István and Klamma, Ralf. OakStreaming: A Peer-to-PeerVideo Streaming Library. *Journal of Web Engineering*, 17(6):527–560, 2019. doi:10.13052/jwe1540-9589.17675.

[Kova14] Kovachev, Dejan. *Mobile Multimedia Services in the Cloud*. Ph.D. thesis, RWTH Aachen University, Aachen, 2014.

[Kraus18] Krause, Justin Judd-Jean. *Machine Vision im Umfeld der Mensch-Roboter-Kollaboration auf Basis der Microsoft Kinect*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, 2018.

[KRLJ16] Klamma, Ralf, Renzel, Dominik, de Lange, Peter, and Janßen, Holger. las2peer – A Primer, 2016. doi:10.13140/RG.2.2.31456.48645.

[KRN*14] Kovachev, Dejan, Renzel, Dominik, Nicolaescu, Petru, Koren, István, and Klamma, Ralf. DireWolf: A Framework for Widget-based Distributed User Interfaces. *Journal of Web Engineering*, 13(3&4):203–222, 2014.

[KRNK13] Kovachev, Dejan, Renzel, Dominik, Nicolaescu, Petru, and Klamma, Ralf. DireWolf - Distributing and Migrating User Interfaces for Widget-Based Web Applications. In Florian, Daniel, Dolog, Peter, Li, Qing, and (Keine Angabe) (Eds.), *13th International Conference on Web Engineering (ICWE 2013)*, *LNCS*, vol. 7977, pp. 99–113. Springer, Berlin, 2013. ISBN 978-3-642-39199-6. doi:10.1007/978-3-642-39200-9_10.

[KSB*17] Krawiec, Piotr, Sosnowski, Maciej, Batalla, Jordi Mongay, Mavromoustakis, Constandinos X., Mastorakis, George, and Pallis, Evangelos. Survey on Technologies for Enabling Real-Time Communication in the Web of Things. In Batalla, Jordi Mongay, Mastorakis, George, Mavromoustakis, Constandinos X., and Pallis, Evangelos (Eds.), *Beyond the Internet of Things*, Internet of Things, pp. 323–339. Springer International Publishing, Cham, 2017. ISBN 978-3-319-50756-9. doi:10.1007/978-3-319-50758-3_13.

[Kuns07] Kunszt, Peter. Grid Middleware Development in Large International Projects - Experience and Recommendations. In *International Conference on Software Engineering Advances*, pp. 82–86. IEEE, 2007. ISBN 978-0-7695-2937-0. doi:10.1109/ICSEA.2007.37.

[Kus19] Kus, Dominik Adam. *Migration Paths from REST-based APIs to GraphQL – Formalization and Tool Support*. Bachelor Thesis, RWTH Aachen University, Aachen, Germany, 2019.

[Lada08] Ladas, Corey. *Scrumban: Essays on Kanban Systems for Lean Software Development*. Modus cooperandi lean series. 2008. ISBN 978-0-578-00214-9.

[LAGM12] Luigi Atzori, Antonio Iera, Giacomo Morabito, and Michele Nitti. The Social Internet of Things (SIoT) – When Social Networks Meet the Internet of Things: Concept, Architecture and Network Characterization. *Computer Networks*, 56(16):3594–3608, 2012. doi:10.1016/j.comnet.2012.07.010.

[LaHa99] Law, John and Hassard, John (Eds.). *Actor-Network Theory and After*. Blackwell Publishers, 1999. ISBN 0-631-211942.

[LaJa12] Labrinidis, Alexandros and Jagadish, H. V. Challenges and Opportunities with Big Data. *Proc. VLDB Endow*, 5(12):2032–2033, 2012. doi:10.14778/2367502.2367572.

[LCD*14] Ley, Tobias, Cook, John, Dennerlein, Sebastian, Kravcik, Milos, Kunzmann, Christine, Pata, Kai, Purma, Jukka, Sandars, John, Santos, Patricia, Schmidt, Andreas, Al-Smadi, Mohammad, and Trattner, Christoph. Scaling informal learning at the

workplace: A model and four designs from a large-scale design-based research effort. *British Journal of Educational Technology*, 45(6):1036–1048, 2014. doi: 10.1111/bjet.12197.

[LCRK12] Law, Effie Lai-Chong, Chatterjee, Arunangsu, Renzel, Dominik, and Klamma, Ralf. The Social Requirements Engineering (SRE) Approach to Developing a Large-Scale Personal Learning Environment Infrastructure. In Ravenscroft, Andrew, Lindstaedt, Stefanie, Delgado Kloos, Carlos, Hernández-Leo, Davinia, and Kloos, Carlos Delgado (Eds.), *21st Century Learning for 21st Century Skills*, *Lecture Notes in Computer Science*, vol. 7563, pp. 194–207. Springer Verlag and Springer, Berlin and Heidelberg, 2012. ISBN 978-3-642-33262-3. doi:10.1007/978-3-642-33263-0_16.

[LeSa09] Lewis, James R. and Sauro, Jeff. The Factor Structure of the System Usability Scale. In Hutchison, David, Kanade, Takeo, Kittler, Josef, Kleinberg, Jon M., Mattern, Friedemann, Mitchell, John C., Naor, Moni, Nierstrasz, Oscar, Pandu Rangan, C., Steffen, Bernhard, Sudan, Madhu, Terzopoulos, Demetri, Tygar, Doug, Vardi, Moshe Y., Weikum, Gerhard, and Kurosu, Masaaki (Eds.), *Human Centered Design*, *Lecture Notes in Computer Science*, vol. 5619, pp. 94–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02805-2. doi:10.1007/978-3-642-02806-9_12.

[LFK*14b] Lasi, Heiner, Fettke, Peter, Kemper, Hans-Georg, Feld, Thomas, and Hoffmann, Michael. Industrie 4.0. *Wirtschaftsinformatik*, 56(4):261–264, 2014. doi:10.1007/s11576-014-0424-4.

[LGLo10] Lakhani, Karim R., Garvin, David A., and Lonstein, Eric. Topcoder (A): Developing Software Through Crowdsourcing. *Harvard Business School General Management Unit Case*, (610-032), 2010.

[LiDo17] Lin, Zhiting and Dong, Liang. Clarifying Trust in Social Internet of Things. *IEEE Transactions on Knowledge and Data Engineering*, p. 1, 2017. doi:10.1109/TKDE.2017.2762678.

[LiYi07] Li, Bo and Yin, Hao. Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges. *Communications Magazine, IEEE*, 45(6):94–99, 2007. doi:10.1109/MCOM.2007.374425.

[LMC*16] La Torre, Giuseppe, Monteleone, Salvatore, Cavallo, Marco, D'Amico, Valeria, and Catania, Vincenzo. A Context-Aware Solution to Improve Web Service Discovery and User-Service Interaction. In El Baz, Didier and Bourgeois, Julien (Eds.), *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, pp. 180–187. IEEE Computer Society, 2016. ISBN 978-1-5090-2771-2. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0047.

[LNKK16] de Lange, Peter, Nicolaescu, Petru, Klamma, Ralf, and Koren, István. DevOpsUse for Rapid Training of Agile Practices Within Undergraduate and Startup Commu-

nities. In Verbert, Katrien, Sharples, Mike, and Klobučar, Tomaž (Eds.), *Adaptive and Adaptable Learning*, *Lecture Notes in Computer Science*, vol. 9891, pp. 570–574. Springer International Publishing, Cham, 2016. ISBN 978-3-319-45152-7. doi:10.1007/978-3-319-45153-4_65.

[LPKW06] Lieberman, Henry, Paternò, Fabio, Klann, Markus, and Wulf, Volker. End-User Development: An Emerging Paradigm. In Lieberman, Henry, Paternò, Fabio, and Wulf, Volker (Eds.), *End User Development*, *Human-Computer Interaction Series*, vol. 9, pp. 1–8. Springer, Dordrecht, 2006. ISBN 1-4020-5309-6. doi:10.1007/1-4020-5386-X_1.

[LTT*17] Lindley, Siân E., Thieme, Anja, Taylor, Alex S., Vlachokyriakos, Vasilis, Regan, Tim, and Sweeney, David. Surfacing Small Worlds through Data-In-Place. *Computer Supported Cooperative Work (CSCW)*, 26(1-2):135–163, 2017. doi:10.1007/s10606-017-9263-3.

[LTW*15] Luther, Kurt, Tolentino, Jari-Lee, Wu, Wei, Pavel, Amy, Bailey, Brian P., Agrawala, Maneesh, Hartmann, Björn, and Dow, Steven P. Structuring, Aggregating, and Evaluating Crowdsourced Design Critique. In Cosley, Dan, Forte, Andrea, Ciolfi, Luigina, and McDonald, David (Eds.), *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15*, pp. 473–485. ACM Press, New York, New York, USA, 2015. ISBN 9781450329224. doi:10.1145/2675133.2675283.

[LVJ*19] Limbu, Bibeg, Vovk, Alla, Jarodzka, Halszka, Klemke, Roland, Wild, Fridolin, and Specht, Marcus. WEKIT.One: A Sensor-Based Augmented Reality System for Experience Capture and Re-enactment. In Scheffel, Maren, Broisin, Julien, Pammer-Schindler, Viktoria, Ioannou, Andri, and Schneider, Jan (Eds.), *Transforming Learning with Meaningful Technologies*, *Lecture Notes in Computer Science*, vol. 11722, pp. 158–171. Springer International Publishing, Cham, 2019. ISBN 978-3-030-29735-0. doi:10.1007/978-3-030-29736-7_12.

[LVM*04] Limbourg, Quentin, Vanderdonckt, Jean, Michotte, Benjamin, Bouillon, Laurent, and López-Jaquero, Víctor. USIXML: a Language Supporting Multi-Path Development of User Interfaces. In Bastide, Rémi, Palanque, Philippe, and Roth, Jörg (Eds.), *Engineering Human Computer Interaction and Interactive Systems*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26097-4. doi:10.1007/11431879_12.

[MaBo17] Marttila, Sanna and Botero, Andrea. Infrastructuring for Cultural Commons. *Comput. Supported Coop. Work*, 26(1-2):97–133, 2017. doi:10.1007/s10606-017-9273-1.

[MAHa15] McIlroy, Stuart, Ali, Nasir, and Hassan, Ahmed E. Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store. *Empirical Software Engineering*, 21(3):1346–1370, 2016. doi:10.1007/s10664-015-9388-2.

[Marc11] Marcotte, Ethan. *Responsive Web Design*. A Book Apart, New York, 2011. ISBN 978-0984442577.

[Mass12] Massé, Mark. *REST API design rulebook*. O'Reilly, Farnham, 2012. ISBN 9781449310509.

[McCa13] McEwen, Aian and Cassimally, Hakim. *Designing the Internet of Things*. Wiley, Hoboken, New Jersey, USA, 2013. ISBN 978-1-118-43065-1.

[MCJ*15] Mikusz, Mateusz, Clinch, Sarah, Jones, Rachel, Harding, Mike, Winstanley, Christopher, and Davies, Nigel. Repurposing Web Analytics to Support the IoT. *Computer*, 48(9):42–49, 2015. doi:10.1109/MC.2015.260.

[Meyn12] Meyn, Antony J. R. *Browser to Browser Media Streaming with HTML5*. Master Thesis, Aalto University, 2012.

[Micr19] Microsoft. HoloLens 2 - Overview, Features and Specs | Microsoft HoloLens, 2019. https://www.microsoft.com/en-us/hololens/hardware.

[MiKi94] Milgram, Paul and Kishino, Fumio. A Taxonomy of Mixed Reality Visual Displays. *IEICE Transactions on Information Systems*, E77-D(E77-D No. 12):1321–1329, 1994.

[MMPa02] Molina, Pedro J., Meliá, Santiago, and Pastor, Oscar. JUST-UI: A User Interface Specification Model. In Kolski, Christophe and Vanderdonckt, Jean (Eds.), *Computer-Aided Design of User Interfaces III*, vol. 4, December, pp. 63–74. Springer Netherlands, Dordrecht, 2002. ISBN 978-94-010-3915-4. doi:10.1007/978-94-010-0421-3_5.

[MoHe17] Mossner, Christian and Herhausen, Dennis. Video: the New Rules of Communication. *Marketing Review St. Gallen*, 34(2):36–44, 2017.

[Moli19] Molina, Pedro J. Quid: Prototyping Web Components on the Web. In Panach, Ignacio, Vanderdonckt, Jean, and Pastor, Óscar (Eds.), *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '19*, pp. 1–5. ACM Press, New York, New York, USA, 2019. ISBN 9781450367455. doi:10.1145/3319499.3330294.

[MoMa18] Moulay, Mohamed and Mancuso, Vincenzo. Experimental Performance Evaluation of WebRTC Video Services over Mobile Networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 541–546. IEEE, 15042018 - 19042018. ISBN 978-1-5386-5979-3. doi:10.1109/INFCOMW.2018.8407020.

[Mule17] MuleSoft, Inc. RAML, 2017. https://raml.org/.

[MWDT07] Maximilien, E. Michael, Wilkinson, Hernan, Desai, Nirmit, and Tai, Stefan. A Domain-Specific Language for Web APIs and Services Mashups. In Krämer, BerndJ J., Lin, Kwei-Jay, and Narasimhan, Priya (Eds.), *Service-Oriented Computing – ICSOC*, *Lecture Notes in Computer Science*, vol. 4749, pp. 13–26. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74973-8. doi:10.1007/978-3-540-74974-5_2.

[Myer16] Myerson, Jeremy. Scaling Down: Why Designers Need to Reverse Their Thinking. *She Ji: The Journal of Design, Economics, and Innovation*, 2(4):288–299, 2016. doi: 10.1016/j.sheji.2017.06.001.

[NCFD06] Nicholson, Daren T., Chalk, Colin, Funnell, W Robert J, and Daniel, Sam J. Can virtual reality improve anatomy education? A randomised controlled study of a computer-generated three-dimensional anatomical ear model. *Medical education*, 40(11):1081–1087, 2006. doi:10.1111/j.1365-2929.2006.02611.x.

[NDMN14] Norrie, Moira C., Di Geronimo, Linda, Murolo, Alfonso, and Nebeling, Michael. The Forgotten Many? A Survey of Modern Web Development Practices. In Casteleyn, Sven, Rossi, Gustavo, and Winckler, Marco (Eds.), *Web Engineering*, *LNCS*, vol. 8541, pp. 290–307. Springer International Publishing, Cham, Switzerland, 2014. ISBN 978-3-319-08244-8. doi:10.1007/978-3-319-08245-5_17.

[Newm15] Newman, Sam. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, Sebastopol, CA, USA, 2015. ISBN 9781491950357.

[NFH*10] Nestler, Tobias, Feldmann, Marius, Hübsch, Gerald, Preußner, André, and Jugel, Uwe. The ServFace Builder - A WYSIWYG Approach for Building Service-Based Applications. In Benatallah, Boualem, Casati, Fabio, Kappel, Gerti, and Rossi, Gustavo (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 6189, pp. 498–501. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13910-9. doi: 10.1007/978-3-642-13911-6_37.

[NKR*14] Nussbaumer, Alexander, Kravčík, Miloš, Renzel, Dominik, Klamma, Ralf, Berthold, Marcel, and Albert, Dietrich. A Framework for Facilitating Self-Regulation in Responsive Open Learning Environments. *CoRR*, abs/1407.5891, 2014.

[NMJ*13] Nurminen, Jukka K., Meyn, Antony J. R., Jalonen, Eetu, Raivio, Yrjo, and Marrero, Raúl Garcıa. P2P Media Streaming with HTML5 and WebRTC. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 63–64. 2013. doi:10.1109/INFCOMW.2013.6970739.

[NNAn10] Namoun, Abdallah, Nestler, Tobias, and De Angeli, Antonella. Conceptual and Usability Issues in the Composable Web of Software Services. In Daniel, Florian and Facca, Federico Michele (Eds.), *Current Trends in Web Engineering: 10th International Conference on Web Engineering*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16984-7 // 978-3-642-16985-4. doi:10.1007/978-3-642-16985-4_35.

[NNAn10b] Namoun, Abdallah, Nestler, Tobias, and De Angeli, Antonella. Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations. In *2010 IEEE 8th European Conference on Web Services (ECOWS)*, pp. 123–130. 2010. doi: 10.1109/ECOWS.2010.17.

[NoGo14] Nogueira Barbosa, Flávio Ribeiro and Gomes Soares, Luiz Fernando. Towards the Application of WebRTC Peer-to-Peer to Scale Live Video Streaming over the Internet. In *Simposio Brasileiro de Redes de Computadores (SBRC)*. 2014.

[NRK*14] Nicolaescu, Petru, Renzel, Dominik, Koren, István, Klamma, Ralf, Purma, Jukka, and Bauters, Merja. A Community Information System for Ubiquitous Informal Learning Support. In *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*, pp. 138–140. 2014. doi:10.1109/ICALT.2014.48.

[OASI07] OASIS Web Services Business Process Execution Language (WSBPEL) TC. Web Services Business Process Execution Language Version 2.0, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[Open18] OpenAPI Initiative. The OpenAPI Specification: Version 3.0.2, 2018. https://www.openapis.org.

[Orli10] Orlikowski, Wanda J. The Sociomateriality of Organisational Life: Considering Technology in Management Research. *Cambridge Journal of Economics*, 34(1):125–141, 2010. doi:10.1093/cje/bep058.

[OrRo91] Orlikowski, Wanda J. and Robey, Daniel. Information Technology and the Structuring of Organizations. *Information Systems Research*, 2(2):143–169, 1991.

[PaMa13] Pagano, Dennis and Maalej, Walid. User Feedback in the AppStore: An Empirical Study. In *21st IEEE International Requirements Engineering Conference: RE 2013*, pp. 125–134. IEEE, Piscataway, NJ, 2013. ISBN 978-1-4673-5765-4. doi:10.1109/RE.2013.6636712.

[Parm17] Parmiggiani, Elena. This Is Not a Fish: On the Scale and Politics of Infrastructure Design Studies. *Comput. Supported Coop. Work*, 26(1-2):205–243, 2017. doi:10.1007/s10606-017-9266-0.

[PaSa17] Paternò, Fabio and Santoro, Carmen. A Design Space for End User Development in the Time of the Internet of Things. In Paternò, Fabio and Wulf, Volker (Eds.), *New Perspectives in End-User Development*, vol. 25, pp. 43–59. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60290-5. doi:10.1007/978-3-319-60291-2_3.

[Pate00] Paternò, Fabio. *Model-Based Design and Evaluation of Interactive Applications*, *Springer-Verlag series on applied computing*, vol. 2000: 2. Springer, London, 2000. ISBN 1-85233-155-0.

[PaZi17] Pautasso, Cesare and Zimmermann, Olaf. The Web as a Software Connector: Integration Resting on Linked Resources. *IEEE Software*, 35(1):93–98, 2017. doi:10.1109/MS.2017.4541049.

[PeCa14] Peters, Max and ten Cate, Olle. Bedside teaching in medical education: a literature review. *Perspectives on medical education*, 3(2):76–88, 2014. doi:10.1007/s40037-013-0083-y.

[PeMa17] Peón-Quirós, Miguel and Mancuso, Vincenzo. MONROE Measuring Mobile Broadband Networks in Europe: Deliverable D1.3 Final Implementation, 28022017. https://www.monroe-project.eu/wp-content/uploads/2015/12/Attachment_0-19.pdf.

[PGH*19] Pennekamp, Jan, Glebke, Rene, Henze, Martin, Meisen, Tobias, Quix, Christoph, Hai, Rihan, Gleim, Lars, Niemietz, Philipp, Rudack, Maximilian, Knape, Simon, Epple, Alexander, Trauth, Daniel, Vroomen, Uwe, Bergs, Thomas, Brecher, Christian, Buhrig-Polaczek, Andreas, Jarke, Matthias, and Wehrle, Klaus. Towards an Infrastructure Enabling the Internet of Production. In *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, pp. 31–37. IEEE, 06052019 - 09052019. ISBN 978-1-5386-8500-6. doi:10.1109/ICPHYS.2019.8780276.

[PGP*13] Papagiannaki, Konstantina, Gummadi, Krishna, Partridge, Craig, Zhao, Mingchen, Aditya, Paarijaat, Chen, Ang, Lin, Yin, Haeberlen, Andreas, Druschel, Peter, Maggs, Bruce, Wishon, Bill, and Ponec, Miroslav. Peer-assisted content distribution in Akamai netsession. In *IMC '13 Proceedings of the 2013 Internet Measurement Conference*, pp. 31–42. 2013. doi:10.1145/2504730.2504752.

[PHSh10] Patni, Harshal, Henson, Cory, and Sheth, Amit. Linked Sensor Data. In *2010 International Symposium on Collaborative Technologies and Systems*, pp. 362–370. 2010. doi:10.1109/CTS.2010.5478492.

[PIP*18] Papadopoulos, Panagiotis, Ilia, Panagiotis, Polychronakis, Michalis, Markatos, Evangelos P., Ioannidis, Sotiris, and Vasiliadis, Giorgos. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation, 30092018. http://arxiv.org/pdf/1810.00464v1.

[PiSy06] Pipek, Volkmar and Syrjänen, Anna-Liisa. Infrastructuring as Capturing In-Situ Design. In *7th Mediterranean Conference on Information Systems*. 2006.

[PiWu09] Pipek, Volkmar and Wulf, Volker. Infrastructuring: Towards an Integrated Perspective on the Design and Use of Information Technology. *Journal of the Association for Information Systems*, 10(5):447–473, 2009.

[PMMe97] Paterno, F., Mancini, C., and Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Howard, Steve, Hammond, Judy, and Lindgaard, Gitte (Eds.), *Human-Computer Interaction INTERACT '97*, pp. 362–369. Springer US, Boston, MA, 1997. ISBN 978-1-4757-5437-7. doi:10.1007/978-0-387-35175-9_58.

[Pohl10] Pohl, Klaus. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, Heidelberg and New York, 2010. ISBN 9783642125775.

[PRSU16] Pezoa, Felipe, Reutter, Juan L., Suarez, Fernando, Ugarte, Martín, and Vrgoč, Domagoj. Foundations of JSON Schema. In Bourdeau, Jacqueline, Hendler, Jim A., Nkambou, Roger Nkambou, Horrocks, Ian, and Zhao, Ben Y. (Eds.), *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pp. 263–273. ACM Press, New York, New York, USA, 2016. ISBN 9781450341431. doi:10.1145/2872427.2883029.

[PuEi02] Puerta, Angel and Eisenstein, Jacob. XIML: A Common Representation for Interaction Data. In Hammond, Kristian, Gil, Yolanda, and Leake, David (Eds.), *Proceedings of the 7th international conference on Intelligent user interfaces - IUI '02*,

p. 214. ACM Press, New York, New York, USA, 2002. ISBN 1581133820. doi: 10.1145/502716.502763.

[PZCG14] Perera, Charith, Zaslavsky, Arkady, Christen, Peter, and Georgakopoulos, Dimitrios. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014. doi:10.1109/SURV.2013.042313.00197.

[RBKJ13] Renzel, Dominik, Behrendt, Malte, Klamma, Ralf, and Jarke, Matthias. Requirements Bazaar: Social Requirements Engineering for Community-Driven Innovation. In *21st IEEE International Requirements Engineering Conference: RE 2013*, pp. 326–327. IEEE, Piscataway, NJ, 2013. ISBN 978-1-4673-5765-4. doi:10.1109/RE.2013.6636738.

[RCCa17] Rodriguez-Echeverria, Roberto, Cánovas Izquierdo, Javier Luis, and Cabot, Jordi. Towards a UML and IFML Mapping to GraphQL. In Cabot, Jordi, de Virgilio, Roberto, and Torlone, Riccardo (Eds.), *Web Engineering*, vol. 10544, pp. 149–155. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60130-4. doi: 10.1007/978-3-319-74433-9_13.

[ReKl14b] Renzel, Dominik and Klamma, Ralf (Eds.). *Large-Scale Social Requirements Engineering*, vol. 2. IEEE Special Technical Community on Social Networking (IEEE STCSN), 2014.

[ReMa17] Reiter, Andreas and Marsalek, Alexander. WebRTC: Your Privacy is at Risk. In *Proceedings of the Symposium on Applied Computing*, pp. 664–669. ACM, Marrakech, Morocco, 2017. ISBN 978-1-4503-4486-9. doi:10.1145/3019612.3019844.

[Renz12] Renzel, Dominik. An Introduction to ROLE Interwidget Communication, 2012. http://dbis.rwth-aachen.de/gadgets/iwc/resources/iwc.manual.pdf.

[Renz16] Renzel, Dominik. *Information Systems Success Awareness for Professional Long Tail Communities of Practice*. Ph.D. thesis, RWTH Aachen University, Aachen, Germany, 2016.

[RKJa15] Renzel, Dominik, Klamma, Ralf, and Jarke, Matthias. IS Success Awareness in Community-Oriented Design Science Research. In Donnellan, Brian, Helfert, Markus, Kenneally, Jim, VanderMeer, Debra, Rothenberger, Marcus, and Winter, Robert (Eds.), *New Horizons in Design Science: Broadening the Research Agenda*, *LNCS*, vol. 9073, pp. 413–420. Springer International Publishing, Switzerland, 2015. ISBN ISBN 978-3-319-18714-3. doi:10.1007/978-3-319-18714-3_33.

[RKJW07] Rohde, Markus, Klamma, Ralf, Jarke, Matthias, and Wulf, Volker. Reality is our Laboratory: Communities of practice in applied computer science. *Behaviour and Information Technology*, 26(1):81–94, 2007.

[RKKJ17] Renzel, Dominik, Koren, István, Klamma, Ralf, and Jarke, Matthias. Preparing Research Projects for Sustainable Software Engineering in Society. In *Proceedings 2017*

IEEE/ACM 39th IEEE International Conference on Software Engineering (ICSE). 2017. ISBN 978-1-5386-1589-8. doi:10.1109/ICSE-SEIS.2017.4.

[Roge03] Rogers, Everett M. *Diffusion of Innovations: 5th ed.* Free Press, New York, 2003. ISBN 978-0743222099.

[RoHo14] Roverso, Roberto and Hogqvist, Mikael. Hive.js: Browser-Based Distributed Caching for Adaptive Video Streaming. In *IEEE International Symposium on Multimedia*, pp. 143–146. 2014. doi:10.1109/ISM.2014.45.

[RoLe04] Rose, Daniel E. and Levinson, Danny. Understanding User Goals in Web Search. In *The 13th International World Wide Web Conference*, WWW '04, p. 13. Association for Computing Machinery, New York, 2004. ISBN 1-58113-844-X. doi:10.1145/988672.988675.

[RSKl12] Renzel, Dominik, Schlebusch, Patrick, and Klamma, Ralf. Today's Top "RESTful" Services and Why They Are Not RESTful. In Wang, X. Sean, Cruz, Isabel, Delis, Alex, and Huang, Guangyan (Eds.), *Web Information Systems Engineering (WISE 2012)*, Lecture Notes in Computer Science, pp. 354–367. Springer and Springer Berlin Heidelberg, Heidelberg, Berlin, 2012. ISBN 978-3-642-35063-4. doi:10.1007/978-3-642-35063-4_26.

[Rupp16] Ruppert, Alexander. *An Extensible Visual Analytics Platform for Software Development Communities*. Master Thesis, RWTH Aachen University, Aachen, Germany, 2016.

[RVP*14] Rhinow, Florian, Veloso, Pablo Porto, Puyelo, Carlos, Barrett, Stephen, and Nuallain, Eamonn O. P2P Live Video Streaming in WebRTC. In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, pp. 1–6. 2014. doi:10.1109/WCCAIS.2014.6916588.

[RWTH18] RWTH Aachen University. Internet of Production: Cluster of Excellence Funding Line | Proposal, 2018.

[SAAS18] Sulema, Yevgeniya, Amram, Noam, Aleshchenko, Oleksii, and Sivak, Olena. Quality of Experience Estimation for WebRTC-based Video Streaming. In *European Wireless 2018; 24th European Wireless Conference*. VDE Verlag GmbH, Berlin, Germany, 2018. ISBN 978-3-8007-4560-9.

[SAMa15b] Sohan, S. M., Anslow, Craig, and Maurer, Frank. SpyREST: Automated RESTful API Documentation Using an HTTP Proxy Server (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 271–276. IEEE, 09112015 - 13112015. ISBN 978-1-5090-0025-8. doi:10.1109/ASE.2015.52.

[SAMa17] Sohan, S M, Anslow, Craig, and Maurer, Frank. Automated Example Oriented REST API Documentation at Cisco. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pp. 213–222. IEEE, 20052017 - 28052017. ISBN 978-1-5386-2717-4. doi:10.1109/ICSE-SEIP.2017.11.

[SaSi19] Sanctorum, Audrey and Signer, Beat. Towards end-user development of distributed user interfaces. *Universal Access in the Information Society*, 18(4):785–799, 2019. doi:10.1007/s10209-017-0601-5.

[SaSt08] Sanders, Elizabeth B.-N. and Stappers, Pieter Jan. Co-Creation and the New Landscapes of Design. *CoDesign*, 4(1):5–18, 2008. doi:10.1080/15710880701875068.

[Saty17] Satyanarayanan, Mahadev. The Emergence of Edge Computing. *Computer, title=Enhancing the User Experience in Mobile Phones*, 50(1):30–39, 2017. doi:10.1109/MC.2017.9.

[Saty19] Satyanarayanan, Mahadev. How we created edge computing. *Nature Electronics*, 2(1):42, 2019. doi:10.1038/s41928-018-0194-x.

[SBCD09] Satyanarayanan, Mahadev, Bahl, Paramvir, Cáceres, Ramón, and Davies, Nigel. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009. doi:10.1109/MPRV.2009.82.

[SBJ*14] Sakimura, Nat, Bradley, John, Jones, Michael B., de Medeiros, Breno, and Mortimore, Chuck. OpenID Connect Core 1.0, 2014. https://openid.net/specs/openid-connect-core-1_0.html.

[ScZi98] Schunk, Dale H. and Zimmerman, Barry J. (Eds.). *Self-Regulated Learning: From Teaching to Self-Reflective Practice*. Guilford Press, New York, 1998. ISBN 1572303069.

[SDSc18] Schuh, Günther, Dölle, Christian, and Schlößer, Elmar Sebastian. Agile Prototyping for technical systems : towards an adaption ot the Minimum Viable Product principle. In *DS 91 Proceedings of 13th biennial Norddesign Conference Design 2018, Linköping, Sweden, 14th- 17th August 2018*. LiU Tryck, Linköping, 2018.

[SGBS16] Schuh, Günther, Gartzen, Thomas, Basse, Felix, and Schrey, Elisabeth. Enabling Radical Innovation through Highly Iterative Product Expedition in Ramp up and Demonstration Factories. *Procedia CIRP*, 41:620–625, 2016. doi:10.1016/j.procir.2016.01.014.

[SGR*14] Schuster, Daniel, Grubitzsch, Philipp, Renzel, Dominik, Koren, István, Klauck, Ronny, and Kirsche, Michael. Global-Scale Federated Access to Smart Objects Using XMPP. In Bilof, Randall (Ed.), *Proceedings 2014 IEEE International Conference on Internet of Things (iThings 2014), Green Computing and Communications (GreenCom 2014), and Cyber-Physical-Social Computing (CPSCom 2014)*, pp. 185–192. IEEE Computer Society, Los Alamitos, CA, USA, 2014. ISBN CFP14GCC-CDR. doi:10.1109/iThings.2014.35.

[Shah18] Shahriari, Mohsen. *Detection and Analysis of Overlapping Community Structures for Modelling and Prediction in Complex Networks*. Dissertation, RWTH Aachen University, Aachen, Germany, 2018.

[Shne96] Shneiderman, Ben. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *1996 IEEE Symposium on Visual Languages*, pp. 336–343. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996. doi:10.1109/VL.1996.545307.

[SiPa03] da Silva, Paulo Pinheiro and Paton, Norman W. User Interface Modeling in UMLi. *IEEE Softw*, 20(4):62–69, 2003. doi:10.1109/MS.2003.1207457.

[SMKo11] Steen, Marc, Manschot, Menno, and de Koning, Nicole. Benefits of Co-design in Service Design Projects. *International Journal of Design*, 5(2):53–60, 2011.

[SOC*17] Stade, Melanie, Oriol, Marc, Cabrera, Oscar, Fotrousi, Farnaz, Schaniel, Ronnie, Seyff, Norbert, and Schmidt, Oleg. Providing a User Forum is not enough: First Experiences of a Software Company with CrowdRE. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pp. 164–169. IEEE, 04092017 - 08092017. ISBN 978-1-5386-3488-2. doi:10.1109/REW.2017.21.

[SPW*14] Schuh, Günther, Potente, Till, Wesch-Potente, Cathrin, Weber, Anja Ruth, and Prote, Jan-Philipp. Collaboration Mechanisms to Increase Productivity in the Context of Industrie 4.0. *Procedia CIRP*, 19:51–56, 2014. doi:10.1016/j.procir.2014.05.016.

[SSX*15] Satyanarayanan, Mahadev, Simoens, Pieter, Xiao, Yu, Pillai, Padmanabhan, Chen, Zhuo, Ha, Kiryong, Hu, Wenlu, and Amos, Brandon. Edge Analytics in the Internet of Things. *IEEE Pervasive Computing*, 14(2):24–31, 2015. doi:10.1109/MPRV.2015.32.

[Star01] Starner, Thad. The Challenges of Wearable Computing: Part 2. *IEEE Micro*, 21(4):54–67, 2001. doi:10.1109/40.946683.

[Star01a] Starner, Thad. The Challenges of Wearable Computing: Part 1. *IEEE Micro*, 21(4):44–52, 2001. doi:10.1109/40.946681.

[StBo02] Star, Susan Leigh and Bowker, Geoffrey C. How to Infrastructure. In Lievrouw, Leah A. and Livingstone, Sonia (Eds.), *Handbook of New Media: Social Shaping and Consequences of ICTs*, pp. 151–162. SAGE Publications, Ltd, 1 Oliver's Yard, 55 City Road London EC1Y 1SP, 2002. doi:10.4135/9781848608245.n12.

[Stoc11] Stockhammer, Thomas. Dynamic adaptive streaming over HTTP – Standards and Design Principles. In Begen, Ali C. and Mayer-Patel, Ketan (Eds.), *Proceedings of the second annual ACM conference on Multimedia systems - MMSys '11*, p. 133. ACM Press, New York, New York, USA, 2011. ISBN 9781450305181. doi:10.1145/1943552.1943572.

[StRu96] Star, Susan Leigh and Ruhleder, Karen. Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces. *Information Systems Research*, 7(1):111–134, 1996. doi:10.1287/isre.7.1.111.

[SuWi15] Suter, Philippe and Wittern, Erik. Inferring Web API Descriptions from Usage Data. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pp. 7–12. IEEE, 12112015 - 13112015. ISBN 978-1-4673-9688-2. doi:10.1109/HotWeb.2015.19.

[ThCo06] gThomas, James J. and Cook, Kristin A. A Visual Analytics Agenda. *Computer Graphics and Applications, IEEE*, 26(1):10–13, 2006.

[TMDi17] Turchi, Tommaso, Malizia, Alessio, and Dix, Alan. TAPAS: A Tangible End-User Development tool supporting the repurposing of Pervasive Displays. *Journal of Visual Languages & Computing*, 39:66–77, 2017. doi:10.1016/j.jvlc.2016.11.002.

[Tuom01] Tuomi, Ilkka. Internet, Innovation, and Open Source: Actors in the Network. *First Monday*, 6(1), 2001. doi:10.5210/fm.v6i1.824.

[Ullm04] Ullman, Michael T. Contributions of memory circuits to language: the declarative/procedural model. *Cognition*, 92:231–270, 2004.

[VMS*17] Vaziri, Mandana, Mandel, Louis, Shinnar, Avraham, Siméon, Jérôme, and Hirzel, Martin. Generating Chat Bots from Web API Specifications. In Torlak, Emina, van der Storm, Tijs, and Biddle, Robert (Eds.), *The 2017 ACM SIGPLAN International Symposium*, pp. 44–57. 2017. doi:10.1145/3133850.3133864.

[VPBL15] Virnes, Marjo, Purma, Jukka, Bauters, Merja, and Leinonen, Teemu. Creating and Sharing Knowledge Through Experiences: A Case Study with Ach so! in Healthcare Education. In Conole, Gráinne, Klobučar, Tomaž, Rensing, Christoph, Konert, Johannes, and Lavoué, Élise (Eds.), *Design for Teaching and Learning in a Networked World*, *Lecture Notes in Computer Science*, vol. 9307, pp. 642–645. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24257-6. doi:10.1007/978-3-319-24258-3_78.

[WBJ*12] Walsh, Greg, Brewer, Robin, Joshi, Asmi, Brown, Richelle, Druin, Allison, Guha, Mona Leigh, Bonsignore, Elizabeth, Foss, Elizabeth, Yip, Jason C., Golub, Evan, Clegg, Tamara, and Brown, Quincy. DisCo. In Schelhowe, Heidi (Ed.), *Proceedings of the 11th International Conference on Interaction Design and Children - IDC '12*, p. 11. ACM Press, New York, New York, USA, 2012. ISBN 9781450310079. doi:10.1145/2307096.2307099.

[WCLa18] Wittern, Erik, Cha, Alan, and Laredo, Jim A. Generating GraphQL-Wrappers for REST(-like) APIs. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 65–83. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi:10.1007/978-3-319-91662-0_5.

[Weng98] Wenger, Etienne. *Communities of Practice: Learning, Meaning, and Identity*. Learning in doing. Cambridge University Press, Cambridge, UK, 1998. ISBN 9780521663632.

[Werq10] Werquin, Patrick. *Recognising Non-Formal and Informal Learning: Outcomes, Policies and Practices*. Organization for Economic Cooperation & Development, Washington, 2010. ISBN 9789264063846.

[West02] Westland, J.Christopher. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1):1–9, 2002. doi:10.1016/S0164-1212(01)00130-3.

[WoBl11] Worsley, Marcelo and Blikstein, Paulo. What's an Expert? Using learning analytics to identify emergent markers of expertise through automated speech, sentiment and sketch analysis. In Pechenizkiy, Mykola, Calders, Toon, Conati, Cristina, Ventura, Sebastián, Romero, Cristobal, and Stamper, John (Eds.), *4th International Conference on Educational Data Mining*, pp. 235–240. 2011. ISBN 978-90-386-2537-9.

[WrLu16] Wright, Austin and Luff, Geraint. JSON Schema: A Media Type for Describing JSON Documents: draft-wright-json-schema-00, 2016. https://tools.ietf.org/html/draft-wright-json-schema-00.

[WSW*15] Wolenetz, Matthew, Smith, Jerry, Watson, Mark, Colwell, Aaron, and Bateman, Adrian. Media Source Extensions: W3C Candidate Recommendation 12 November 2015, 2015. https://www.w3.org/TR/2015/CR-media-source-20151112/.

[WuJa04] Wulf, Volker and Jarke, Matthias. The Economics of End-User Development. *Communications of the ACM*, 47(9):41–42, 2004.

[XHBa14] Xu, Anbang, Huang, Shih-Wen, and Bailey, Brian. Voyant: Generating Structured Feedback on Visual Designs Using a Crowd of Non-Experts. In Fussell, Susan, Lutters, Wayne, Morris, Meredith Ringel, and Reddy, Madhu (Eds.), *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*, pp. 1433–1444. ACM and ACM Press, New York, NY, USA, 2014. ISBN 9781450325400. doi:10.1145/2531602.2531604.

[YoLu17] Young, Alyson L. and Lutters, Wayne G. Infrastructuring for Cross-Disciplinary Synthetic Science: Meta-Study Research in Land System Science. *Comput. Supported Coop. Work*, 26(1-2):165–203, 2017. doi:10.1007/s10606-017-9267-z.

[Yu97] Yu, Eric. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pp. 226–235. 1997. doi:10.1109/ISRE.1997.566873.

List of Figures

List of Tables

Appendices

Appendix A

OpenAPI Example

Listing 1: OpenAPI Specification of an Address Book Service

```yaml
openapi: 3.0.0
servers:
- description: Development Server
  url: http://127.0.0.1:3000
info:
  version: 1.0.0
  title: Address Book Service
  description: The API of the Address Book Service.
tags:
- name: contact
  description: Everything about contacts.
paths:
  "/contacts":
    get:
      tags:
      - contact
      description: Returns all contacts.
      operationId: getContacts
      responses:
        '200':
          description: All the contacts.
          content:
            application/json:
              schema:
                type: array
                items:
                  "$ref": "#/components/schemas/Contact"
  "/contacts/{contactId}":
    get:
```

```yaml
      tags:
      - contact
      description: Returns a particular contact.
      operationId: getContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
      responses:
        '200':
          description: A specific category.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/Contact"
    delete:
      tags:
      - contact
      description: Deletes a contact.
      operationId: deleteContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
      responses:
        '200':
          description: Contact deleted.
        '404':
          description: Contact not found.
components:
  schemas:
    Contact:
      type: object
      properties:
        id:
          type: integer
          format: int64
```

```
75          name:
76            type: string
77          lastname:
78            type: string
79          email:
80            type: string
```

Appendix B

Own Publications

Relevant Refereed Publications

[BKKl19] Bonilla Oliva, Delcy Carolina, Koren, István, and Klamma, Ralf. Infrastructuring for Crowdsourced Co-Design. *Interaction Design and Architecture (IxD&A)*, 2019.

[DRN*15] Derntl, Michael, Renzel, Dominik, Nicolaescu, Petru, Koren, István, and Klamma, Ralf. Distributed Software Engineering in Collaborative Research Projects. In *Proceedings 2015 IEEE 10th International Conference on Global Software Engineering (ICGSE 2015)*, pp. 105–109. IEEE Computer Society, Los Alamitos, CA, USA, 2015. ISBN 978-1-4799-8409-1. doi:10.1109/ICGSE.2015.12.

[HKKH18] Hensen, Benedikt, Koren, István, Klamma, Ralf, and Herrler, Andreas. An Augmented Reality Framework for Gamified Learning. In Hancke, Gerhard, Spaniol, Marc, Osathanunkul, Kitisak, Unankard, Sayan, and Klamma, Ralf (Eds.), *Advances in Web-Based Learning – ICWL 2018*, vol. 11007, pp. 67–76. Springer International Publishing, Cham, 2018. ISBN 978-3-319-96564-2. doi:10.1007/978-3-319-96565-9_7.

[KAKo19] Klamma, Ralf, Ali, Rizwan, and Koren, István. Immersive Community Analytics for Wearable Enhanced Learning. In Zaphiris, Panayiotis and Ioannou, Andri (Eds.), *Learning and Collaboration Technologies. Ubiquitous and Virtual Environments for Learning and Collaboration*, *Lecture Notes in Computer Science*, vol. 11591, pp. 162–174. Springer International Publishing, Cham, 2019. ISBN 978-3-030-21816-4. doi:10.1007/978-3-030-21817-1_13.

[KBKl14] Koren, István, Bavendiek, Jens, and Klamma, Ralf. DireWolf Goes Pack Hunting: A Peer-to-Peer Approach for Secure Low Latency Widget Distribution Using WebRTC. In Casteleyn, Sven, Rossi, Gustavo, and Winckler, Marco (Eds.), *Web Engineering*, LNCS, pp. 507–510. Springer International Publishing, Cham, Switzerland, 2014. ISBN 978-3-319-08244-8. doi:10.1007/978-3-319-08245-5_38.

[KGKl13] Koren, István, Guth, Andreas, and Klamma, Ralf. Shared Editing on the Web: A Classification of Developer Support Libraries. In *Proceedings of the 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*

(Collaboratecom 2013), pp. 468–477. 2013. doi:10.4108/icst.collaboratecom.2013.254097.

[KHKl18b] Koren, István, Hensen, Benedikt, and Klamma, Ralf. Co-Design of Gamified Mixed Reality Applications. In *Proceedings of the IEEE ISMAR 2018 Workshop on Creativity in Design with & for Mixed Reality*, pp. 315–317. 2018. doi:10.1109/ISMAR-Adjunct.2018.00094.

[KKJa20] Koren, István, Klamma, Ralf, and Jarke, Matthias. Direwolf Model Academy: An Extensible Collaborative Modeling Framework on the Web. In Michael, Judith and Bork, Dominik (Eds.), *Modellierung 2020 Short, Workshop and Tools & Demo Papers*, pp. 213–216. 2020.

[KNKl15] Koren, István, Nicolaescu, Petru, and Klamma, Ralf. Collaborative Drawing Annotations on Web Videos. In Frasincar, Flavius, Houben, Geert-Jan, and Schwabe, Daniel (Eds.), *Engineering the Web in the Big Data Era*, *Lecture Notes in Computer Science*, vol. 9114, pp. 671–674. Springer, 2015. doi:10.1007/978-3-319-19890-3_54.

[KoKl15] Koren, István and Klamma, Ralf. Smart Ambient Learning with Physical Artifacts Using Wearable Technologies. In *Workshop Proceedings of the 11th International Conference on Intelligent Environments, Prague, Czech Republic, July 15-17, 2015*, pp. 325–332. 2015. doi:10.3233/978-1-61499-530-2-325.

[KoKl16] Koren, István and Klamma, Ralf. Smart Ambient Learning with Physical Artifacts Using Wearable Technologies. *EAI Endorsed Transactions on Future Intelligent Educational Environments*, 2(6), 2016. doi:10.4108/eai.27-6-2016.151526.

[KoKl16b] Koren, István and Klamma, Ralf. The Direwolf Inside You: End User Development for Heterogeneous Web of Things Appliances. In Bozzon, Alessandro, Cudre-Maroux, Philippe, and Pautasso, Cesare (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 9671, pp. 484–491. Springer International Publishing, Cham, 2016. ISBN 978-3-319-38790-1. doi:10.1007/978-3-319-38791-8_35.

[KoKl17] Koren, István and Klamma, Ralf. Community Learning Analytics with Industry 4.0 and Wearable Sensor Data. In Beck, Dennis, Allison, Colin, Morgado, Leonel, Pirker, Johanna, Khosmood, Foaad, Richter, Jonathon, and Gütl, Christian (Eds.), *Immersive Learning Research Network*, pp. 142–151. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60632-3. doi:10.1007/978-3-319-60633-0_12.

[KoKl18] Koren, István and Klamma, Ralf. Enabling visual community learning analytics with Internet of Things devices. *Computers in Human Behavior*, 89:385–394, 2018. doi:10.1016/j.chb.2018.07.036.

[KoKl18b] Koren, István and Klamma, Ralf. Generation of Web Frontends from API Documentation with Direwolf Interaction Flow Designer. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 492–495. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi:10.1007/978-3-319-91662-0_42.

[KoKl18c] Koren, István and Klamma, Ralf. Peer-to-Peer Video Streaming in HTML5 with WebTorrent. In Mikkonen, Tommi, Klamma, Ralf, and Hernández, Juan (Eds.), *Web Engineering*, *Lecture Notes in Computer Science*, vol. 10845, pp. 404–419. Springer International Publishing, Cham, 2018. ISBN 978-3-319-91661-3. doi: 10.1007/978-3-319-91662-0_33.

[KoKl18d] Koren, István and Klamma, Ralf. The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. In Champin, Pierre-Antoine, Gandon, Fabien, Gandon, Fabien, Lalmas, Mounia, and Ipeirotis, Panagiotis G. (Eds.), *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*, pp. 781–787. ACM Press, New York, New York, USA, 2018. ISBN 9781450356404. doi: 10.1145/3184558.3188740.

[KoKl19] Koren, István and Klamma, Ralf. OakStreaming: A Peer-to-PeerVideo Streaming Library. *Journal of Web Engineering*, 17(6):527–560, 2019. doi:10.13052/jwe1540-9589.17675.

[KRN*14] Kovachev, Dejan, Renzel, Dominik, Nicolaescu, Petru, Koren, István, and Klamma, Ralf. DireWolf: A Framework for Widget-based Distributed User Interfaces. *Journal of Web Engineering*, 13(3&4):203–222, 2014.

[LNKK16] de Lange, Peter, Nicolaescu, Petru, Klamma, Ralf, and Koren, István. DevOpsUse for Rapid Training of Agile Practices Within Undergraduate and Startup Communities. In Verbert, Katrien, Sharples, Mike, and Klobučar, Tomaž (Eds.), *Adaptive and Adaptable Learning*, *Lecture Notes in Computer Science*, vol. 9891, pp. 570–574. Springer International Publishing, Cham, 2016. ISBN 978-3-319-45152-7. doi: 10.1007/978-3-319-45153-4_65.

[NRK*14] Nicolaescu, Petru, Renzel, Dominik, Koren, István, Klamma, Ralf, Purma, Jukka, and Bauters, Merja. A Community Information System for Ubiquitous Informal Learning Support. In *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*, pp. 138–140. 2014. doi:10.1109/ICALT.2014.48.

[RKKJ17] Renzel, Dominik, Koren, István, Klamma, Ralf, and Jarke, Matthias. Preparing Research Projects for Sustainable Software Engineering in Society. In *Proceedings 2017 IEEE/ACM 39th IEEE International Conference on Software Engineering (ICSE)*. 2017. ISBN 978-1-5386-1589-8. doi:10.1109/ICSE-SEIS.2017.4.

[SGR*14] Schuster, Daniel, Grubitzsch, Philipp, Renzel, Dominik, Koren, István, Klauck, Ronny, and Kirsche, Michael. Global-Scale Federated Access to Smart Objects Using XMPP. In Bilof, Randall (Ed.), *Proceedings 2014 IEEE International Conference on Internet of Things (iThings 2014), Green Computing and Communications (GreenCom 2014), and Cyber-Physical-Social Computing (CPSCom 2014)*, pp. 185–192. IEEE Computer Society, Los Alamitos, CA, USA, 2014. ISBN CFP14GCC-CDR. doi:10.1109/iThings.2014.35.

Project Deliverables

[BBB*14] Bauters, Merja, Burchert, Joanna, Burchert, Michael, Klamma, Ralf, Koren, István, Müller, Werner, Ngua, Kiarii, Nicolaescu, Petru, Nikkilä, Leo, Pejoska, Jana, and Purma, Jukka. Layers Tools for Artefact and Mobile Layer: D4.2: Learning Layers Project Deliverable, 2014.

[BBF*13] Bauters, Merja, Burchert, Joanna, Funke, Tobias, Klamma, Ralf, Koren, István, Kämäräinen, Pekka, Müller, Werner, Ngua, Kiarii, Nicolaescu, Petru, and Purma, Jukka. Concept & Prototype for Artefact and Mobile Layer: D4.1: Learning Layers Project Deliverable, 2013.

[BFK*13] Bilyatdinova, Anna, Fominykh, Mikhail, Koren, István, Jesionkowska, Joanna, Karsakov, Andrey, Khoroshavin, Aleksandr, Klamma, Ralf, Klimova, Alexandra, Molka-Danielsen, Judith, Rasool, Jazz, Smith, Carl H., and Wild, Fridolin. Existing Teaching Practices and Future Labour Market Needs in the Field of Augmented Reality: Analytical report, 2019.

[DHK*14] Derntl, Michael, Hannemann, Anna, Klamma, Ralf, Koren, István, Nicolaescu, Petru, Renzel, Dominik, Kravčík, Miloš, Shahriari, Mohsen, Purma, Jukka, Bachl, Martin, Bellamy, Edward, Elferink, Raymond, Tomberg, Vladimir, Theiler, Dieter, and Santos, Patricia. Customizable Architecture for Flexible Small-Scale Deployment: D6.2: Learning Layers Project Deliverable, 2014.

[DKK*13] Derntl, Michael, Klamma, Ralf, Koren, István, Kravčík, Miloš, Nicolaescu, Petru, Renzel, Dominik, Ngua, Kiarii, Purma, Jukka, Attwell, Graham, Gray, Owen, Ley, Tobias, Tomberg, Vladimir, Henry, Christina, Whitehead, Chris, Theiler, Dieter, Trattner, Christoph, Maier, Ronald K., Manhart, Markus, Schett, Maria, and Thalmann, Stefan. Initial Architecture for Fast Small-Scale Deployment: D6.1: Learning Layers Project Deliverable, 2013.

[KHK*17] Koren, István, Hug, Martin, Klamma, Ralf, Limbu, Bibeg, Vizzi, Carlo, Holand, Morten, and Minetti, Irene. Requirements for Scenarios and Prototypes: D1.6: WEKIT Project Deliverable, 2017.

[KKF*17] Koren, István, Klamma, Ralf, Fominykh, Mikhail, Wild, Fridolin, and Rossi, Elisa. Outreach & Dissemination Report 2: D7.3: WEKIT Project Deliverable, 2017.

[KKF*18] Koren, István, Klamma, Ralf, Fominykh, Mikhail, Wild, Fridolin, and Limbu, Bibeg. Final Outreach & Dissemination Report: D7.4: WEKIT Project Deliverable, 2018.

[KKK*16] Klamma, Ralf, Kensche, Zinayida, Kravčík, Miloš, Renzel, Dominik, Koren, István, de Lange, Peter, Neulinger, Kateryna, Nicolaescu, Petru, Shahriari, Mohsen, and Toubekis, Georgios. Advanced Community Information Systems (ACIS) - Annual Report 2016, 2016. doi:10.13140/RG.2.2.32380.95367.

[KKL*16] Kravčík, Miloš, Koren, István, de Lange, Peter, Fominykh, Mikhail, Klamma, Ralf, and Kupriyanova, Veronika. Dissemination Plan: D7.1: WEKIT Project Deliverable, 2016.

[KKN*15] Klamma, Ralf, Koren, István, Nicolaescu, Petru, Renzel, Dominik, Kravčík, Miloš, Shahriari, Mohsen, Derntl, Michael, Peffer, Gilbert, and Elferink, Raymond. DevOpsUse - Scaling Continuous Innovation: D6.3: Learning Layers Project Deliverable, 2015.

[KKR*18] Koren, István, Klamma, Ralf, Ravagnolo, Liliana, Fruscione, Marco, and Brox, Chris. Requirements for Scenarios and Prototypes: D1.8: WEKIT Project Deliverable, 2018.

[KLK*16] Kravčík, Miloš, de Lange, Peter, Koren, István, Klamma, Ralf, Helin, Kaj, Kuula, Timo, Rasool, Jazz, Smith, Carl H., Zics, Brigitta, and Klemke, Roland. Requirements for Scenarios and Prototypes: D1.4: WEKIT Project Deliverable, 2016.

[RFK*16] Renzel, Dominik, Fominykh, Mikhail, Klamma, Ralf, Koren, István, de Lange, Peter, Lefrere, Paul, Rubattino, Cinzia, and Wild, Fridolin. Outreach & Dissemination Report 1: D7.2: WEKIT Project Deliverable, 2016.

[SXW*17] Sharma, Puneet, Xue, Hui, Wild, Fridolin, Klemke, Roland, Koren, István, Guest, Will, Vovk, Alla, Limbu, Bibeg, Schneider, Jan, and Di Mitri, Daniele. Requirement analysis and sensor specifications - Final version: D3.4: WEKIT Project Deliverable, 2017.

Other Publications

[DKN*14] Derntl, Michael, Koren, István, Nicolaescu, Petru, Renzel, Dominik, and Klamma, Ralf. Blueprint for Software Engineering in Technology Enhanced Learning Projects. In Rensing, Christoph, de Freitas, Sara, Ley, Tobias, and Muñoz Merino, Pedro J. (Eds.), *Open Learning and Teaching in Educational Communities*, Lecture Notes in Computer Science, pp. 404–409. Springer Switzerland and Springer, Berlin, 2014. ISBN 978-3-319-11199-5.

[KHKl18] Koren, István, Hensen, Benedikt, and Klamma, Ralf. Augmented Reality Lernkontexte – Eine Europäische Perspektive. In Schiffner, Daniel (Ed.), *Proceedings der Pre-Conference-Workshops der 16. E-Learning Fachtagung Informatik*. CEUR-WS, 2018. ISBN 1613-0073.

[Kore14] Koren, István. Turning User Requirements Into Technical Features With the House of Quality. In Renzel, Dominik and Klamma, Ralf (Eds.), *Large-Scale Social Requirements Engineering*. IEEE Special Technical Community on Social Networking (IEEE STCSN), 2014.

[KSSp13] Koren, István, Schuster, Daniel, and Springer, Thomas. Session Mobility for Collaborative Pervasive Apps Using XMPP. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 169–174. 2013. ISBN 978-1-4673-5075-4. doi:10.1109/PerComW.2013.6529476.

[SKS*13] Schuster, Daniel, Koren, István, Springer, Thomas, Hering, Dirk, Söllner, Benjamin, Endler, Markus, and Schill, Alexander. Creating Applications for Real-Time Collaboration with XMPP and Android on Mobile Devices. In Alencar, Paulo and Cowan,

Donald (Eds.), *Handbook of Research on Mobile Software Engineering: Design; Implementation and Emergent Applications*. IGI Global, 2013.

Appendix C

Curriculum Vitae

Name:	István Koren
Birthday:	July 23, 1986 in Budapest, Hungary
Email:	koren@dbis.rwth-aachen.de
Language Skills:	Hungarian (native), German (native), English (fluent), French (advanced), Portuguese (advanced)
Academic Education:	01/2013 - present: *PhD Candidate at RWTH Aachen University* 10/2008 - 06/2012: *Computer Science and Media diploma studies at Technische Universität Dresden* (Dipl.-Medieninf., distinction) 10/2005 - 09/2008: *Computer Science and Media bachelor studies at Technische Universität Dresden* (Bakk.-Medieninf., distinction)
Research Projects:	01/2019 - present: *DFG Cluster of Excellence Internet of Production* 01/2017 - 12/2018: *EU Erasmus Plus AR-FOR-EU* 01/2016 - 12/2018: *EC H2020 WEKIT* 01/2013 - 12/2016: *EU FP7 IP Learning Layers*
Professional Experience:	01/2013 - present: *Research & Teaching Assistant at Chair of Computer Science 5 (Databases & Information Systems), RWTH Aachen University* 06/2012 - 08/2012: *Research stay at Shizuoka University, Japan (funded by DAAD & JSPS)* 01/2010 - 06/2010: *Internship at Amadeus Labs Bangalore, India (funded by DAAD)* 11/2008 - 04/2009: *Research stay at PUC Rio de Janeiro, Brazil (funded by BMBF & CNPq)* 02/2006 - 09/2008: *Working student at Qimonda (Infineon) Dresden*