ArchWiki: Using Web 2.0 for Architecture Knowledge Management

Master's Thesis

Didier Liauw

ArchWiki: Using Web 2.0 for Architecture Knowledge Management

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Didier Liauw
born in Rotterdam, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
http://eemcs.tudelft.nl

ArchWiki: Using Web 2.0 for Architecture Knowledge Management

Author: Didier Liauw
Student id: 1177834
Email: d.liauw@student.tudelft.nl

Abstract

Software architecture plays an important part in program comprehension, which is one of the most time consuming tasks in software development. If software developers don't properly share their architectural knowledge with team members, the team will act based on an incomplete or even possibly incorrect view on the code base, and this can lead to architectural degradation.

Recently there has been a surge of collaboration, communication and sharing with the advent of Web 2.0 applications. In this thesis we have investigated how Web 2.0 can be used to support software architecture management. In particular in the area of architecture documentation, architecture retrieval, and collaboration.

We created an approach which applies Web 2.0 concepts such as traceability, integration, usability, navigability, and user experience, to software architecture management. This approach is supported by a prototype tool called ArchWiki, which has features such as traceability between different artifacts (e.g. source code, architectural diagrams, architectural documentation), context-sensitive views, hyperlinks, notifications, tags, and bookmarks. We performed an initial evaluation study to assess ArchWiki. In this study we found that Web 2.0 has the potential to support software architecture knowledge management.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. Ir. M. Pinzger, Faculty EEMCS, TU Delft
Committee Member:	Dr. Ir. J. Hidders, Faculty EEMCS, TU Delft

Preface

Before you lies the result of my master's thesis project. It represents the knowledge and skills I have acquired during many years of study at the TU Delft. For my thesis project I had the opportunity to explore my own ideas in a scientific way. This exploration took a long time to finish and exposed me to the world of scientific research, most of it unknown to me at the time. In the end I learned many things during my journey and I am proud of the result.

My guide during this scientific journey was Martin Pinzger. I would like to thank him for his supervision. During the many meetings we discussed ideas, he provided feedback and guidance, and helped me stay within scope and focused on finishing the project.

I would like to thank Andy Zaidman for his help with the evaluation study of ArchWiki and the volunteers that participated in this study.

Finally I would like to thank my fellow master's students in room 8.240 that kept me company and were always ready to help me with problems I had.

Didier Liauw
Delft, the Netherlands
October 4, 2012

Contents

List of Figures

Chapter 1

Introduction

The emergence and popularity of Web 2.0 applications such as Wikipedia, Facebook and Twitter has sparked a surge of collaboration. Because of these applications, people are sharing information they would otherwise not have shared. At the same time the information they share is consumed by a whole range of people, from close friends with, for example, Facebook, to total strangers with, for example, Wikipedia. People are enticed to use the applications by offering superior usability which often enables them to use the web application without the guidance of a user manual. The utility of the application also increases by enabling users to share more than one type of artifact. Facebook, for example, allows users to share not only some personal contact information, but also other artifacts, such as photos, videos and small blog-like messages. These different kinds of artifacts are then interconnected. Photos can, for example, be connected to a message posted on a wall. The same photo can contain links to the persons shown on the photo. This is one of the great strengths of Web 2.0: everything is connected. These interconnections allow users of Web 2.0 applications to do things that previously would not have been possible. A small effort by the contributor will result in benefit for multiple consumers.

Software development is an area in which collaboration and communication are very important. A software system is usually build by multiple developers. The system is created with underlying design decisions in mind. The principal design decisions constitute the software architecture of that system. While working with the system it is important to have knowledge of the system's software architecture.

Software developers sometimes look like detectives trying to find information. They chase leads and follow traces to get a good picture of the situation and reconstruct what happened. But unlike the detective's case, in which a criminal would want to cover up their traces, the developer follows traces left by his fellow developers. It would be in the best interest of everyone if software developers would be able to get a clear picture of the situation, which is in this case understanding the code base. Reading software architecture documentation is an important part of comprehending the code base [21]. This is only possible if the software developers have documented the software architecture in the past.

In practice, most of the software architecture is still only present in the minds of the developers. To acquire this knowledge developers spend a large amount of time on unplanned communication with other coworkers [56].

Both documentation and informal communication are important forms of architecture knowledge management. For example, in globally distributed software development, where informal communication is harder to do, work items take 2.5 times longer to complete than in the case of co-located development [34]. Unfortunately, both forms have their limitations. If a developer wants to find the documentation for a piece of source code, he first has to be able to find the documentation and in the documentation he has to find the relevant information for him. If the relevant information has been found it is often not clear if the information is still up to date. Also if the documentation is not complete it is often not clear who the developer should contact to get further information.

We believe that using the Web 2.0 way of thinking and its resulting features, such as artifact traceability and high usability, have the potential to support software architecture knowledge management and that idea served as the starting point for our master's thesis.

1.1 Research Motivations

In this Web 2.0 era the areas of collaboration, participation, contribution and usability are at an all time high. Software architecture has received increasing attention as an important subfield of software engineering [28]. It plays an important role in program comprehension, which is one of the most time consuming tasks in software development [46]. It is also seen as an area that involves more stakeholders than just the software architect and is involved in more software development phases than just the early one-time design phase, as described in the waterfall model [61].

However, the use of approaches and tools in the area of software architecture knowledge management is lagging behind these developments. Although technology plays a key role in knowledge management in organizations [63], many of the current approaches and tools are difficult to use, require training and a manual to be used, are hard to maintain, they do not match with the current practice of software architecture knowledge management, which currently is mostly based on informal discussions and drawings on whiteboards, and do not provide support for collaboration.

This is especially the case with the advent of agile development methods, which focus less on comprehensive big up front documentation and more on face-to-face communication to convey information.[1] However, not documenting the architecture and solely relying on face-to-face communication is not ideal either [18]. For example, the architectural knowledge disappears if the only person that has the knowledge leaves the company. These agile methodologies, may benefit more from collaborative, lightweight, incremental, and highly usable methods and tools. These aspects are all important in the Web 2.0 way of thinking. Therefore, in this research we explore the possibilities to apply Web 2.0 to support the area of software architecture knowledge management.

1.2 Research Question

The central research question that guides this thesis is:

[1] http://agilemanifesto.org/principles.html

> Is it possible to support architecture knowledge management by using Web 2.0?

We focus our efforts on the following areas of architecture knowledge management:

- Architectural knowledge retrieval
- Architectural knowledge documentation
- Collaboration

The first two items are straightforward. The third item is chosen, because the current practice of software architecture knowledge management involves a lot of collaboration, which is an important aspect of Web 2.0.

1.3 Research Objective and Research Methodology

The objective of this research is to answer the research question of the previous section, in particular in the areas of documentation, retrieval and collaboration. The research methodology is threefold:

1. Envision an approach for software architecture knowledge management inspired by Web 2.0.
2. Create a tool to support this approach.
3. Evaluate the approach and tool in how it supports software architecture knowledge management.

1.4 Contributions

The contributions we make in this master's thesis are as follows:

- The ArchWiki approach, which uses Web 2.0 concepts to support software architecture knowledge management.
- The ArchWiki tool, which is developed to support the ArchWiki approach.
- An initial evaluation of the ArchWiki approach and tool.

1.5 Organization of this Thesis Report

This thesis report is organized as follows. Chapter 2 gives background information and provides the context of this thesis. The Background chapter is divided into two parts: Section 2.1 provides information about the field of software architecture and Section 2.2 explains the concepts of Web 2.0. Chapter 3 describes our approach of using Web 2.0 for architecture knowledge management: ArchWiki. The following chapter (Chapter 4) describes to implementation of the ArchWiki tool used to support our approach. This tool is evaluated in Chapter 5 using a small user study. Related work is discussed in Chapter 6. Chapter 7 concludes this thesis report and gives a summary

of the work, answers to the research questions, the contributions, discussions, future work, and suggestions for future evaluation studies.

Chapter 2

Background

This chapter presents background information related to this master's thesis. The first section starts with a description of software architecture, and its current role and status during software development. The next section will describe the rise of Web 2.0 and its essence.

2.1 Software Architecture

2.1.1 Definition

Software architecture is an immature but developing aspect of software development. The level of maturity is signified by its current inconsistent use in the software development process and the level of tool support that supports this process. But it is most characterized by the very large collection of different definitions of software architecture [65]. In this thesis the definition of Taylor et al. [71] of software architecture is used:

> A software system's architecture is the set of principal design decisions made about the system.

Design decisions encompass every aspect of the system under development and might be related to the system structure, functional behavior, interaction, the system's non-functional properties and the system's implementation. It is up to the stakeholders of the architecture to decide which of the system's design decisions are considered principal. Every system has underlying design decisions, so every system has an architecture. However, this architecture is mostly present in the stakeholders' minds, until they document the architecture. Documenting the architecture prevents it from fading away and allows the architecture to be communicated to other stakeholders.

We use another term that has a number of different definitions [20]: Architectural Knowledge (AK). In this report we use the definition from Kruchten et al. [41]:

> Architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is.

2.1.2 Importance of Documenting the Software Architecture

Ideally the software architecture is documented in the source code. You would want someone to look at a piece of unfamiliar source code and think: "I know this piece of source code as if I had just written it myself". Unfortunately, reading source code is much harder than to writing source code.[1] If you write source code, you understand, or at least are aware of the problem you try to address, the architecture of the context, and also the design decisions that are made when writing the source code. These factors are all forms of architectural knowledge. The combination of these factors and other factors, like the personal style of the developer, will lead to a particular piece of source code. Because there are so many factors that influence the resulting source code, it is unlikely that two developers will create the same source code.

"What I cannot create, I do not understand."

Richard Feynman

This quote by Richard Feynman is applicable to software development. It means that to understand something, you should be able to create it yourself. In software development this is often not the case and you would have probably created a different piece of source code instead, because factors such as architectural knowledge, are most likely not the same. Because of this, it is hard to understand source code made by others and documenting architectural knowledge can help to increase the understanding.

Reenskaug and Coplien [70] advocate the use of software architectures that are specifically designed to match the mental model of the end user with the Data, Context and Interaction Architecture, so the source code becomes easier to understand and to maintain. However, they also argue that code alone is not enough to understand the software architecture, because source code lacks context and the rationale behind the design decisions. Along a similar line, Parnas and Clements [55] argue that it is impossible to create software through a rational design process, though it is possible to produce documentation that makes it appear that the software was designed by such a process.

Program comprehension is one of the most important and time consuming processes during software development [46], which maintainers spend 40% to 60% of their time on [57]. This is especially valid for new developers who are totally unfamiliar with a code base. This is one of the main causes of Brooks's law:

"Adding manpower to a late software project makes it later" [8]

Therefore it is important to document the software architecture of a software application.

2.1.3 Software Architecture Documentation

Every system has underlying design decisions, so every system has an architecture. However, this architecture is mostly present in the minds of the stakeholders that made

[1] http://www.joelonsoftware.com/articles/fog0000000069.html

the decisions. Documenting the architecture prevents it from fading away and allows the architecture to be communicated to other stakeholders. To document the architecture it is captured in an architectural model. Taylor et al. [71] define this as:

> An architectural model is an artifact that captures some or all of the design decisions that comprise a system's architecture. Architectural modeling is the reification and documentation of those design decisions.

Architectural modeling notations are defined as:

> An architectural modeling notation is a language or means of capturing design decisions [71].

Architectural modeling notations are also known as architectural description languages (ADLs). There is a wide range of different ADLs in use today. They range from informal whiteboard drawings, box and line drawings on a piece of paper, text documents, PowerPoint diagrams to ADLs specifically designed to capture software architecture such as UML diagrams. Often a combination of ADLs are used, with each documenting a different part of the architecture. For example, UML may be used to document to the structure of a system, a text document may be used to document the requirements and design decisions including the rationale behind these design decisions, and a whiteboard drawing may be created to explain the behavior aspect of a particular part of the system.

The purpose of architecture documentation is to serve as a means of communication and to increase the program comprehension. The end result is that a well documented architecture facilitates a shorter development time [43]. A pleasant side-effect of documentation efforts is often, the improvement of the software [54]. Having a proper understanding of the software architecture is important to many types of stakeholders during all phases of development [71]:

- Requirements engineers can properly define which requirements are important or even feasible by looking at existing designs. New software applications are rarely created in isolation. Often the architecture an already existing system forms the basis of reasoning. For example, it would almost be impossible to create a new social networking website without (unconsciously) looking at Facebook[2], which is the current market leader.

- Software design is an important task during the early phases of development. During these phases many of the design decisions are taken and the software architecture is created. These phases are very important, because the software architecture will form the basis on top of which future features will be created. Having a good understanding of domain specific or general software architecture, such as architectural styles or design patterns, is important here.

- When implementing new features, the software developer should have a good understanding of the functional requirements, but also of the non-functional requirements of the feature he is implementing. Understanding the underlying

[2]http://www.facebook.com

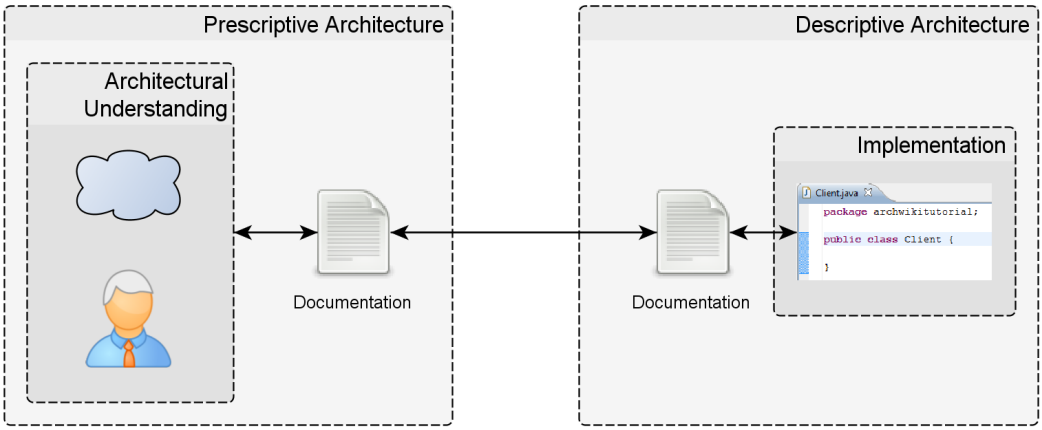

Figure 2.1: The prescriptive architecture and the descriptive architecture

architecture of the overall system is also important to avoid violating architectural constraints.

- During software maintenance tasks, such as fixing errors, it is important to have a proper understanding the software architecture. Not only to isolate the location of the error, but also to correct the error without violating architectural constraints and thus introducing new errors.
- Software testers who work in quality assurance can coordinate their tests around the established design decisions, and prioritize that the more important design decisions are tested. Understanding the software architecture can also help testers identify potential errors early and create tests for them.
- Project managers can more easily determine what the most desirable features are, how they are to be implemented in the current architecture and estimate how long the implementation will take.

2.1.4 Consequences of Not Documenting the Architecture

Software architects will make design decisions that reflect their intent. This set of design decisions is called the prescriptive architecture. When they document these design decisions these documents act as a blueprint for the implementation. The descriptive architecture is the set of the design decisions which are actually realized in the implementation. These design decisions can also be documented.

Ideally the prescriptive architecture and the descriptive architecture are the same, and their respective documentation are the same documents. However, this is rarely the case and over the time the prescriptive architecture, containing the designer's intent, and the descriptive architecture, containing the realized design decisions, will diverge, which is called architectural degradation. Architectural degradation consists of two related phenomena:

Architectural drift is introduction of principal design decisions into a system's descriptive architecture that (a) are not included in, encompassed

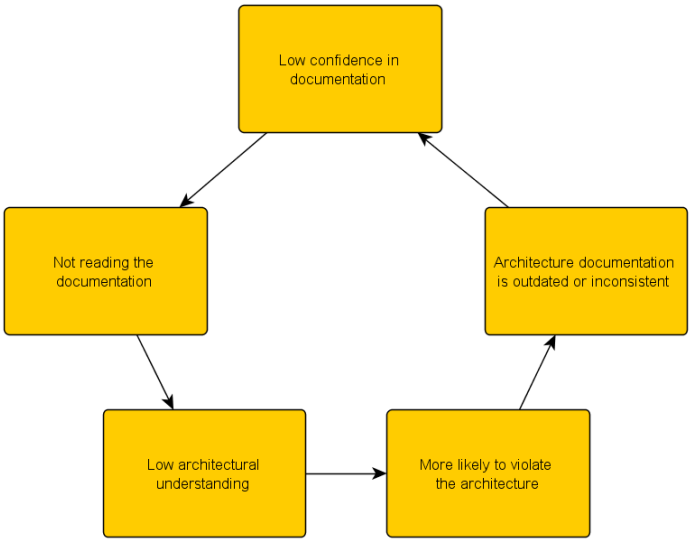

Figure 2.2: The vicious circle of low confidence in documentation, low architectural understanding, and architectural degradation.

> by, or implied by the prescriptive architecture, but which (b) do not violate any of the prescriptive architecture's design decisions [71].
>
> Architectural erosion is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture [71].

Architectural degradation happens for several reasons: developer sloppiness, perception of short deadlines that prevent thinking through and documenting the impact on the prescriptive architecture, lack of a documented prescriptive architecture, need or desire to optimize the system "which can be done only in the code," inadequate techniques and tool support, and so forth [71, 33].

Architectural drift is easier to rectify, because it only requires the architectural design decisions to be documented and incorporated into the prescriptive architecture. On the other hand, architectural erosion also requires changes to the prescriptive and/or descriptive architecture. If not properly addressed, architectural drift will eventually result in architectural erosion [71].

When the architecture degrades, and the inconsistencies between the prescriptive and descriptive architecture documentation become larger, the confidence in the correctness of the documentation decreases. If this confidence drops below a certain point the documentation will not be used anymore and the benefits of using the documentation will be gone. The developers are less likely to be aware of the design decisions, and in turn are more likely to violate these design decisions.

Figure 2.2 shows the vicious circle in which the low confidence in the documentation, will lead to low understanding of the architecture, which will lead to more violations of the software architecture, which will lead to more inconsistencies in the

documentation, if documented at all, which will lead to even lower confidence in the documentation.

2.1.5 Current Practices

Unfortunately, in practice, the documentation is often outdated and inconsistent. Most programmers regard documentation as a necessary evil, writing as an afterthought only because some bureaucrat requires it [55]. In many organizations, this documentation ends up on the shelves, unused and collecting dust [7].

On one end of the spectrum, design documents are created with a Big Design Up Front (BDUF). The problem with this approach is that not all requirements are known up front and the requirements themselves can change during development. Software developers are all in (at least) some small part an architect themselves and make design decisions while writing code. Design decisions do not only include new design, but also evaluations of older design. These design decisions are often not recorded and integrated in the already existing software architecture documentation [54]. Even if it is documented the documentation is not sufficiently shared within the organization [42].

On the other end of the spectrum is the agile software development method. This method has become very popular recently. Agile[3] software development is about creating value early and incrementally add more value. It recognizes the emerging and changing requirements and therefore tries to avoid BDUF and comprehensive documentation. However, the code as the primary source of the software architecture is not ideal. Without documentation the developer has to use architecture recovery to recover the architecture from the source code, which can take a long time because design decisions are difficult to track and reconstruct from the system [76]. In most cases the industrial practice is not what is recommended by applicable text books. They do not document their architecture in a formal manner, but instead rely on a practice, which they have coined a 'walking architecture', which is defined by Unphon and Dittrich as:

> A key person, or a number of key persons, who maintain and update the structure of the software, and are involved in discussions of changes motivated in the development, or by new requirements, and who introduce newcomers to the structure of the software [75].

The problem with the 'walking architecture' is that the architectural knowledge is lost when the 'walking architecture' leaves the company before documenting the architecture.

2.1.6 Architecture Knowledge Management

The way architectural knowledge is acquired and shared is the area of architecture knowledge management (AKM) [5]. The current state-of-the-practice of architecture knowledge management is face-to-face communication [75, 64]. This is very understandable as the main source of architectural knowledge is in the heads of the 'walking architecture' and face-to-face communication is one of the most effective means of communication as can be seen in Figure 2.3.

[3] http://www.agilemanifesto.org/

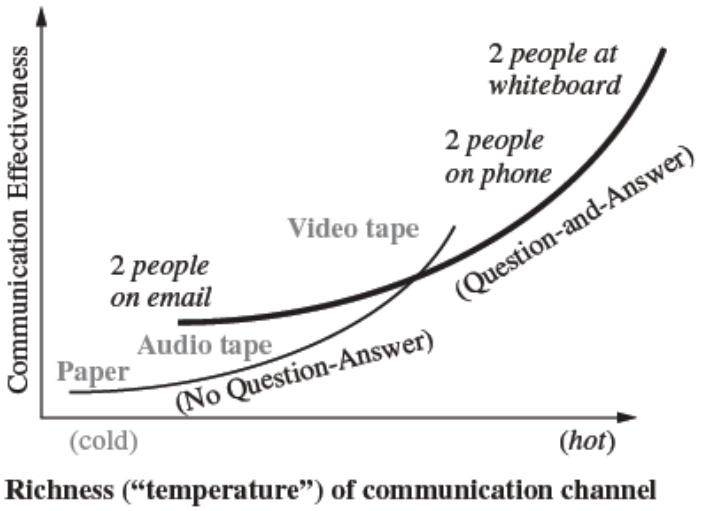

Figure 2.3: Forms of communication [16]

New developers gain architectural knowledge by having conversations with (senior) developers, which can, for example, start spontaneously around the water cooler. Another way of gaining architectural knowledge is doing pair programming with another developer and ask them questions about the structure, design decisions and rationale behind the currently viewed piece of source code. Regular meetings provide awareness and are used to synchronize everybody's understanding of the current architecture. [75]

This current practice has some issues:

- The architectural knowledge in face-to-face communication rarely is documented [64].
- Many of the conversations and meetings will result in architectural knowledge that has been established in previous encounters, and thus redo previous efforts, which may not be the most efficient way of sharing knowledge.
- A piece of architecture information can only be shared if someone who has the architecture information participates in the communication.
- If the person that 'owns' a particular piece of knowledge leaves the organization, the knowledge may be lost forever [75].
- Face-to-face communication can be hard for distributed teams, that may not even be active simultaneously due to different time zones [34, 33].
- If developer is not present during a conversation or meeting he will not receive the architectural knowledge.
- You can't really be sure that people have the same understanding of a particular piece of architectural knowledge.

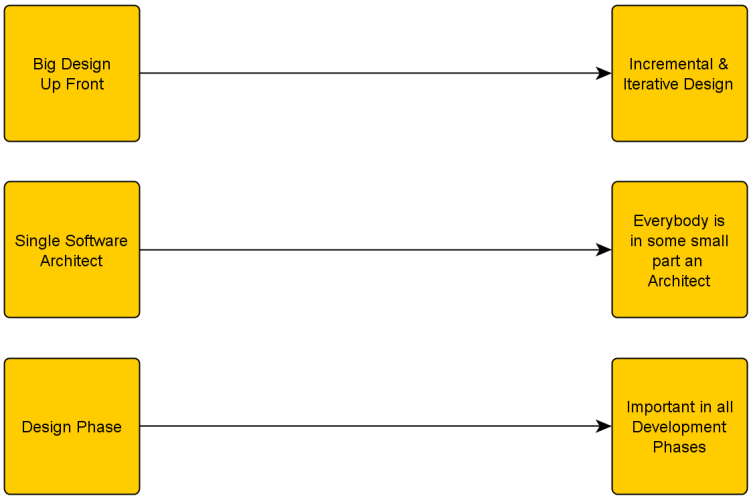

Figure 2.4: Shift in software development regarding software architecture

- Because the architectural knowledge is distributed across the minds of the individual developers, it can be hard to find a particular piece of architectural knowledge [33].

2.1.7 The Future of Software Architecture

Water cooler conversations are the process of choice in current architecture knowledge management [33]. Whiteboards and pen-and-papers are the tools of choice [75]. The most common solution for architecture documentation is the pairing of an editing tool (e.g. Microsoft Word) with a configuration management tool (e.g., Concurrent Versions System) [7]. These practices indicate the level of maturity in the area of software architecture knowledge management and the place of software architecture in the whole software development process. However, in the recent years the field of software architecture and AKM has been given more attention not only in the research community [39], but also in the industry. Taylor et al. [71] even argue in their Software Architecture Centric View that the software architecture is the most important artifact and should be used to guide the whole development process.

Figure 2.4 shows the shift from the idea that software architecture is only done by a single person, the software architect, and only during one phase, the design phase, to create one BDUF, which serves as a blueprint, to the idea that everyone is an architect [18] and that software architecture is used throughout all phases of software development, and that the software architecture is created iteratively and incrementally. These new practices involve more stakeholders and as such require better communication and collaboration between them. Because the software architecture is not set in stone, but a constantly evolving artifact it is also important to be aware of the changes made to the software architecture.

2.2 Web 2.0

Not only in software development there has been a surge of communication, collaboration and awareness. Recently a great number of so called Web 2.0 websites have emerged that change the way people look at these areas. In this section we will introduce Web 2.0, explain what it is, and illustrate with examples how Web 2.0 changed the way things previously had been done. The purpose of this section is to provide an understanding of our view of Web 2.0. This understanding provides the basis for our application of Web 2.0 concepts to architecture knowledge management, which will be explained in the next chapter.

Web 2.0 websites differ from the traditional Web 1.0 websites in that they are not static, but dynamic. Web 2.0 websites can provide a different experience from one access to another. The sources of this dynamic nature are various:

- The contents of a Web 2.0 website is often determined by dynamic data, often in the form of a network, and often from multiple sources.
- Web 2.0 websites use the connections between the data elements and the usage of data elements to generate meta data, and create personalized content.
- The user can interact with the website to control the visualization of the website content.
- The users can contribute themselves new content or edit existing contents of the website
- Users are themselves apart of the contents of the website.

Web 2.0 can perhaps best be explained by looking at a company that everybody knows: Google.[4] Google started with a search engine that used a different method to compute the search results. Search engines at the time focused on the content of a web page, Google, instead, used the link structure of the web. They assumed that if more people linked to a web page, then the web page would be more important. What they did was harnessing the collective intelligence available on the web to improve search result quality.

Some other Web 2.0 features are made clear by Google's advertising service AdSense.[5] Instead of providing the placement of ads to only a small amount of large websites, AdSense success came by providing ad placement to virtually any web page. In particular the small web sites, which make up the bulk of the web.

Another Web 2.0 application can be found in Google Maps.[6] It is a good example of combining and presenting different sources of data. Google Maps seamlessly integrates different streams of data such as: map views, satellite views overlaid with labels, terrain views, several overlays such as real time traffic, Wikipedia articles, photo's from different sources, video's, and real-estate.

[4] https://www.google.com
[5] https://www.google.com/adsense
[6] https://maps.google.com

One of the best known and used Web 2.0 application is Wikipedia.[7] It is a web-based collaborative encyclopedia that, thanks to many volunteers, is currently the largest and most popular reference work on the Internet. It success comes from the ability to easily edit almost all of its articles by anyone with access to the site.

The previous examples give an idea what Web 2.0 is all about. It's about people and data, and bringing those together for sharing and collaboration. It's about bringing power to the masses and harnessing their collective intelligence. This requires that a broad audience is reached. This is achieved by making Web 2.0 websites easy to use and to contribute. Web 2.0 websites are user-centric and often allow the user to customize the Web 2.0 to their needs.

[7] http://www.wikipedia.org

Chapter 3

ArchWiki Approach

3.1 ArchWiki Approach

Program comprehension is an important part of software engineering and that properly understanding the architecture of a software system helps greatly with this comprehension [40]. Unfortunately, current practices often do not explicitly include software architecture management in the work flow and as a result of that software architecture documentation is incomplete, often to a degree that it does not help with program comprehension or even counteracts it, because the information is incorrect or outdated. The software architecture centric view supports program comprehension by bringing software architecture into more prominence.

We created an approach, which is in line with the architecture centric view. In this approach the architecture is a central component during the whole development process for multiple stakeholders. Current software architecture tools do not support the architecture centric approach very well, mostly because they have not been developed with collaboration in mind. To support our approach we created a tool called ArchWiki. This tool puts emphasis on collaboration and is developed with the Web 2.0 mind set. We identify three major aspects important to our approach and should be supported by the ArchWiki tool.

Viewing Software architecture tasks are currently not done very frequently even by the software architects themselves, let alone other stakeholders such as developers and testers. An approach in which multiple users are frequently using the architecture would require that the users are able to easily view software architecture documentation.

Documenting Using the architecture consists not only of viewing the architecture, but also documenting the architecture. Currently this is a task mainly done by the software architect, but the software architecture centric view involves multiple stakeholders and requires that the software architecture is easily documented by several stakeholders.

Collaboration Making the software architecture documentation a central artifact for multiple stakeholders implies more collaboration on software architecture. Current

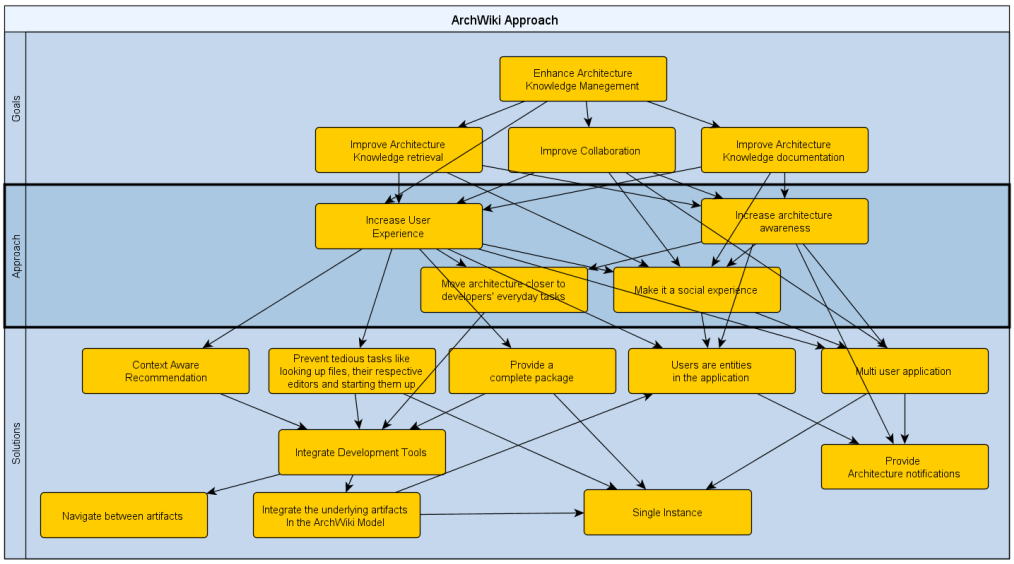

Figure 3.1: ArchWiki Approach, goals and solutions

software architecture tools are not focused on collaboration and a tool that supports the architecture centric view should make it easy for stakeholders to collaborate on software architecture [12]. This means having central artifacts to work on, easier sharing of changes to the data, and staying aware of the other software architecture users.

Figure 3.1 visualizes the ArchWiki goals, the approaches taken to achieve these, and the concrete solutions. The ArchWiki approach tries to support architecture knowledge management by bringing the architecture closer to the developer´s everyday tasks and making it a social experience. These characteristics can be found in many of today's Web 2.0 applications and our approach is based on ways to incorporate Web 2.0 to achieve our goals.

For the master's thesis we created a tool to support architecture knowledge management using Web 2.0. In this chapter we will describe the ArchWiki approach and the tool that supports it. In Section 3.2 we will describe several key Web 2.0 ideas and how we think they could be applied to architecture knowledge management. In Section 3.3 we will give a general description of the ArchWiki tool, while in Section 3.4 we go in more detail of ArchWiki's features by presenting a scenario of a current work flow of software architecture management and linking the scenario to detailed subsections of each of ArchWiki's features. In those subsections we will further elaborate the ArchWiki features by explaining how they can support software architecture management using the Web 2.0 ideas mentioned in Section 3.2.

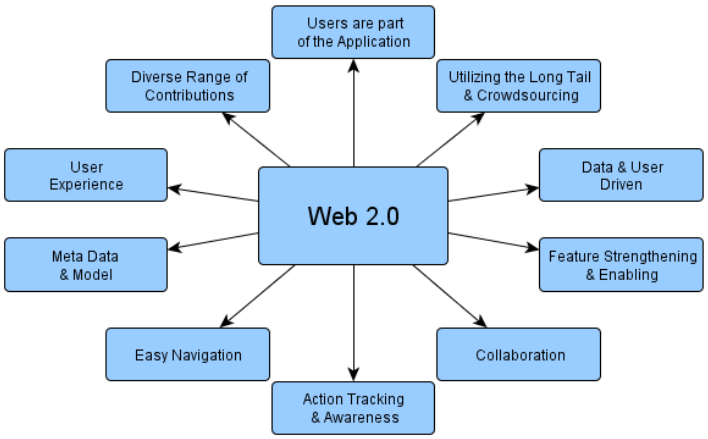

Figure 3.2: Web 2.0 Concepts

3.2 Applying Web 2.0 to Software Architecture Knowledge Management

There are a number of ways we apply Web 2.0 to architecture knowledge management. The first thing to note is that we do not use Web 2.0 by just using a set of Web 2.0 technologies. ArchWiki tries to use the Web 2.0 attitude and way of thinking. One main idea of Web 2.0 is: ´´Get more involved´´.

You can read the word 'more' in two ways: If you read it as a noun then it means getting things such as more people, businesses and data involved. You can think of Wikipedia, which involves more contributors than traditional encyclopedias, Google AdWords, which involves more advertisers, or Google Maps which involves more streams of different kinds of data. You can also think of the word 'more' as an adverb, which means using a Web 2.0 application more intensively than a traditional application, even to the point that it has become a part of daily life. Connecting the users and artifacts into a network is also a important feature of many Web 2.0 applications. Good examples of Web 2.0 applications that connect users and artifacts are social applications such as Facebook and Twitter. All this creates a network effect: The value of a product or service is dependent of the number of others using it.

For architecture knowledge management and the software architecture centric view we also want to get more involved: We want more stakeholders getting involved in software architecture and we want them to get more involved by making software architecture part of the work flow.

In Figure 3.2 we show some of the main concepts and ideas of Web 2.0 and how we want to incorporate them in ArchWiki. They are elaborated further in the following sections.

3.2.1 Collaboration

Collaboration is an important aspect of many Web 2.0 applications. Some Web 2.0 applications emphasize on the collaborative aspect. An example of this is the col-

laborative encyclopedia Wikipedia. This encyclopedia has surpassed all other (non-collaborative) encyclopedias in a short time and is one of the great success stories of collaboration and Web 2.0. The results from an investigation by Nature [49] suggest that quality of a Wikipedia is comparable to that of Encyclopædia Britannica[1].

In the past software architecture has not focused on the collaboration aspects and possibly as a result of that the software architecture tools we have come in contact with usually have not been developed with collaboration in mind. ArchWiki, on the other hand, is envisioned to be a collaboration tool, so all the features are implemented from a collaborative perspective. Collaborative features range from the basic sharing of the software architecture to specific Web 2.0 features such as tags and wiki´s.

3.2.2 Data and User Driven Functionality and the Application Serves As a Framework

In many Web 2.0 applications the functionality is provided by the data, which is often provided by the users themselves. The success of a Web 2.0 application is therefore often determined by the users and the data they provide.

ArchWiki is also a data- and user driven application and many features focus on making it easier for users to add new data. A lot of these features are based on interconnected artifacts in the ArchWiki model. This allows users to easily navigate within the architecture and add own data.

3.2.3 Users Are Part of the Application

Users are in many Web 2.0 applications an important aspect. Usually so important that it becomes an artifact in the applications model. Some Web 2.0 applications such as Facebook even make the users the most important and central artifact.

In ArchWiki the users are also explicitly modeled and included in the model. This enables users to be connected to actions that change the architecture, or even connect them to non-change actions such as views. It can be seen as a form of human computation [59] or social computing [53].

3.2.4 User Experience

Web 2.0 applications are often lightweight and usable without requiring a manual. A lot of tasks can be performed with a low number of operations. There are even whole types of applications which focus on the lightweightness. You can think of microblogging applications, which are lightweight versions of regular blogs, of which Twitter[2] is the most well known. User experience is also important for a AKM tool such as ArchWiki, because if the tool is not adapted to the users than they will be driven away [1].

Shipp and Johnson [64] found that frequent informal design discussions are occurring, while inadequate support is provided by existing formal documentation for recording this. They propose that a solution should be lightweight, which is supported

[1] http://www.britannica.com
[2] https://twitter.com/

by the way ArchWiki adopts the Web 2.0 approach. An example of a lightweight feature of ArchWiki is the use of tags to document the architecture.

3.2.5 Interconnected Artifacts and Traceability

Web 2.0 often interconnects different types of artifacts. Think about all the different views of Google Maps, in which satellite data, mapping data, Wikipedia data, and panoramic photographs are integrated into a single application.

The ArchWiki model integrates all different types of data such as source code, architectural diagrams, and textual documentation to provide traceability. By connecting artifacts, users are able to utilize the full expressiveness of that particular artifact, instead of, for example, create diagrams with text characters [33]. Traceability between the artifacts will open up new ways to obtain a piece of architectural knowledge and enables navigation as described in Section 3.2.6. This model is used to support the data driven aspects of the system. Integration is one of the issues for design rationale systems identified by Lee [44].

3.2.6 Highly Visual and Easy Navigation Between Homogeneous and Heterogeneous Artifacts

Because Web 2.0 applications are often data based applications they usually are highly visual and allow easy navigation between the different artifacts, whether they are homogeneous artifacts such as navigating from a page to another page in Wikipedia, or heterogeneous artifacts such as navigating in Google Maps from a panoramic street view image to the same location on the map or navigating from a photograph to the profile of a friend in that photograph on Facebook.

ArchWiki tries to make it as easy as possible to navigate between the different artifacts. This is primarily done by hyperlinking to the different artifacts, but ArchWiki also provides multiple entry points to the model by making the editors of the respective artifacts context-sensitive [22]: the user will see the connected architectural elements when editing a piece of source code.

3.2.7 Use of Meta Data and Model

Web 2.0 applications allow users to add meta data to artifacts or collect meta data themselves. This meta data is often used to enhance and enlarge the model. An example of this can be found on Facebook, where you can add information about the people which are visible on the picture and link them to their profiles, thereby enhancing the network. Some Web 2.0 applications are dedicated to adding meta data to artifacts. One class of Web 2.0 application are called social bookmarking websites. An example of such as website is Delicious,[3] which is used to tag and bookmark websites you encounter on the web. Other websites such as the online web shop Amazon.com[4] collect meta data about the purchases of items and recommend items to potential buyers based on purchases by previous buyers or even views by other users.

[3] http://www.delicious.com

[4] http://www.amazon.com

In ArchWiki it is also possible to add meta data to the artifacts, such as the recording of actions and corresponding users, adding tags to architectural elements or bookmarking certain architectural elements.

3.2.8 Action Tracking and Awareness

Awareness is an important part of many Web 2.0 applications. Applications such as blogs and micro blogs are basically all about awareness. With an application such as Twitter, users can keep their followers updated about their thoughts. In many other applications awareness plays an important role. For example, in Wikipedia users are aware of other contributions, so they can stop vandalism before it gets out of hand.

Awareness is also an important aspect in software architecture. Informal communication, such as water cooler conversations, often provides group awareness. This is more difficult in the case of distributed software development, because of the geographical and temporal distances [31]. In software development it is important that all stakeholders have the same picture of the software architecture. Differences could lead to architectural degradation, so to prevent this it is important that all stakeholders are aware of architectural changes as soon as possible. Another advantage of immediately being aware of changes is that the changes are bite-sized and easier to understand. Being aware of activity of other on the artifacts can also help users to assess the correctness of certain artifacts, such as documentation. A user might have more confidence in a piece of documentation if it is changed frequently and recently.

ArchWiki supports awareness through RSS feeds. It is possible to apply filters to items in the feed, such as filtering actions that only apply to certain elements or actions that have been performed by certain users.

3.2.9 Diverse Range of Contributions

Web 2.0 applications allow a wide range of contributions. In Facebook you can, for example, add text, photographs, your cycling routes, invitations to events, and many more types of contributions. In Wikipedia you can contribute a whole article, but it is also possible to only contribute a spelling mistake. This way Web 2.0 applications appeal to a wide range of users, who use the Web 2.0 application all in their own way.

ArchWiki also tries to support a wide range of contributions ranging from whole architectural documents or structures to individual architectural elements, paragraphs of text, or even adding a single tag. Supporting such a wide range of contributions will help users to personalize how to work with ArchWiki and allows them to easier fit it to their work flow. This also means that amount of information being passed between stakeholders is smaller and the passing occurs more frequent.

If it becomes normal to make small contributions, it may help to encourage users to contribute architectural knowledge as soon as it becomes available. This can prevent the loss of context and implicit knowledge, and avoids the opportunity cost of not providing the information earlier [51].

3.2.10 Utilizing The Long Tail and Crowdsourcing

Web 2.0 applications also utilize the so called long tail [3]. Brynjolfsson et al. [9] found that 30-40 % of Amazon.com's sales are in books that normally wouldn't be

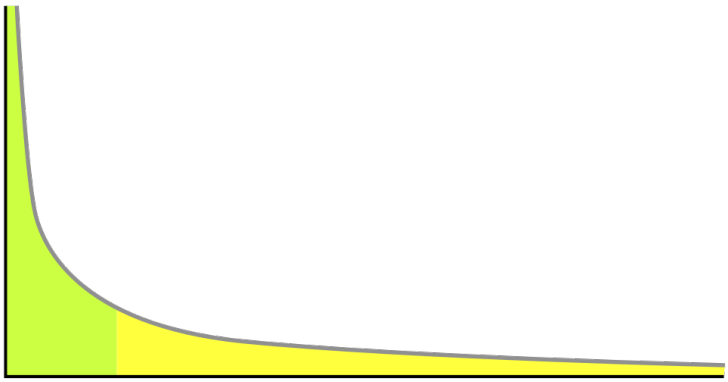

Figure 3.3: Utilizing the long tail

found in a brick-and-mortar store. In 2004 more than halve of the sales from Amazon.com came from book titles that were not available in the average Barnes and Noble store. It did so by attracting buyers that were interested in the not so popular books, but had no chance to buy them. Amazon used meta data such as reviews and purchase patterns of other users to bring those books to the users' attention and as such made use of the long tail. Google did a similar thing with selling ads: At least half of the ads they sold in 2005 was to advertisers which would not consider calling themselves advertisers before.[5] They did so by dramatically lowering the barrier to entry. In the case of Wikipedia higher degree of participation may have led to improved quality, owing to faster corrections of errors and omissions and more comprehensive views according to Parameswaran et al. [53].

ArchWiki also tries to utilize the long tail by lowering the barrier to entry by making the user experience as good as possible, offering a diverse range of contributions to the model, which consists of heterogeneous artifacts, and making it a social experience by including users and awareness. This way not only the software architects with comprehensive knowledge about the software architecture can contribute, but also other stakeholders such as developers with less extensive, but still valuable and specific knowledge. Figure 3.3 shows a graph visualizing the long tail. In the instance of software management the green color indicates the architectural knowledge of the software architects and the yellow color indicates the knowledge of other stakeholders such as developers. ArchWiki tries to utilize the long tail by addressing the other stakeholders and capture their knowledge, which may or may not overlap with the knowledge of software architects.

3.2.11 Feature Strengthening and Enabling

Many of the ideas mentioned above influence and strengthen each other or even enable each other. An application that shows traffic information in a text format might not appeal to a lot of users, because they need to know the names of the roads on their route.

[5] http://www.longtail.com/the_long_tail/2005/02/googles_long_ta.html

However if integrated with an application like Google Maps, where the information about traffic-jams are integrated with the maps themselves, it becomes a much more usable application. Some features enable other features. For example adding meta data to pictures enables navigation to the profile of the tagged user through a picture.

ArchWiki also tries to use features that strengthen or enable each other. An example of this is connecting different artifacts together, which in turn enables the navigation between the different artifacts.

3.3 ArchWiki Tool

To support our approach we have created the ArchWiki tool. The goal of the tool is to support software architecture knowledge management using the Web 2.0 concepts as mentioned in Section 3.2. As can be seen in Figure 3.1 ArchWiki takes the following approaches:

- Increase user experience.
- Move architecture closer to developer's everyday tasks.
- Increase architecture awareness.
- Make architecture knowledge management a social experience.

To achieve these things we focused on two solutions:

- Integrate different tools and artifacts within the IDE.
- Make the tool multi-user with social features and make it easy to use.

In the following subsection we will give a general overview of the ArchWiki tool and how mentioned solutions are implemented.

3.3.1 General Overview

We have chosen to use a Model-View-Controller (MVC) architectural style for our application. Figure 3.4 shows a layered diagram of the architectural style and the different roles and responsibilities of each of the elements and their dependencies:

- The view is an instance of the IDE and shows the common tools such as the source code editor and architecture structure editor. ArchWiki adds extra GUI elements to the view such as the Element View which is used to visualize the integrated ArchWiki model. It uses the controller to access the model and provides context in the form of the current state of the visualization. Every user has its own IDE and instance of the view and as such there can be multiple simultaneous views at the same time.

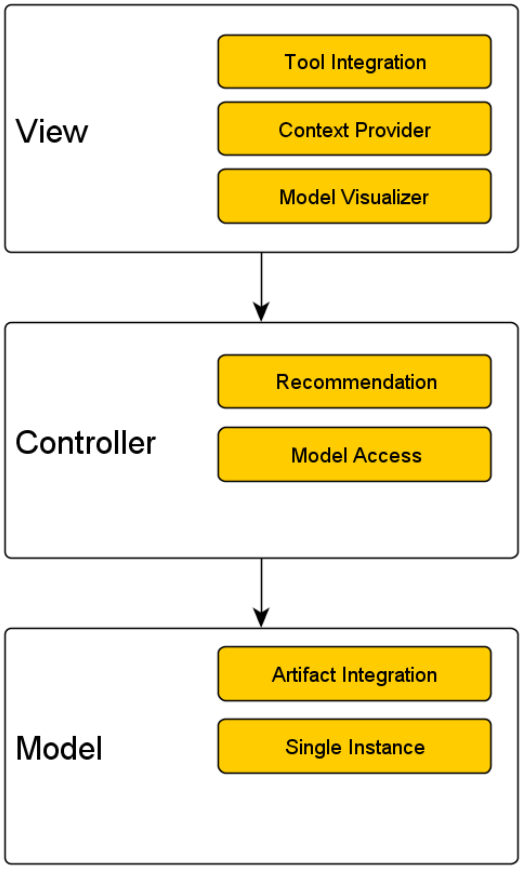

Figure 3.4: ArchWiki Architecture

- The controller has direct access to the model. Views can issue requests to the controller, such as the visualization of an architectural element, upon which the controller will access the model to find relevant data such as the connected elements, tags, etc.
- The model contains all the artifacts used by the tools in the views. For proper integration between the different tools their respective artifacts also need to be integrated. This integration is done by interconnecting all the artifacts within the model. There is only a single instance and a such every user will have the same and most up to date view on the model.

In the next two subsections we will discuss the ArchWiki model.

3.3.2 Archwiki Model

An important part of the ArchWiki tool is the ArchWiki model. This model is important for the integration of the different tools, making ArchWiki a multi-user application and adding social features. It is used to connect all the different artifacts, which are contained in the model, and forms the basis of the artifact traceability. The model con-

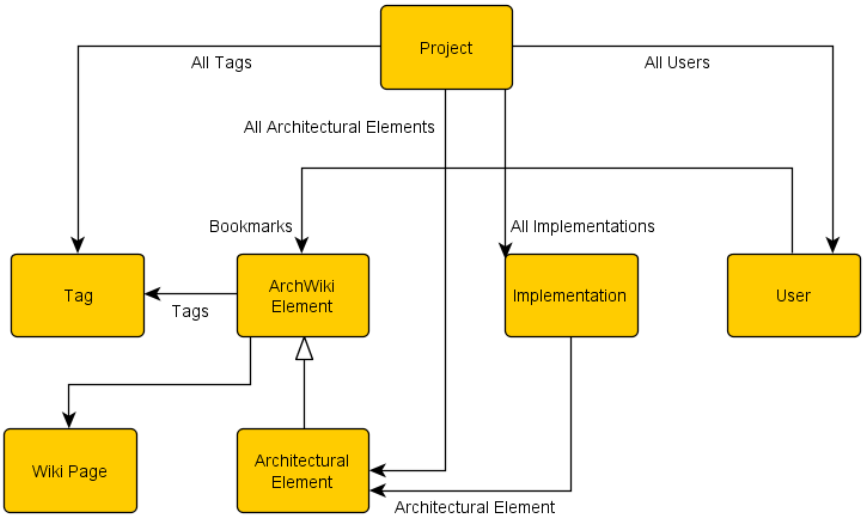

Figure 3.5: The ArchWiki internal model

sists of both internal ArchWiki artifacts, mainly used for social features, and external artifacts, for integrating the tools.

The internal artifacts are artifacts native to ArchWiki. Examples of these artifacts are:

- Users
- Actions
- Tags
- Bookmarks

External artifacts are artifacts that are not native to ArchWiki and are created outside of ArchWiki. Currently implemented external artifacts are:

- MediaWiki pages
- Architectural elements from structure diagrams
- Implementation

The difference between internal and external artifacts is that internal artifacts are fully contained in the ArchWiki model, while the external artifacts only contain a reference to the artifact.

We have chosen to make the architectural element the central artifact in the model. This means that other elements connect to the model through an architectural element. The architectural element is a sub class of an ArchWiki element, that can be enriched with Tags and Wiki Pages. In Figure 3.5 you can see the ArchWiki model. In this model the Architectural Element is the central artifact in the model and other artifacts such as textual documentation and implementation connect to the model through this element.

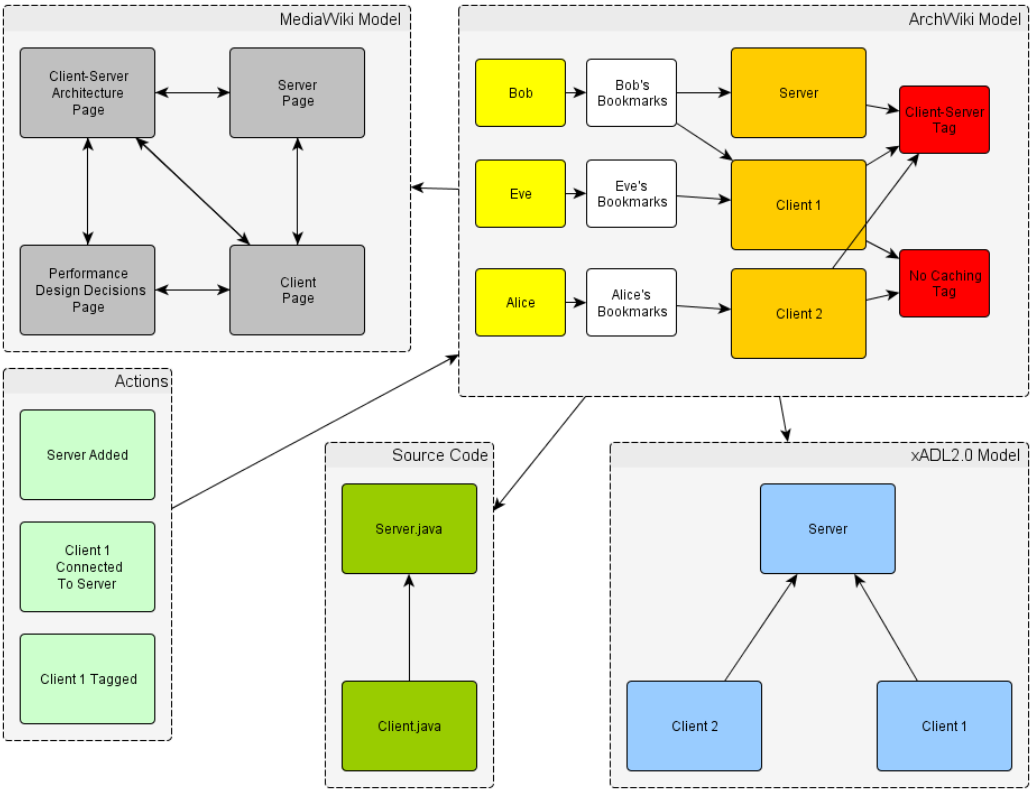

Figure 3.6: An example of a complete ArchWiki Model

External Artifact Mappings

Because we include external artifacts in the ArchWiki model we need to provide a way to map these external artifacts to the ArchWiki model. These external artifacts form models themselves. For example, the structure diagram in Archipelago is model with components connected to other components with connectors. Also the Pages on a MediaWiki server also form a model of Pages that hyperlink to other pages.

Figure 3.6 shows the mappings of the external artifacts to the ArchWiki Model. Because the architectural elements are the central elements in the model the IDs are stored in the internal ArchWiki model. The other elements are connected to these IDs. However, when we speak of the ArchWiki model we will mean the internal ArchWiki model and the models of the external artifacts combined.

3.3.3 ArchWiki Controller

The ArchWiki Controller is responsible for providing the View with data to visualize. For example, if a user clicks on a component in the architecture structure editor the view will send request for the element view of that component. Upon receipt the controller will access the ArchWiki model to find relevant information such as the connected components, tags, concrete implementation and textual documentation. The Controller will collect this information and send to the view to be visualized in the

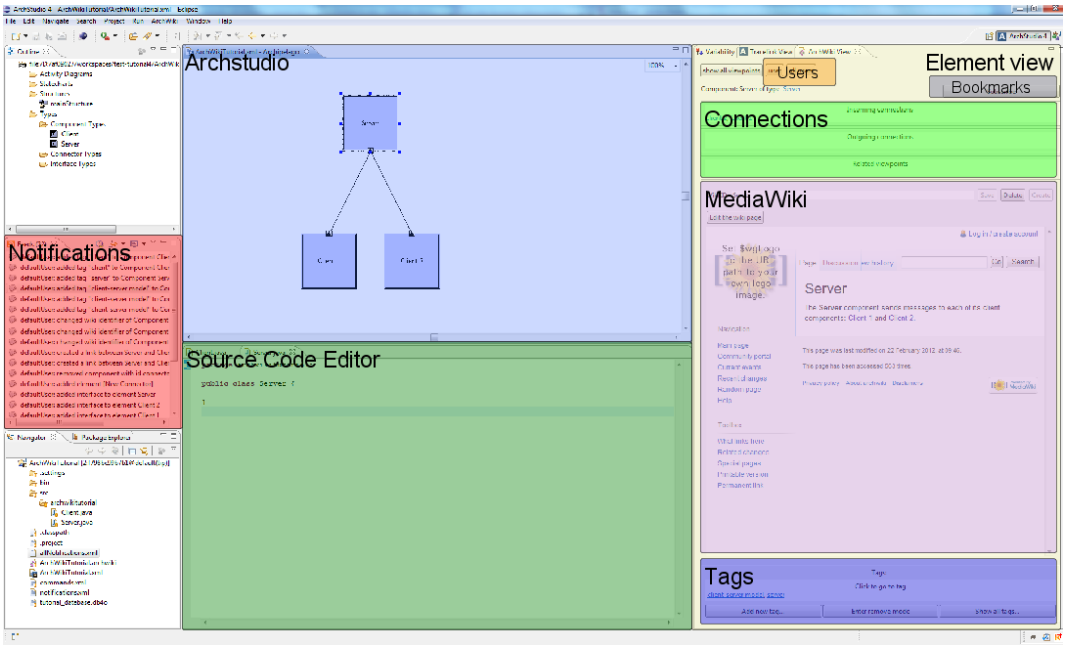

Figure 3.7: ArchWiki user interface

Element View. The Controller is also responsible for manipulating the model and will do so on request of a view. The Controller has an additional ability to record the actions of the users and persist these actions in the ArchWiki model.

3.3.4 ArchWiki View

For the view we use the Eclipse IDE, because it is a popular development environment and the integration can be easily implemented by means of a plug-in. We use it in combination with the ArchStudio 4 tool suite (Section 6.1.2). ArchWiki currently integrates the Java source code editor, the Archipelago architecture structure editor and the MediaWiki browser.

In Figure 3.7 the different GUI Elements of ArchWiki are shown overlayed with different colors.

- Java source code editor (green)
- ArchStudio's Archipelago structure editor (blue)
- MediaWiki browser (pink)
- RSS feed reader (red)

ArchWiki adds its own elements such as the Element View (yellow), which shows an architectural element as the central artifact and its connections to other artifacts in the ArchWiki model. It provides textual architecture documentation embedded in the form of a MediaWiki page. The Element View also provides bookmarking features and tags. The Element View functions as a central hub and provides access to artifacts connected

to the architectural element. Access to those elements is provided through hyperlinks and will switch to a view depending upon the type of the underlying artifact:

- Clicking on a tag in the tag area (purple) will switch to the tag view, which will show all other architectural elements with the same tags.
- Clicking on a hyperlink in the MediaWiki browser (pink) will make the browser change the wiki page. If the ArchWiki Controller detects that the wiki page is connected to an architectural element it will switch to the Element View of that element.
- Clicking on a architectural element in the connections area (green) will change the Element View to that architectural element

These GUI elements are integrated with the IDE, which allows the Element View to be context-sensitive: using the source code editor (green) or the Archipelago structure editor (blue) will make the Element View switch to the corresponding architecture documentation. The Element View also uses locality showing links to the connecting elements in the structure editor. Browsing the ArchWiki model can be done through hyperlinks and a specific editor will be opened according to the underlying artifact type of the link. For example, clicking on the source code link will open the source code editor. Awareness is provided by notifications, which users can access with their preferred RSS reader (red).

Users can customize their experience by bookmarking architectural elements (grey) and setting filters for notifications such as only showing notifications regarding bookmarked architectural elements. They can access their bookmarked elements by going to the user view (brown).

3.4 ArchWiki Features

Figure 3.8 shows a layered diagram of ArchWiki's features mapped on the architecture diagram of Figure 3.4. In this section we describe the ArchWiki features in detail and we describe how the features are enhanced with Web 2.0. To illustrate how the ArchWiki features can be used in software development we present the features through a use case scenario.

3.4.1 Use Case Scenario

In this section we will describe ArchWiki and its features in detail by presenting a use case scenario consisting of elements of different work flows. In this scenario we will highlight instances where the architecture centric approach and ArchWiki could support architecture knowledge management. The highlighted parts refer to the subsections where the ArchWiki approach for that respective instance is elaborated further. The scenario is:

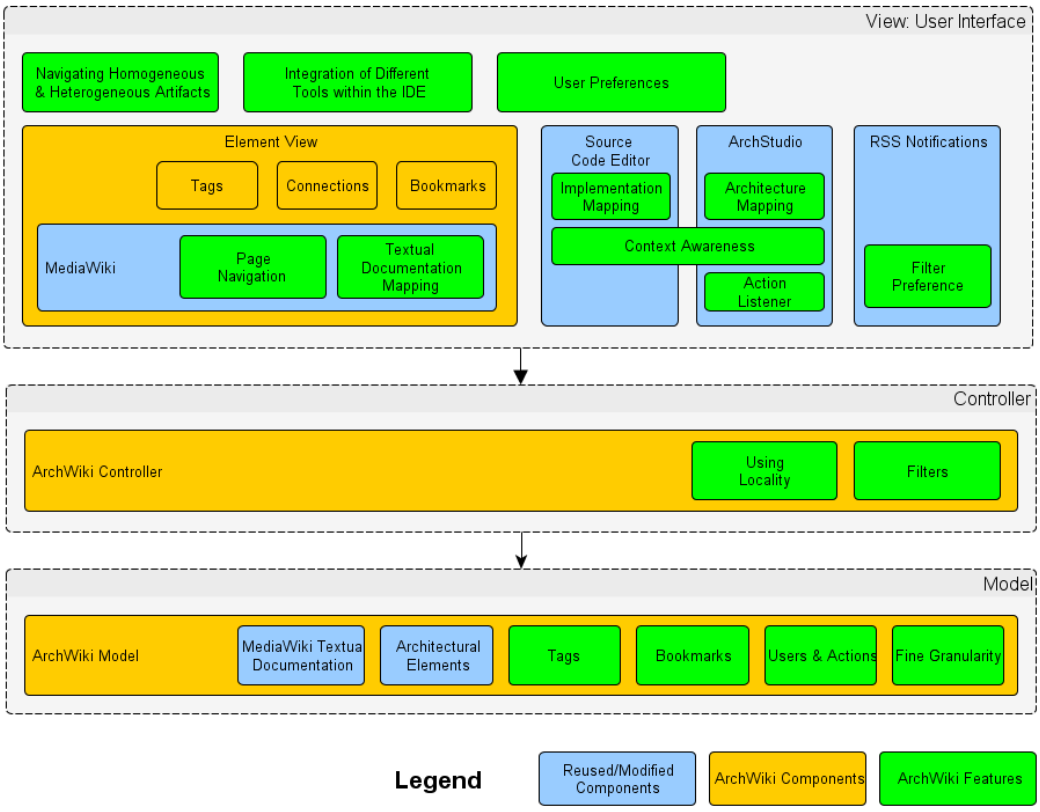

Figure 3.8: ArchWiki Features

Software architect Bob brainstorms about a new to be developed software system. The requirements engineers have written up a requirements document in which one of the requirements state that centralizing the data is an important requirement. He uses a **whiteboard and a marker (Section 3.4.4)** to create a high level overview of the system. He decides to use a client-server model and creates a structure diagram on the whiteboard. The main reason for choosing a client-server model, instead of, for example, a peer-to-peer model is that the data is centralized. He writes up this rationale for choosing the client-server model in a **Word file (Section 3.4.5)** and **saves it (Section 3.4.2)** to his computer.

Bob calls in a meeting and tasks are assigned to begin building the system. Eve is tasked with building the client component. At this point the architectural considerations are rather small and Bob confers the relevant consideration to each developer personally. For Eve this means that she must not store local data, because all the data must be centralized on the server. While Eve is implementing the functionality she notices that the implementation is rather slow. She considers to do client side caching to enhance the performance, but she remembers the "centralized data" requirement and **decides (Section 3.4.5)** to keep the code as it is. She

does not document this **small performance concern (3.4.14)**, because she thinks it is more trouble than it's worth.

The software system is constructed and years later Bob and Eve have left the company and new developers such as Alice have joined. Unfortunately the architecture has become degraded, partly because Bob was the one that **synchronized all the different architecture documentation (Section 3.4.2)**, but after he left no one picked this up. During a developer meeting Alice is assigned the task to create additional functionality for the client component. Alice has read the software architecture documentation and requirements documentation in the past, but decides not to read them again because during development she noticed that the implementation was not as was described in the documentation, and therefore she has no confidence in the accuracy of the architecture. Also the documentation is very large and it is hard to **identify relevant documentation (Section 3.4.8)** for her current task. She has access to two large documents: One large UML graph and one large Word documents. She wishes that the architecture documentation would come in a **smaller granularity (Section 3.4.10)** such as only the client component in the structure diagram and the textual documentation about the client component, so it would be much easier to handle. Documentation for connected components would also be interesting, because architectural decisions would be more likely to influence components that are in **the local vicinity (Section 3.4.11)** of the client component in the structure graph. Even if she does find the relevant information she cannot really tell if the information is still valid. If she had known **when the information was created and by whom (Section 3.4.12)**, she might have more confidence in the information if it was recent or she at least could have asked that person about more information. However this information is not available and she is also not in the mood to look for all the documentation and to install Word and an UML editor if she does find them **(Section 3.4.7 and** 3.4.3).

She starts **implementing (Section 3.4.6)** the features she is tasked to implement, but she unknowingly breaks the requirement that the data must be centralized by caching some of the data to perform operations client-side. Alice does document the data caching in the source code with Javadoc. Unfortunately no one that is **aware (Section 3.4.13)** of this particular architectural constraint reads the committed source code and Javadoc and the break in requirements is not spotted for a long time.

Such an architectural violation could have been avoided if Alice was more aware of the software architecture. However the sheer size of the software system, its architecture and the design decisions that shaped it made it hard for her to stay up to date to architectural aspects relevant to her. Quickly reading up with the current architecture is tedious, because **the architecture spans several documents all in different format (3.4.9)** and thus requires different tools. The implemented architecture has now further deviated from the intended architecture.

3.4.2 Single Documentation Instance

The software documentation is currently persisted as a normal file, which means that if new documentation is written it has to be synchronized with the current files. Also Bob has to think where to save the files and if people make a copy and modify that copy it has to be synchronized with the originals. ArchWiki treats the documentation as a single documentation instance, which means that everybody interacts automatically with the same documentation. This means that users do not have to spent time on where to save or find the documentation. They also do not have to wonder if they are using the current version of the documentation, because there is only one version of the documentation. Another consequence is that changes to the documentation do not have to be synchronized and are immediately accessible by everyone.

The main idea behind this feature is to support the collaboration features. Collaboration and many of its features require that there is only a single instance of an artifact. For example, the single instance addresses the problem of sharing documents by getting rid of operations dealing with finding or storing documents, which also lowers the barrier to entry. Many of the successful social computing platforms make collective content creation and sharing a creative adventure for the mainstream user, in which the user doesn't have to worry about actions other than adding content [53]. A single instance of the documentation also signifies the importance of the architecture document and raises the confidence in its correctness and validity.

Web 2.0 concepts related to this feature are:

- Feature Strengthening & Enabling
- Collaboration
- Meta Data & Model
- User Experience

3.4.3 Integration of Different Tools within the IDE

Bob does not use a specific tool to create his diagrams. He prefers to use a whiteboard and marker to create diagrams and uses his digital camera to capture the diagram. To capture his design decisions and provide other textual documentation he uses Word documents. There are more ways to create documentation such as XML and UML documents. Currently all these documents require their own tools to read and edit the documents. The ArchWiki way is to integrate all these tools into the IDE so users can switch between documents without problems.

Although the ArchWiki approach supports the architecture centric view and this view requires some change in work flow, it tries to be compatible with the current work flows. It integrates tools that support creating architectural diagrams and textual documentation. Figure 3.9 shows the different tools ArchWiki integrates into a single IDE:

- Source code editor in the top center
- Architecture structure editor in the bottom center

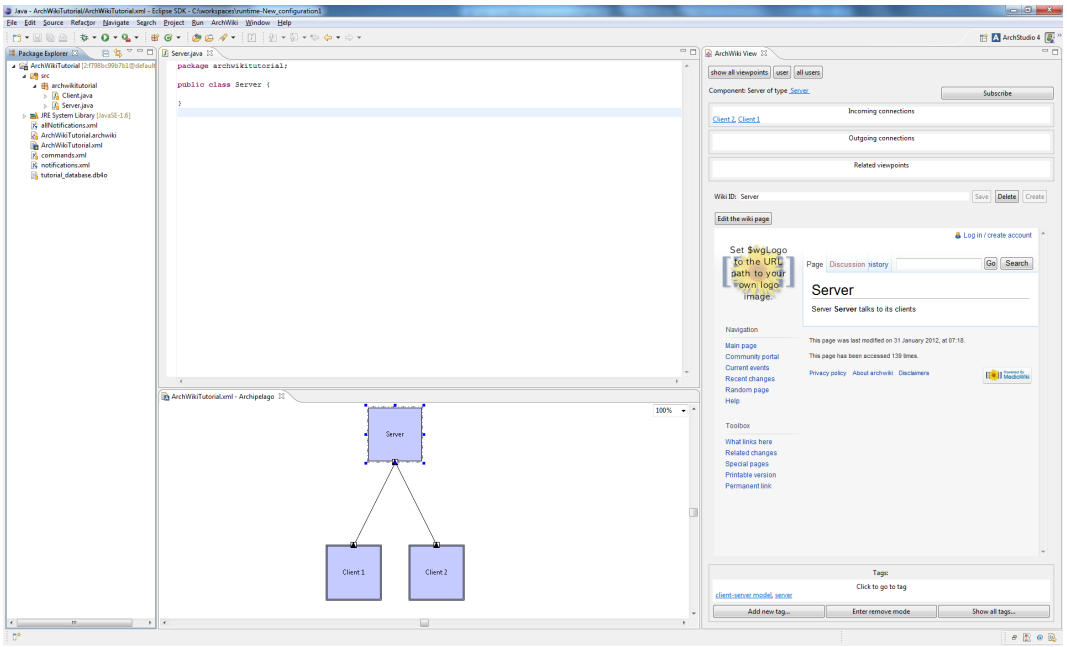

Figure 3.9: ArchWiki Tool Integration in the IDE

- Textual documentation on the right

ArchWiki's integration with the IDE has the goal to make ArchWiki feel like a single application, although it is technically a mashup of different tools. Part of this feel is that users are not required to open each document manually with a tool, but can naturally reach the contents of software architecture documentation based on their current context. This provides a user friendly way to navigate the homogeneous and heterogeneous artifacts of the model. Navigation within ArchWiki will be described in more detail in Subsection 3.4.9.

ArchWiki lets the user incrementally build up and integrate content, which will be described in the subsections about the integrated tools and the Element View (Subsection 3.4.8). Aside from the artifact mapping the integration also consists of recording of actions within the integrated tools to provide awareness (Subsection 3.4.13) and providing context.

Web 2.0 concepts related to this feature are:

- User Experience
- Feature Strengthening & Enabling
- Diverse Range of Contributions
- Meta Data & Model

3.4.4 Visual Documentation: ArchStudio

ArchWiki has chosen to use the ArchStudio tool suite to create the visual documentation such as structure diagrams. The main reason is that the diagrams are written in a well defined architecture description language xADL 2.0. Such a language makes it

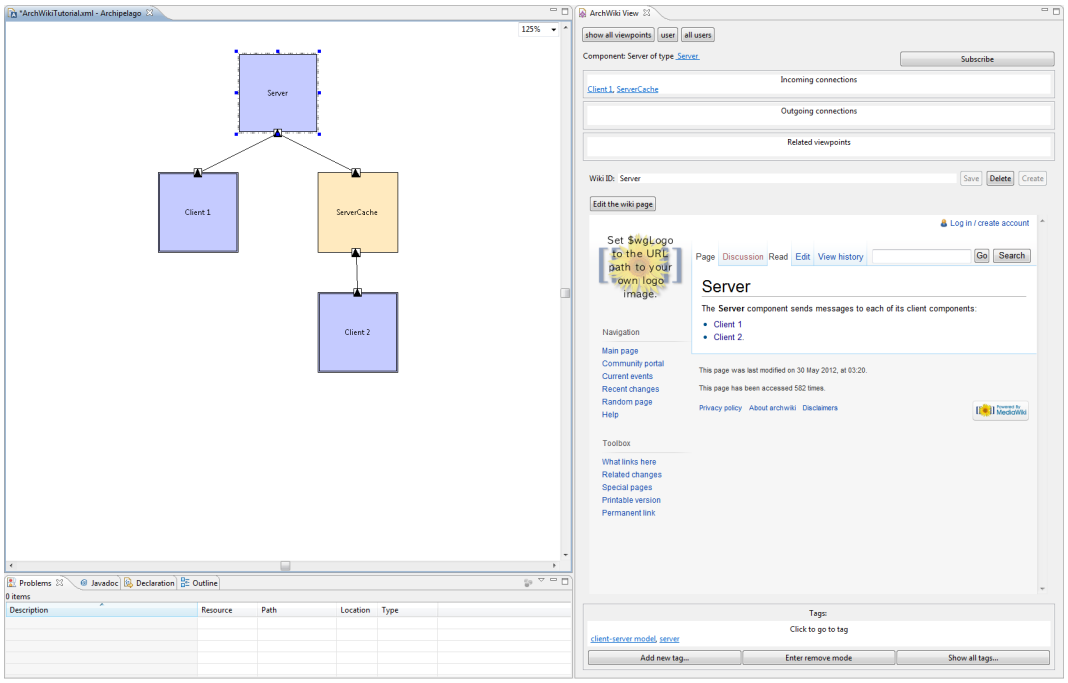

Figure 3.10: ArchStudio's Archipelago structure editor (left) and Element View (right)

easier to support software architecture evolution. Another reason we have chosen to use ArchStudio is that it is already integrated with the IDE and that it is open source, which makes it easier to integrate with ArchWiki.

Figure 3.10 shows the Archipelago editor. When an architectural element is clicked, such as the server, the corresponding Element View will be shown. This way the textual documentation can easily be found by just clicking the elements. If there is no wiki page a new page can be created from the element's context menu.

Web 2.0 concepts related to this feature are:

- Easy Navigation
- User Experience
- Meta Data & Model
- Action Tracking & Awareness

3.4.5 Textual Documentation: MediaWiki

Bob uses a Word document to document the architecture in a textual format. Unfortunately Word documents are not designed for collaborative editing. If someone would want to read the documentation then Bob will have to send them the document. When another person modifies the document then Bob has to manually merge it with his own version. This requires a lot of work from Bobs part, so coworkers can be hesitant to change the document. An example of this is when Alice makes the decision to not

Figure 3.11: ArchWiki autocompletion when linking wiki pages to the model

use caching for the client implementation, but does not document it, because it requires getting the document from Bob, figuring out where in the document to place her design decision, and then sending the document to Bob for merging.

For ArchWiki we have chosen to use MediaWiki[6] to document the architecture in a textual format. MediaWiki is the software used to power Wikipedia and other wiki sites. Bachmann and Merson [7] found using a wiki as a communication tool for software documentation to be promising. It gives us several advantages over Word files.

- Wikis are not file based, but web based. This means that Bob does not have to send people the Word file anymore. Everybody has access to the same instance of the documentation. This has as a consequence that everybody always has the most up-to-date version of the documentation and that Bob does not have to merge documents anymore.
- Wikis are naturally divided up into pages, which makes it easier to access relevant information.
- Wikis are naturally version controlled, which makes it easy to revert changes.
- Design decisions are often the result of informal communication and lightweight tools such as a wikis are a good match for such a setting.

The MediaWiki wiki page browser is embedded in the Element View and shows the textual documentation for that architectural element. There are two ways to link a wiki page to the ArchWiki model:

- Create a new wiki page which will get the same name as the architectural element it is created for. This can be done from either the Archipelago structure editor or the Element View.
- Link an existing wiki page to the model from the Element View as can be seen in Figure 3.11. ArchWiki provides a list of available wiki pages to help the user map the textual documentation to the model.

It is possible to access the wiki pages from a normal browser, because a wiki page used by ArchWiki does not contain any ArchWiki specific content. On top of the embedded

[6] http://www.mediawiki.org/wiki/MediaWiki

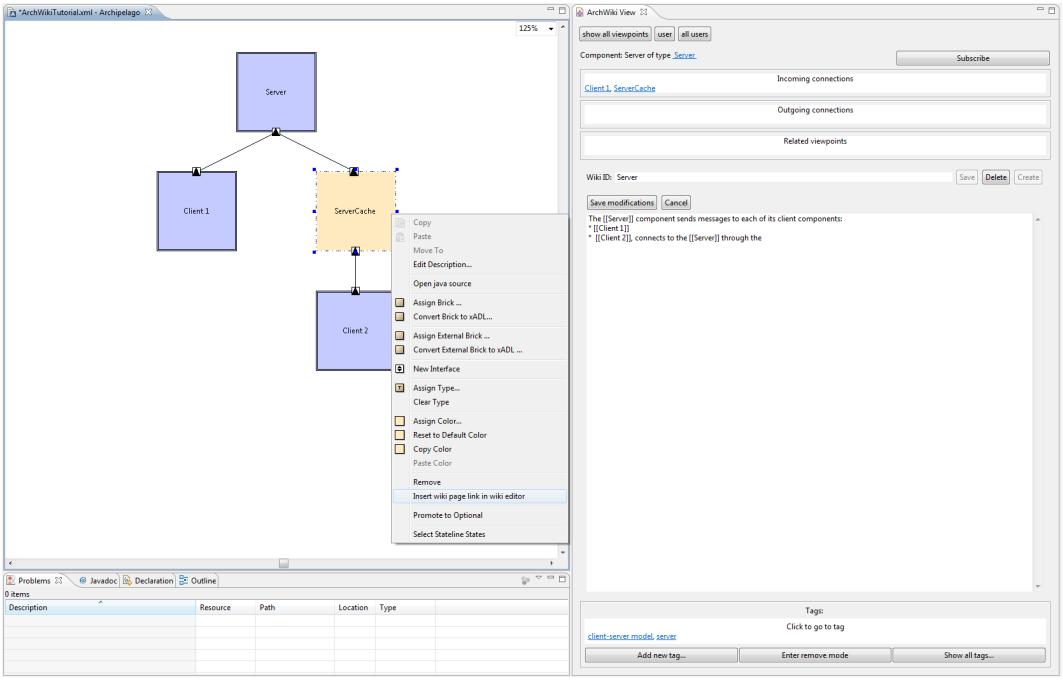

Figure 3.12: Using Archipelago to insert a link to a architectural in a wiki page

browser we have created extensions to further integrate it in ArchWiki. When clicking on a hyperlink within the browser, ArchWiki will analyze the link and search for it in the ArchWiki model. If the wiki page is currently connected to an architectural element, the whole Element View will change to that element instead of just a browser change. Another extension is the editor in which the user can edit the textual documentation. If a user is editing the textual documentation he can create a hyperlink to an architectural element from the Archipelago editor. For example, in Figure 3.12 a link to the ServerCache component is created in the textual documentation of the server component. Other actions such as deleting link to wiki pages are also supported.

These actions for creating, deleting and changing of wiki pages can be saved in the ArchWiki model and users can be notified of these actions.

Web 2.0 concepts related to this feature are:

- User Experience
- Easy Navigation
- Meta Data & Model
- Action Tracking & Awareness
- Collaboration

3.4.6 Implementation: Source Code Editor & Implementation Mapping

Alice starts implementing the feature without first consulting the architecture documentation first. One reason for this is that she did not had an easy way to find the related architecture documentation. ArchWiki tries to address this issue by mapping the implementation in the form of source code to the architectural elements and thus

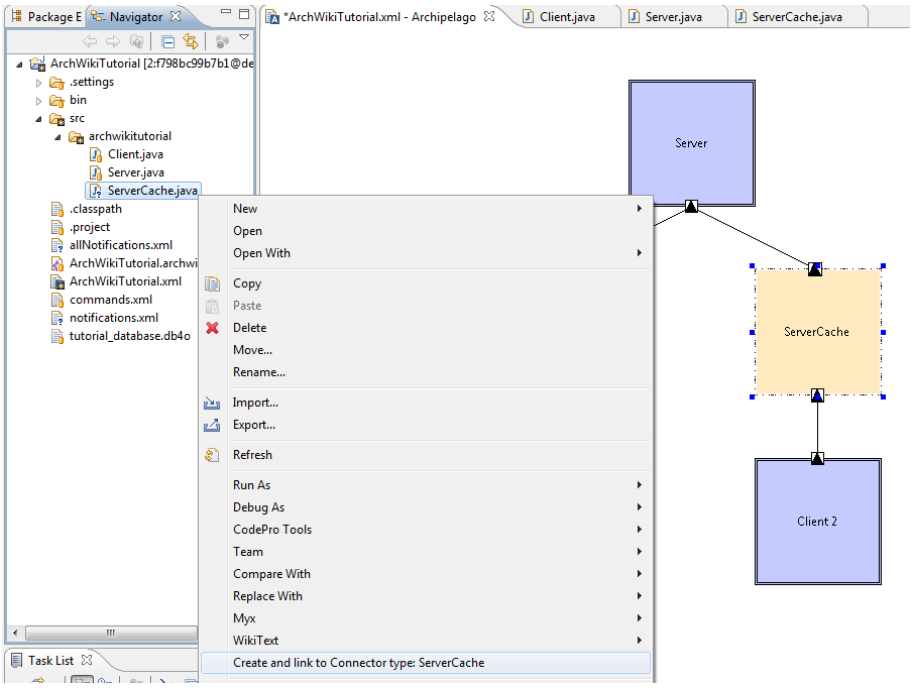

Figure 3.13: Link implementation from file navigator to the ArchWiki model

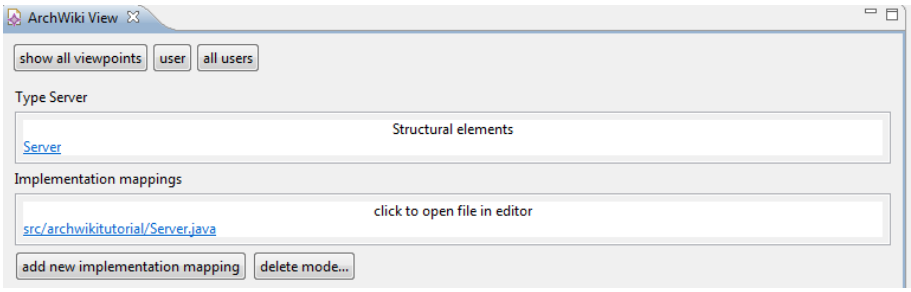

Figure 3.14: Manage implementation mapping from Element View

making it a part of the ArchWiki model. The implementation mapping provides traceability between source code and architecture and is part of the integration of different document types as described in Section 3.4.7 and enables the context-sensitive Element View (Section 3.4.8) when editing source code.

This implementation mapping enables the users to view related software architecture documentation. The Element View is automatically changed to the related documentation when editing source code. The creation of the mapping is one in two different ways:

- The easiest way is the context-sensitive way in which the user only has to select a file in the source code and map it to the currently selected architectural element in Archipelago (Figure 3.13).

- Manually add or edit the mapped filename in the Element View. From this view

it is also possible to delete implementation mappings (Figure 3.14) .

The architecture mapping is intended to increase the user experience by showing relevant architecture documentation during coding. This feature aims to increase program comprehension, which in itself prevents architectural degradation, because a user that understands the architecture is less likely to break the architecture. Another goal is making it easier to document the architecture. Many design decisions are made while implementing a feature. Having the documentation alongside the code will lower the barrier to contribute.

It also helps with program comprehension of other users, because more source code is documented. It is also possible that the quality of the documentation increases. The better quality results from more users that read the documentation, spot flaws in it and improve the quality by adding corrections as is the case with Wikipedia articles [53]. Wilkinson and Huberman [78] have shown that the high-quality articles in Wikipedia are distinguished from the rest by a larger number of edits and distinct editors. ArchWiki tries to raise both of these with the mapping of the implementation to the architecture documentation.

Web 2.0 concepts related to this feature are:

- User Experience
- Meta Data & Model
- Easy Navigation
- Action Tracking & Awareness
- Data & User Driven

3.4.7 Integration of Different Document Types

With ArchWiki, the user perceives the documentation as part of the ArchWiki tool, instead of a incoherent bunch of different file types, each of which needs a different external tool to open the file. To support this ArchWiki allows many types of files to be opened without looking up their location and manually opening them. The different tools are already integrated in ArchWiki (Subsection 3.4.3) and clicking on a hyperlink in ArchWiki will open the appropriate editor for the artifact.

The integration of the different tools is based on the integration of the different artifact types. The different artifacts that are contained in the model are described in Figure 3.6. For the ArchWiki model we have chosen to make architectural element the core element in the ArchWiki model. Other artifacts connect to the ArchWiki model through an architectural element.

The mapping of artifacts to the ArchWiki model can often be done from both directions:

- from their respective editors
- from the Element View, which visualizes the ArchWiki model from the viewpoint of an architectural element

Web 2.0 concepts related to this feature are:

- User Experience

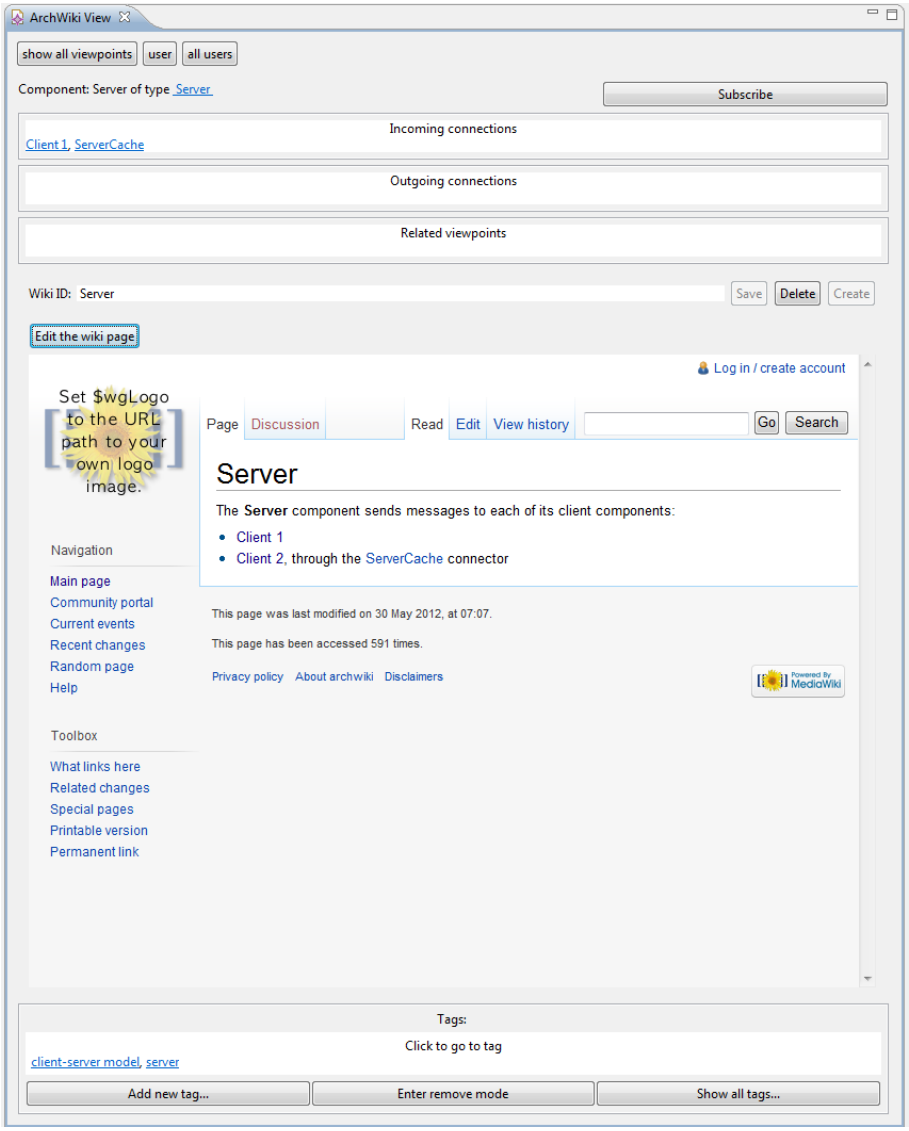

Figure 3.15: The Element View GUI

- Meta Data & Model
- Easy Navigation
- Data & User Driven
- Feature Strengthening & Enabling
- Diverse Range of Contributions
- Collaboration

3.4.8 Context-Sensitive Element View

If Alice would have had all the relevant architecture documentation in view while she was editing the source code, she would at least have been forced to read the architecture documentation in which Eve explains her rationale for not using caching to increase

the performance.

ArchWiki mainly uses the Element View to show architectural documentation. This view is connected to an architectural element defined in the Architectural Description Language xADL2.0. This architectural element is the central artifact in the ArchWiki model and as such, other artifacts, such as source code and tags, are directly connected to an architectural element. This allows the Element View to switch to the architecture documentation based on the context. Currently the Element View changes its view based on currently opened source code editors and the selection of architectural elements in Archipelago.

Figure 3.15 shows the current information displayed on this view:

- name and type of the architectural element
- incoming and outgoing connections with other architectural elements
- tags
- wiki page with textual architectural documentation.

Information on other artifacts in the model are fully navigable. The user can click on types, names of incoming and outgoing connections, tags, and links within wiki pages to navigate to the appropriate Element View or editor. For example, clicking on the ServerCache hyperlink in the browser will change the Element View to that of the ServerCache and clicking on the implementation in Figure 3.14 will open the source code editor for the implementation.

Traceability is used in two ways with the Element View. First, the Element View is obtained through the use of an connected artifact, such as source code. Second, the Element View shows related artifacts and can be seen as a recommendation feature.

Web 2.0 concept related this feature are:

- User Experience
- Diverse Range of Contributions
- Meta Data & Model
- Easy Navigation
- Data & User Driven

3.4.9 Navigating Homogeneous and Heterogeneous Artifacts

When documentation becomes larger it becomes more difficult to read and comprehend. Current tools are not really equipped to handle different document types and the documents themselves do not refer to other artifacts. ArchWiki has been designed with this navigability in mind. Navigation is possible between homogeneous artifacts such as navigating from an architectural element to an element it is connected to, but also between heterogeneous artifacts such as from an architectural element to a tag or implementation. This is made possible by the traceability between the artifacts.

Figure 3.16 shows the navigation path in the ArchWiki GUI. ArchWiki is designed to have multiple paths to reach a desired artifact to accommodate the different contexts the users operate on. For example, the user can reach the Element View of the server component using the following paths:

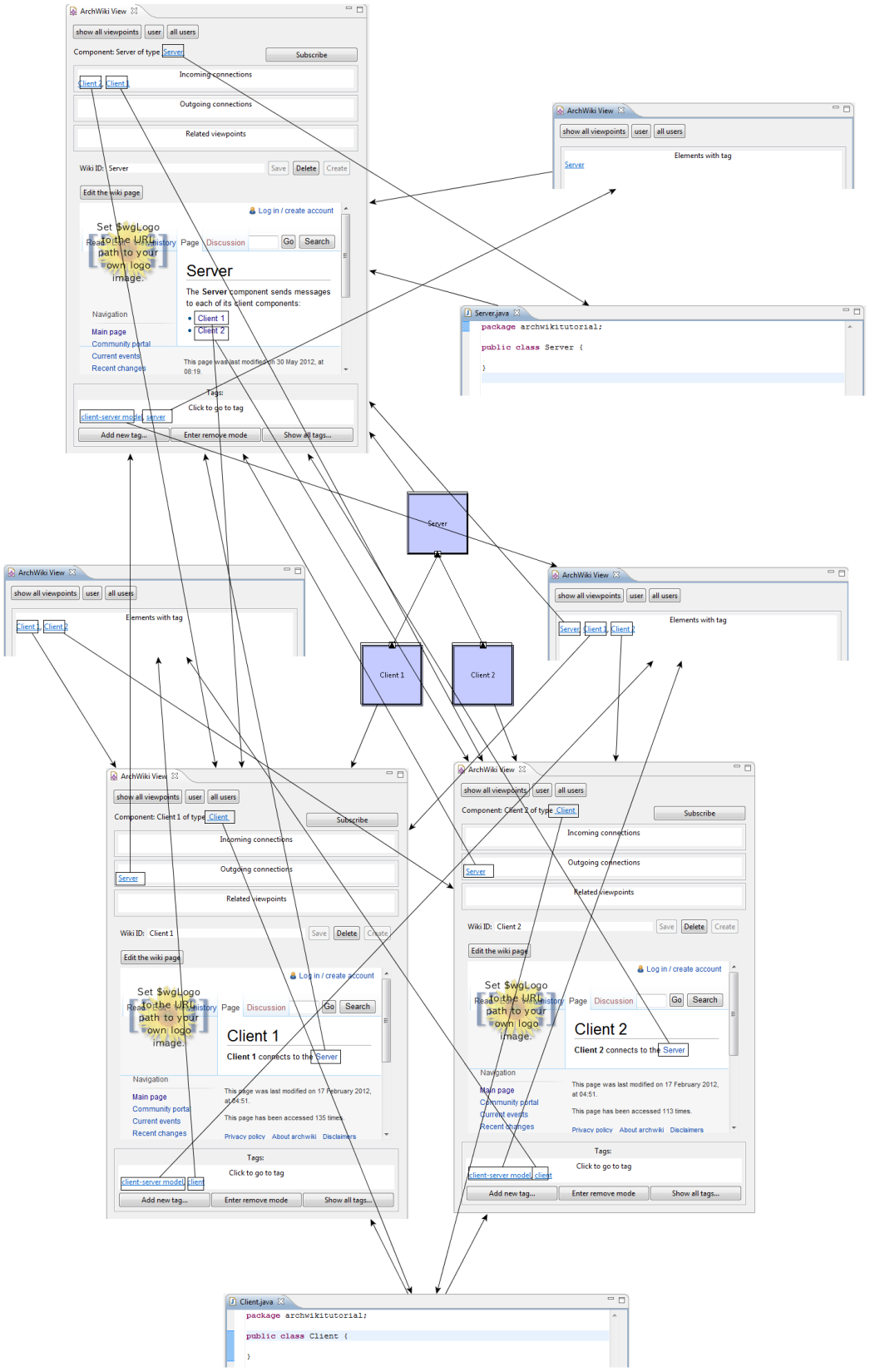

Figure 3.16: Navigation paths in ArchWiki

- In the structure diagram clicking on the server component
- In the file browser opening the "Server.java" source code file
- On the Wiki page by clicking on the hyperlink in the textual documentation
- From either the Element View of the Client 1 or Client 2 component clicking on the connection with the Server
- In the tag view of either the "client-server model" or the "server" tag by clicking on the "server" hyperlink

Web 2.0 concepts related to this feature are:

- Easy Navigation
- Meta Data & Model
- User Experience
- Data & User Driven
- Action Tracking & Awareness

3.4.10 Finer Granularity: Breaking Down Artifacts Into Smaller Pieces

ArchWiki breaks the different artifacts down into smaller pieces and operates on the level of these smaller pieces. This differs for each artifact type. For example, a structure diagram in xADL2.0 can be broken down to components and connectors. The textual documentation on a Wiki can be broken down to pages. Source code in an object oriented language can be broken down into classes. How the artifacts are exactly broken down can be seen in Figure 3.6.

Operating at a finer granularity level has many benefits:

- ArchWiki is able to match how users operate and allows for higher precision by allowing the user to interact with the architecture at a component level instead of being only presented with the whole architecture, which can be quite intimidating and discouraging.
- Breaking the artifacts into smaller pieces also allows ArchWiki to use locality as described in Subsection 3.4.11.
- Modifying an artifact does not affect other artifacts. This helps preventing conflicts such as when two modifications are done concurrently to different parts of a Word file. The two modifications have to be merged manually before a contribution can be added. In the Wiki case this does not have to be done, because the changes are done to two different pages.
- It is a possible to allow for a finer granularity of awareness as described in Subsection 3.4.13.

Web 2.0 concepts related to this feature are:

- User Experience
- Diverse Range of Contributions

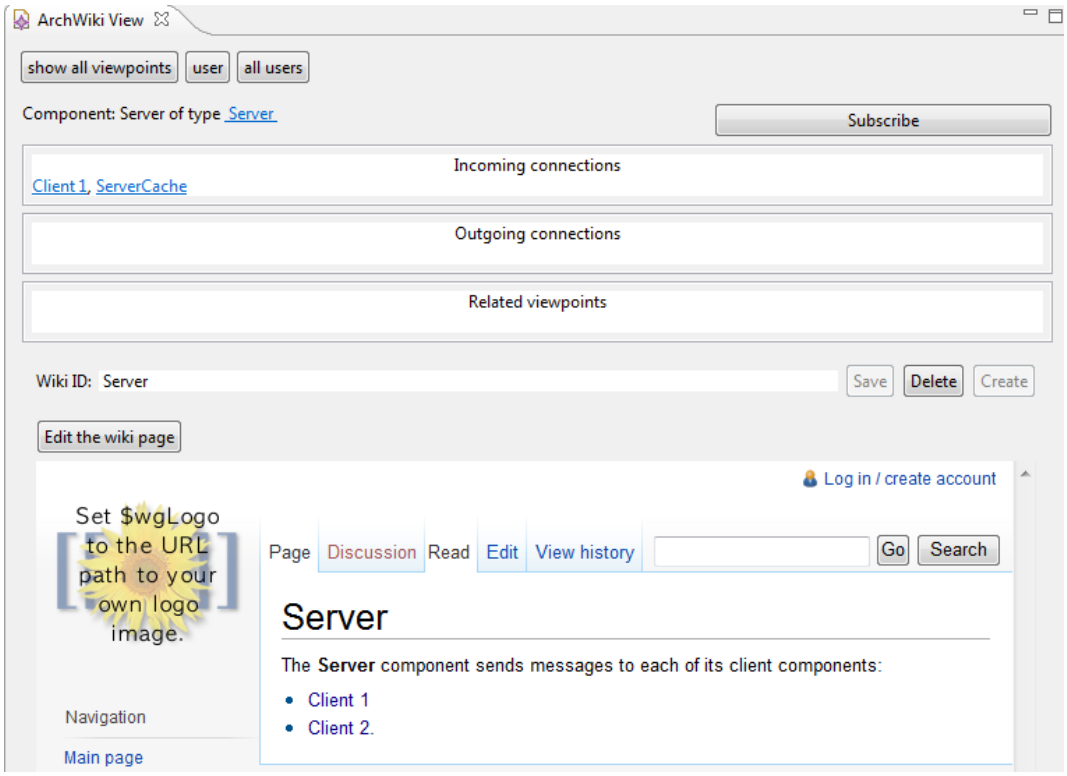

Figure 3.17: Using locality when reading textual documentation

- Meta Data & Model
- Easy Navigation
- Action Tracking & Awareness
- Collaboration

3.4.11 Using Locality

The ArchWiki model is essentially a graph where vertices represent architectural artifacts. Because of this graph representation, the locality within the structure can be exploited. In Figure 3.17 a user may read the wiki page of the server to understand the components the server interacts with. However, the current text does not mention that the Client 2 component connects to the server through a proxy. This is apparent in the structure view and ArchWiki brings this to the user's attention in the connections area in the Element View, where the ServerCache component is shown. This also provides a good opportunity for the user to modify the textual documentation to include the use of proxies.

Web 2.0 concepts related to this feature are:

- Action Tracking & Awareness
- Meta Data & Model
- User Experience
- Easy Navigation

3.4.12 Meta data, Users and Actions

Software architecture tools often do not store meta data, such as who is using the architecture documentation and when changes are made.

ArchWiki's usage of meta data is one of its primary features. There are two types of meta data: content meta data and usage meta data.

Content meta data is added to artifacts. The mapping of an artifact to the model is an example of this. Artifacts can also have other meta data attached them such as tags (Section 3.4.14) and bookmarks (Subsection 3.4.15).

Usage meta data is data related to the usage of ArchWiki. Examples of usage are the addition of new components or even the viewing of an Element View. These actions can be recorded along with other relevant information, such as the user that performed the action, the time it was performed, and the artifacts that were involved.

The addition of meta data enables new features, such as:

- recommendation features, such as a feature that recommends which user could elaborate a certain part of the documentation
- awareness of actions of other users by means of notifications (Section 3.4.13)
- social classification by means of collaborative tagging (Section 3.4.14)
- bookmarking to further help organize the artifacts (Section 3.4.15)
- additional data mining possibilities
- possibility to explore or even move to a previous state of the architecture

Web 2.0 concepts related to this feature are:

- Users are part of the Application
- Meta Data & Model
- Utilizing the Long Tail & Crowdsourcing
- Data & User Driven
- Collaboration
- Feature Strengthening & Enabling
- User Experience
- Diverse Range of Contributions
- Action Tracking & Awareness

3.4.13 Awareness with Notifications

If a team member was more aware of what Alice had done and had spotted that Alice had violated an architectural constraint he could have notified her of her error and it could have been corrected early. The end result would be that Alice had gained valuable architectural knowledge and that the implementation was still in sync with the documentation. Instead, Alice lacked the architectural knowledge and the system became degraded.

ArchWiki offers awareness through notification of actions. Figure 3.18 shows the actions ArchWiki captures, which consists of internal and external actions. Internal

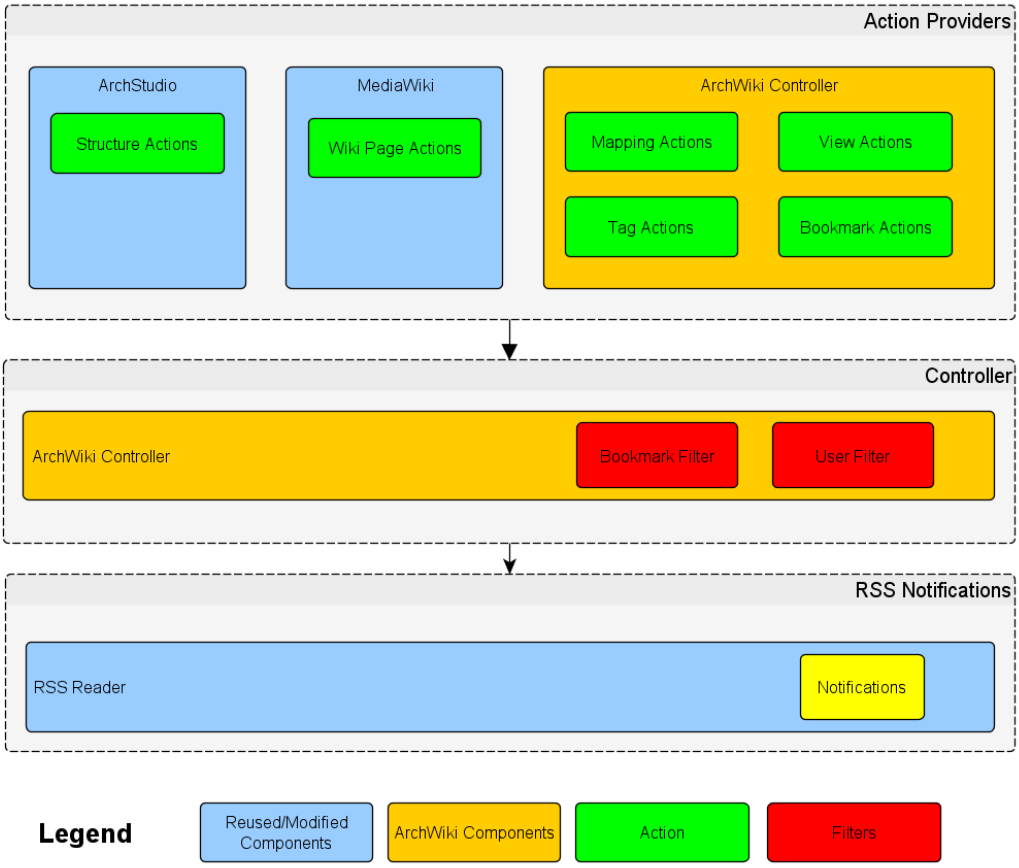

Figure 3.18: Actions and Notifications

actions are actions such as artifacts mapping, view, tagging and bookmarking actions. External actions are actions performed by external tools to external models, such as the actions from ArchStudio and MediaWiki.

Figure 3.19 shows notifications in an RSS/Atom feed, which can be filtered to the user's preference. A large user base can generate a lot of actions, many of which a user might not be interested in. A filter can help the user to only see relevant notifications. A user can, for example, configure ArchWiki to show only notifications of actions concerning elements that are bookmarked or actions that are originated from a particular user.

Web 2.0 concepts related to this feature are:

- Action Tracking & Awareness
- Collaboration
- User Experience
- Users are part of the Application
- Data & User Driven
- Meta Data & Model

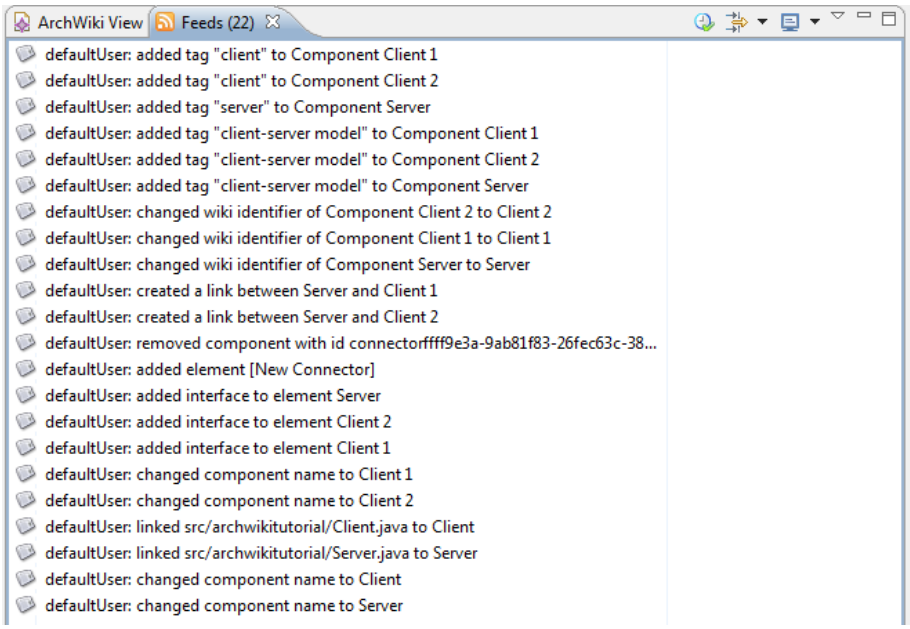

Figure 3.19: Using a RSS reader to receive notifications

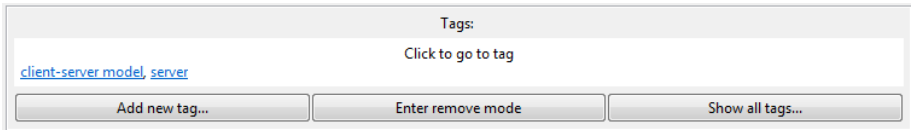

Figure 3.20: Tags of the server component

3.4.14 Tags

Alice did not document a design decision, because the decision was rather small. The documentation of this small concern could have prevented a serious degradation of the architecture which would have happened a long time later.

ArchWiki provides a lightweight way to document an architectural element by using tags. Tags are used to classify and describe architectural components. They are pieces of meta data which are part the ArchWiki model. Tags can point to multiple architectural elements, which makes them suited to document cross cutting concerns such as performance.

Figure 3.20 shows the tags of the server component. If the user clicks on a particular tag he will be presented with all the architectural elements with that particular tag. When adding a new tag the users are presented with a list of already added tags. For instance, a user that specifically want to gain architectural knowledge about performance concerns can use the "performance" tag to find all the elements with performance concerns. The tagging features thus helps with both documenting and retrieving the architecture.

Web 2.0 concepts related to this feature are:

- Meta Data & Model

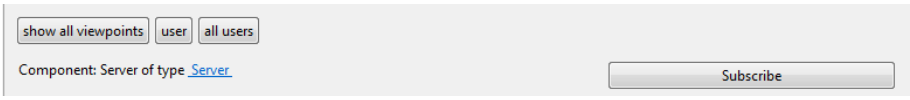

Figure 3.21: Bookmark the component by hitting the Subscribe button

- Easy Navigation
- User Experience
- Diverse Range of Contributions
- Collaboration

3.4.15 Bookmarks

Architectural documentation can become very large and bookmarking specific element in the ArchWiki model can help a user to manage the documentation to his specific needs. A user can use a bookmark to create easy access to that specific architectural element or he can configure a filter to only show notifications concerning elements that are bookmarked.

Users can bookmark architectural elements by hitting the subscribe button of its Element View as can be seen in Figure 3.21.

Web 2.0 concepts related to this feature are:

- Easy Navigation
- User Experience
- Users are part of the Application
- Action Tracking & Awareness

3.5 Summary

In this chapter we presented the ArchWiki approach. We started by stating our goals of enhancing architecture knowledge management, along with our approach and solutions to achieve those goals. The ArchWiki approach is heavily inspired by Web 2.0 and we explained how we apply Web 2.0 to software architecture management. We then presented the ArchWiki tool, which is developed to support the ArchWiki approach. A use case scenario of a current work flow of architecture activities was presented. For each activity where ArchWiki could be applicable in the use case scenario, there were references to detailed descriptions of ArchWiki features.

Chapter 4

Implementation

4.1 Introduction

In this chapter we will describe the concerns with the implementation of the ArchWiki tool that supports our approach. The main ArchWiki technologies are the Java Programming Language [4] and the Eclipse IDE [48]. The ArchWiki tool is implemented in the form of an Eclipse plug-in [15]. We have chosen these technologies due to our familiarity with them: Java as a programming language and Eclipse as an IDE. Also Eclipse is open source and written in Java, which makes these two technologies compatible with each other. Many of the technology choices of ArchWiki are determined by their compatibility with Java and Eclipse. ArchWiki can be divided into three architectural components of the model-view-controller pattern [25]:

- The Model
- The View
- The Controller

We will describe the implementation concepts, such as design patterns, technology choices, design considerations, for each of the components. We also describe features that are implemented in a less than ideal way due to time and resource constraints.

4.2 The Model

The ArchWiki model is one of the most important components of ArchWiki, mainly because it is a prerequisite for many other components of ArchWiki. In Section 3.3.2 we have described the ArchWiki model, which consists of internal and external artifacts. In the next subsections we will describe the implementation of the ArchWiki model in more detail starting with the internal artifacts and followed by the external artifacts.

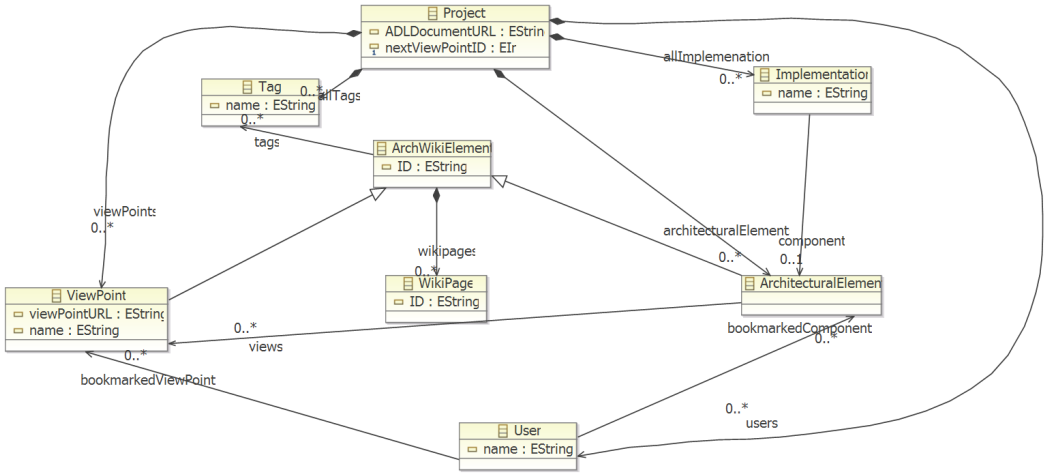

Figure 4.1: Ecore diagram of the ArchWiki model

4.2.1 ArchWiki Internal Artifacts: Eclipse Modeling Framework (EMF)

The ArchWiki model consisting artifacts is created with EMF [67] (Eclipse Modeling Framework). With this framework we have created a meta model in Ecore . EMF is able to generate Java source code to create, edit and access the model. Figure 4.1 shows the Ecore diagram of ArchWiki model. This model contains internal ArchWiki artifacts, such as users, bookmarks, tags and the project artifact in which all artifacts are contained. The model also contains bridge elements that represent how external artifacts map to the ArchWiki model. A bridge element contains a mapping to an external artifact. Bridge elements exists for wiki pages, architectural elements, and implementations. ArchWiki will look for the ArchWiki model in the model directory that is specified in the preference page.

4.2.2 ArchWiki External Artifacts

Figure 4.2 shows how ArchWiki links the ArchWiki Model with the external artifacts through bridge elements. In the next subsections we will describe these external artifacts.

Architectural Description Language: xADL2.0

We use the the Architectural Description Language xADL2.0 to create the architectural description of the structure with a component-and-connector diagram. xADL2.0 stores the architectural description in a file in an XML based format. The architectural elements defined in this language are the core artifacts in the ArchWiki model. The xADL2.0 components and connectors have a 1-to-1 mapping to the ArchWiki bridge elements. Bridge elements are automatically created for the architectural elements and the mapping requires no action from the user. ArchWiki will look for the xADL2.0 file in the model directory.

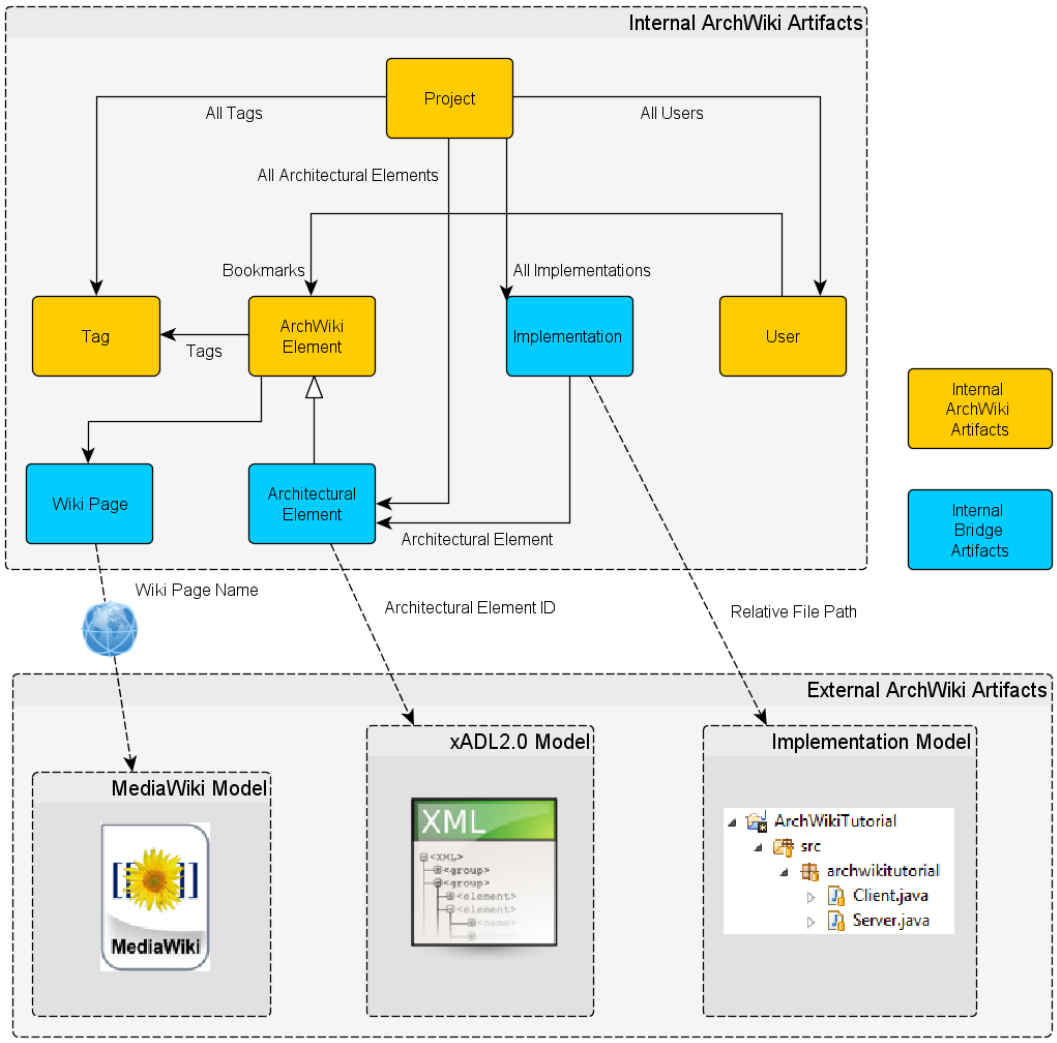

Figure 4.2: Internal and External ArchWiki artifacts are connected through bridge elements

MediaWiki

The MediaWiki server stores its wiki pages in a database. Access to the wiki is done through the MediaWiki API by making HTTP request to the web service. MediaWiki returns a response in an XML format. To access wiki pages users have to specify the URL of the MediaWiki server in the preference page. ArchWiki internal bridge WikiPage elements have an ID attribute that refers to wiki page names.

Source Code

ArchWiki will manage source code that is saved in the model directory. The ArchWiki internal bridge implementation elements have an attribute that refers to the relative path of the file.

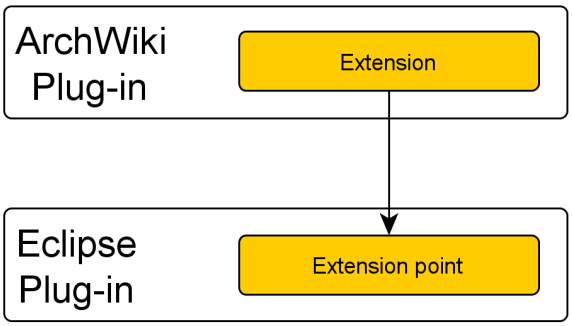

Figure 4.3: The ArchWiki plug-in uses extension points to extend functionality of current Eclipse plug-ins

4.2.3 Model Implementation Limitations

All the artifacts, with the exception of the wiki pages, are saved in files. Collaboration currently requires manually committing the ArchWiki files into an revision control system and other users updating your committed changes. Collaboration done in this way is not instant and requires a significant effort from the user. Our idea of collaboration envisions an application in which changes to the ArchWiki model are automatically and instantly propagated to other users and other users become automatically aware of these changes due to notifications. We did not change the ArchWiki tool to fully match our vision of collaboration, because the ArchWiki tool relies on third party tools and such an effort would require us to significantly change the persistence architecture of those third party tools.

4.3 The View

The View consists of components of ArchWiki that are visible to the user. These components communicate with the ArchWiki Controller by sending Command objects. In Section 4.4.2 this will be described in more detail. The components can be divided into two categories:

- Tool extensions are generally existing native editors of external ArchWiki artifacts that have been modified and integrated with ArchWiki.
- ArchWiki views are custom Eclipse Views that visualize the ArchWiki Model.

4.3.1 Tool Extensions

ArchWiki extends current development tools that manage the underlying ArchWiki artifacts in Eclipse to integrate with ArchWiki. The currently extended tools are: the web browser, the file navigator, the Java source code editor, and ArchStudio. Eclipse has an extension framework to provide support for extensibility, which is one of the core concepts of Eclipse [29]. The Eclipse tools are Eclipse plug-ins,

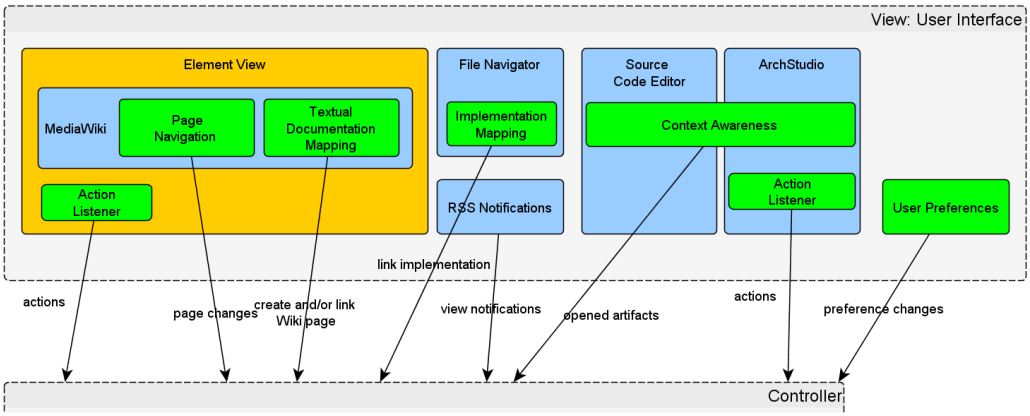

Figure 4.4: Diagram of interaction between Eclipse tools (shown in blue) and the ArchWiki Controller

which define extension points and extensions. Extension points allow other plug-ins to add functionality and extensions add contributions to existing extension points of other plug-ins (Figure 4.3). ArchWiki uses a number of extension points to create extensions and expand the functionality of current plug-ins and better integrate with ArchWiki. The features of the extensions can roughly be divided into three categories:

- Mapping extensions, which enable tools to map the artifacts they manage to the ArchWiki model.
- Action recording extensions, which enable tools to record actions performed in the tools and link it to the ArchWiki model.
- Miscellaneous extensions, such as extensions that send events when an artifact, such as a source code file, is opened.

Figure 4.4 shows the interactions of the extensions with the ArchWiki Controller. In the next subsections we will describe for each tool the extensions we have created.

File Navigator

The Eclipse file navigator has been extended to enable users to map the currently selected architectural element to a source code file. Figure 3.13 shows how the user maps the source code file by opening the context menu of that file and selecting the link option. This functionality is implemented as an org.eclipse.ui.menus Eclipse extension and creates a dynamic menu contribution, which will create additional elements in the context menu based on the file it was activated for and the current state of other integrated tools within ArchWiki.

The file navigator depends on Archipelago, because the context menu option only appears when an architectural element is selected in Archipelago.

Source Code Editor

The Eclipse java code editor has been extended to alert the ArchWiki Controller which source code file is currently being viewed. This is implemented by registering a listener with Eclipse to be notified when the Java code editor is activated. If the activated file is present in the ArchWiki model and mapped to an architectural element, the Controller will automatically show the Element View belonging to that piece of source code.

This also works in the other direction. From the Element View users can click on the source code and the Java code editor will open for that piece of source code.

Archipelago

ArchStudio is the tool suite for the xADL2.0 architectural description language. We have created custom extension points in the Archipelago source code to integrate ArchWiki with Archipelago. With these hooks ArchWiki has direct access to the underlying xADL2.0 model that Archipelago uses. This not only allows ArchWiki to read the contents of the model, but also make changes. The Archipelago structure editor is the main way to interact with the architecture description. ArchWiki installs listeners into Archipelago to capture actions performed within the editor. One of those actions is detecting when an architectural element is selected and showing the corresponding Element View. The Archipelago extension also records other actions a user performs. Examples of such actions are: adding a new component, connecting a component to another component, and removing a component.

Web Browser

The Eclipse Browser is used to access wiki pages. The ArchWiki extension to this Browser maps architectural elements to wiki pages. It is also possible to let the extension create a new wiki page and link it to the architectural element. The extension also provides a wiki page suggest feature, which suggests wiki pages when linking an architectural element to a wiki page.

ArchWiki embeds the Browser in the Element View. Clicks on hyperlinks within this Browser are monitored. If a user clicks on a hyperlink to a wiki page that is contained in the ArchWiki model, the Element View will be changed. A regular browser can also be used to access the wiki pages. The downside to this is that it does not come with all the integration benefits ArchWiki provides.

4.3.2 ArchWiki View

The ArchWiki View is implemented as an Eclipse View extension. The most prominent part of the ArchWiki View is the Element View (Figure 3.15), which is intended to visualize the architectural element as the central hub of the ArchWiki model. With the Element View we provide a mash-up of the related information about a particular architectural element in the model. The most prominent component in the Element View is the embedded wiki page, which serves as the textual documentation. Incoming and outgoing connections within the structure diagram are also shown, along with the tags.

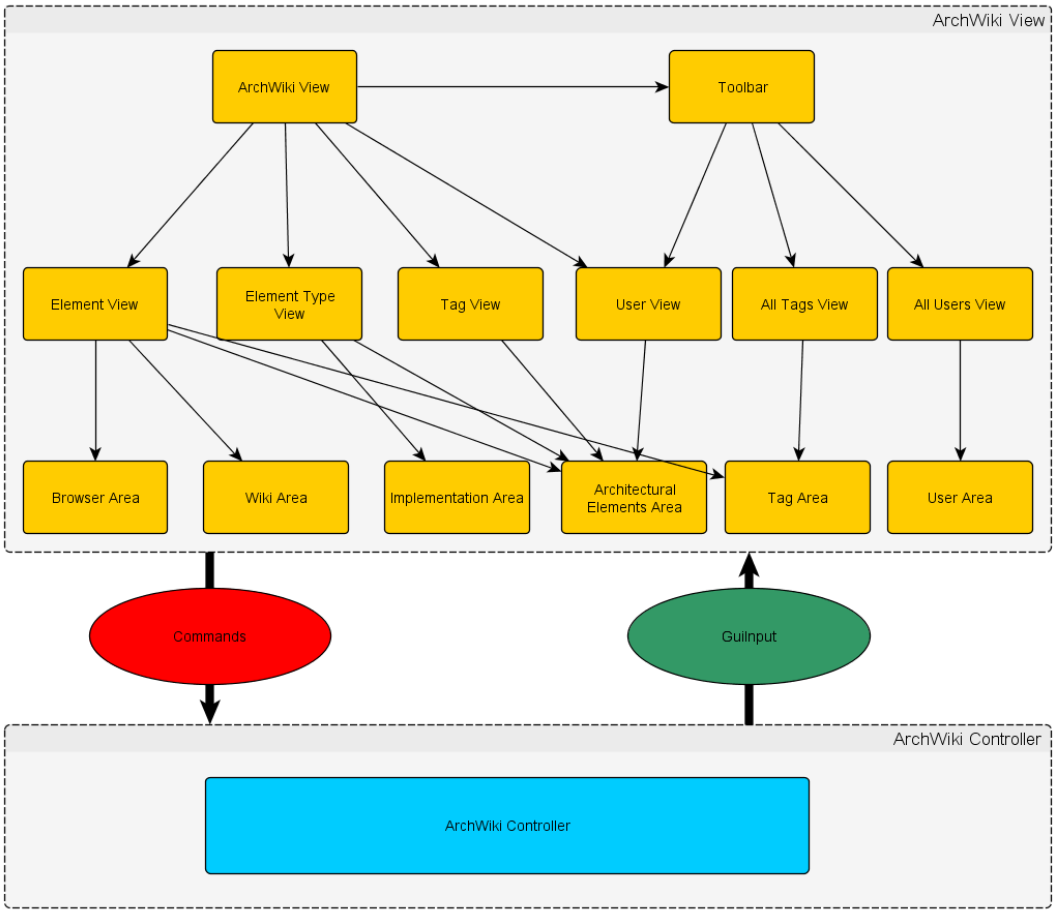

Figure 4.5: The structure of the ArchWiki View, consisting of different Views that are composed of Areas. The Views send Commands to the ArchWiki Controller and receive GuiInput object to visualize the ArchWiki Model

Because the Element View is the central hub it can be obtained in several ways. Element View instances can initially be reached through the usage of tools that are integrated with ArchWiki. For example, when a user edits a piece of source code or clicks on an architectural element in Archipelago the relevant documentation will be shown in the Element View. From there the user can browse the rest of the documentation from within the Element View.

We use the Standard Widget Toolkit (SWT), the widget toolkit Eclipse uses, to create the user interface[1]. Figure4.5 shows the structure of the ArchWiki View. The ArchWiki View consists of two elements:

- A toolbar with buttons that provide easy access to important views such as the Users View.
- A dynamic view of type ArchWikiViewComposite that shows the different types

[1] http://www.eclipse.org/swt/

of ArchWiki views, such as the Element View, the Tag and Tags Views, and the User and Users Views.

The different views are composed of different reusable components called Areas. Examples of such Areas are the Tag Area, the User Area an the Connection Area. The ArchWiki Controller changes the view by setting the input data using a GuiInput object. The Views do not contain computations. The purpose of the Views are to provide visualization to the user and to generate Commands based on the interaction by the user and the current input. Commands will be send to the ArchWiki Controller, which provides the logic to deal with the Commands.

4.4 The Controller

The responsibilities of the ArchWiki Controller can be divided into three parts:

- It interacts with the Model, which is the ArchWiki Model and the underlying artifacts of that model.
- It stores, and executes Commands originated from components in the View.
- It creates a feed based on the current stored and executed Commands.

These three functions will be further elaborated in the next subsections.

4.4.1 Interaction with the Model

The ArchWiki Controller is responsible for the interaction with the ArchWiki model and the underlying artifacts. In particular the architectural description, because the architectural elements are the core artifacts in ArchWiki. The ArchWiki Controller is created when an architecture description file is set in the ArchWiki's preference page. This usually done by choosing the "Initialize ArchWiki" item in the context menu of a xADL2.0 file. This will determine the location of the model files, set that location in the preferences, and create a new .archwiki file, which contains the newly created ArchWiki model. It is also possible to manually set the ArchWiki model files location in the preference page in case an ArchWiki model already exists. Wiki pages and Commands are other artifacts maintained by the Controller. Figure 4.6 shows an overview of the interactions between the Controller and the Model. The implementation details are described in the next subsections.

ArchWiki Model

The ArchWiki Model is not only defined with the Ecore diagram editor of EMF, but EMF also generates Java source code to provide low level access to the model. On top of this we created the ArchWikiDocument class, which provides resource management and offers higher level functionality.

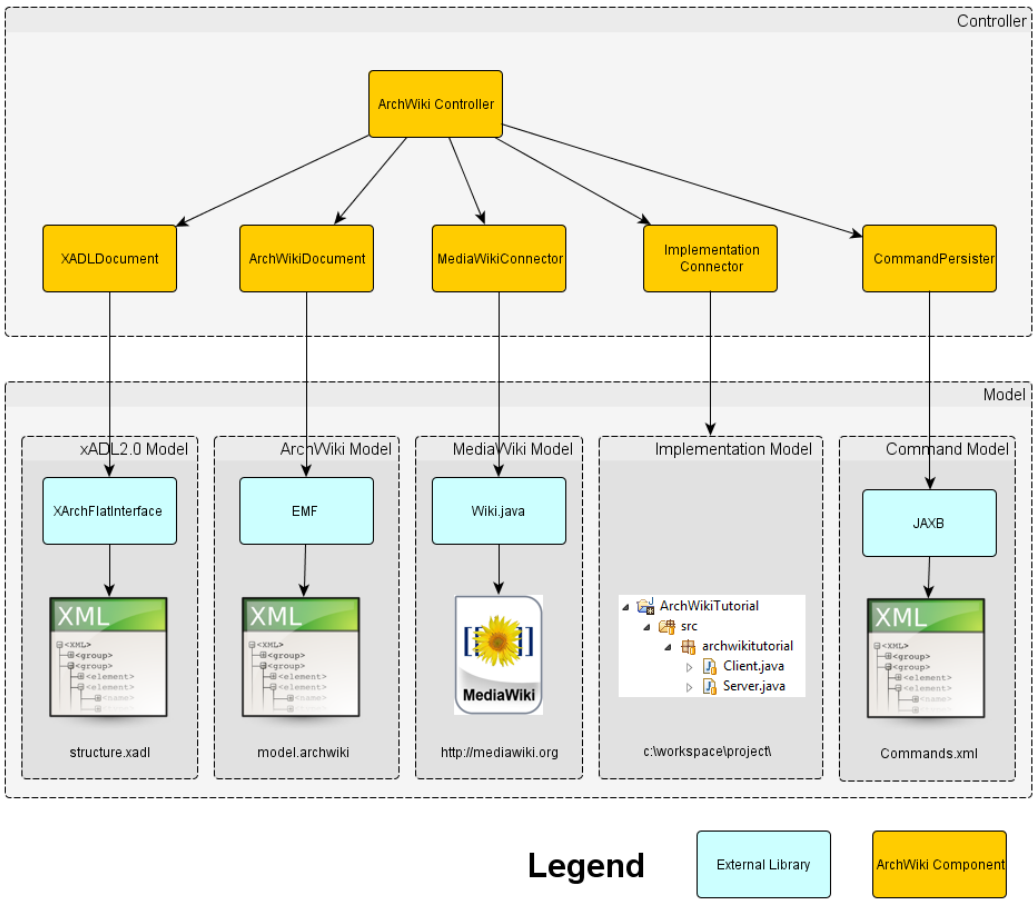

Figure 4.6: Structure of classes involved with interaction between the Controller and the Model

xADL2.0

The xADL2.0 document can be accessed by the XArchFlatInterface class provided by ArchStudio. ArchWiki adds the XadlDocument class to provide functionality specific for ArchWiki.

MediaWiki

ArchWiki uses the third party Wiki.java[2] library to access the MediaWiki server. This library is wrapped around by the MediaWikiConnector component to accommodate for use within ArchWiki .

The performance of the ArchWiki tool was generally very good as it was very responsive to user input. The connection to the MediaWiki server proved to be the bottleneck and was relatively slow with response times measuring in seconds. These response times caused the system to be too slow to work with. To resolve this issue we use multiple threads to retrieve data from the MediaWiki server. This solution reduced

[2]http://wiki-java.googlecode.com/svn/trunk/src/org/wikipedia/Wiki.java

the response times from 4 seconds to 1.2 seconds. However this was still not ideal and to further reduce the response times we created an alternative to the MediaWikiConnector class. This alternative is called the CachedMediaWikiConnector class and uses caching to speed up the retrieval times of MediaWiki data. Caching is a common strategy used to increase performance by lowering the amount communication and lowering the response times, but has two disadvantages:

- Overhead costs of cache misses
- Keeping the cache consistent

In the case of ArchWiki the caching strategy is solely used to lower the response times, because the amount of communication is not a big problem. Of the two disadvantages only the cache coherency problem has to be addressed, because the overhead costs of cache misses are negligible. To address the cache coherency problem we used a very simple scheme to keep the cache consistent: For every cache hit retrieve the remote value in a separate thread and inform the Command that needed the data if the data appears to be inconsistent. For ArchWiki this means in practice that the first retrieval of data from the remote MediaWiki server will be slow, while any subsequent retrievals will be fast, because the data is contained within the cache. In the case of an inconsistency in the cache the response time will still be fast, but with invalid data. However the data will be immediately retrieved and the Command that needed the data will be informed. For example, in the case of a user clicking on an element to view its Element View, the user will see the incorrect page for a split second afterward it will be updated with the correct data. Because the update time is so short most users will not even notice that they were presented with incorrect data, and therefore can not act upon that incorrect data.

This caching strategy requires that Commands that require cached operations to register these operations. In case these operations encounter invalid data they can notify the relevant Commands. Currently only read operations are cached. A problem is that write operations based on invalid data will still processed. However this is not a big problem, as it is very hard to perform write operations on incorrect data. In the case this does occur, it is easy to revert in MediaWiki to the previous version.

The caching mechanism is ArchWiki is designed to be as decoupled from the rest of the system as possible. The caching mechanism in ArchWiki consists of the following four components:

1. CommandCacheController: The CommandCacheController is responsible for registering UpdateCache operations. When all registered UpdateCacheThreads have reported back then the CommandCacheController will decide if the cache was invalid and if the CachedCommand must be alerted of this fact.
2. UpdateCacheThread: Operations that can be cached must define an UpdateCacheThread, which will update the cache and check if the data for that operation was invalid and report it to the CommandCacheController.
3. CachedMediaWikiConnector: The CacheMediaWikiConnector is a subclass of the MediaWikiConnector in which the operations have converted to cached operations. This means that each those operations will contain an UpdateCacheThread which will update the cache of the CachedMediaWikiConnector.

4. CachedCommand: CachedCommand is an abstract UserCommand class which provides functionality to obtain a CommandCacheController and signal the controller to stop registering for UpdateCacheThreads. It also provides a default callback function which will simply repeat the command in case of invalid cache.

To use the caching mechanism only the CachedMediaConnector has to be added to the controller instead of the MediaConnector and commands that require caching need to extend CachedCommand and optionally provide a callback function.

The new CachedMediaWikiConnector class provides response times of 1.2 seconds for first element view and 0.02 second for every subsequent view.

The use of both threading and caching decrease the response times helps with the usability aspect of the ArchWiki tool.

Commands

Commands are stored in an XML format. The Java Command objects are marshaled and unmarshaled into XML with Java Architecture for XML Binding (JAXB). The Java classes of the Commands are annotated to be used with JAXB. Upon activation the CommandPersister object loads the Commands objects in the "commands.xml" file into the collection class CommandStore or creates a new CommandStore object if the "commands.xml" file does not exist. The CommandPersister is responsible for marshaling and unmarshaling the Commands between the CommandStore and the "commands.xml" file.

4.4.2 Interaction with the View

The ArchWiki Controller interacts with the components in the view by receiving Commands and replying to view requests. These features will be discussed in the next subsections.

Commands

ArchWiki uses the Command design pattern [27] for communication between the components in the view and the ArchWiki Controller. In this design pattern the views encapsulate all the information needed to execute a Command in a CommandParameter object. We have chosen this approach for the following reasons:

- Using a Command object makes it easy to maintain a list of previous commands. Storing this kind of meta data enables recommendation and awareness features.
- Sending Command objects to the ArchWiki Controller instead of direct method calls means less coupling between the Controller and the View. It also allows use to create new types of Commands without changing the interface of the ArchWiki Controller.
- The Command pattern allows for more concurrency and can cope better in situations where a Controller communicates with multiple view components from multiple users simultaneously. The Commands can be queued and executed one after another instead of concurrent, which can cause concurrency issues.

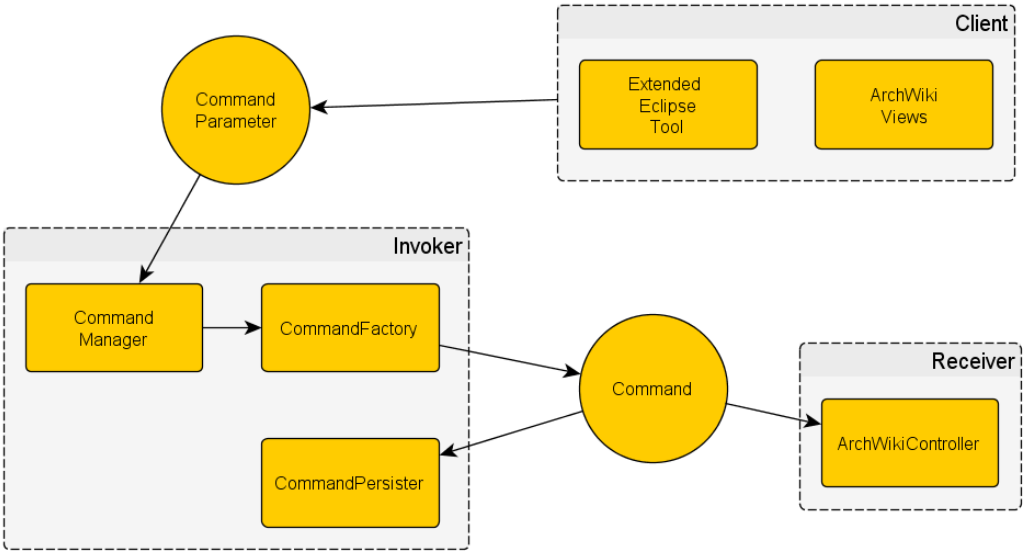

Figure 4.7: ArchWiki uses the Command pattern for communication between the View and Controller. The arrows indicate where the CommandParameter and Command objects are send.

There are three terms always associated with the command pattern:

- The client instantiates the command object.
- The invoker decides when the method should be called.
- The receiver is an instance of the class that contains the methods code.

Figure 4.7 shows how the Command pattern is applied to ArchWiki.

Client In the case of ArchWiki the clients are the components within the View: mostly the extended Eclipse tools and the Element View. The clients can create CommandParameter objects, which contain the information needed to execute a Command. ArchWiki has chosen to let clients only create a CommandParameter object instead of the actual command, because we do not want the client to define the implementation of the Command. We decided to let View to be as thin as possible, so we let it only provide the minimum information needed for the ArchWiki Controller to execute the action, which are the parameters.

Any component can act as a client and create a CommandParameter and send it to the Controller. Examples of commands are the `Edit Wiki Page` Command, the `Architectural Element Selected` Command, and the `Tag Added` Command produced by the Element View. Archipelago creates commands such as `Component Connected` Command and `Name Changed` Command. However, the Archipelago commands are already executed by Archipelago self. The Commands that Archipelago sends function more as notifications.

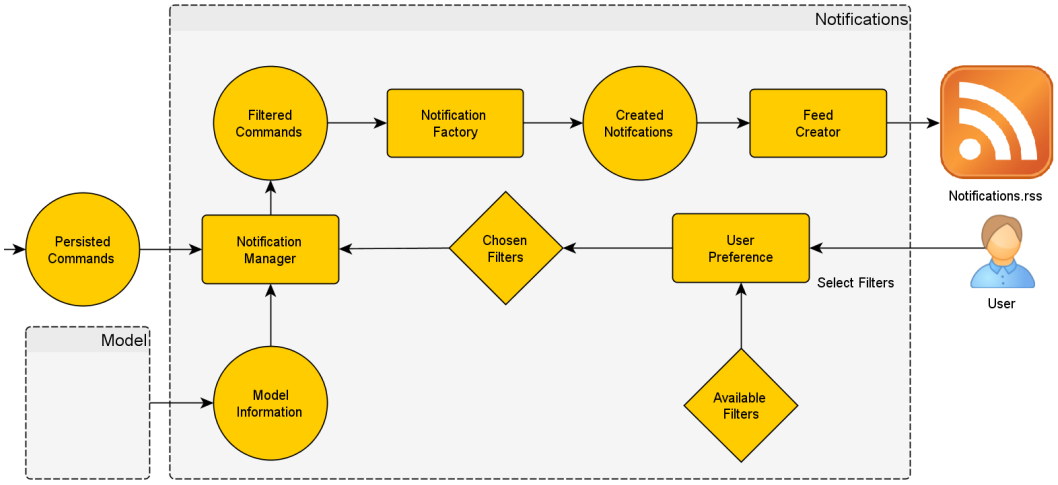

Figure 4.8: Data flow diagram of persisted Commands that are tunred into Notifications using Filters specified by the User, thus creating personalized RSS feeds.

Invoker The CommandParameters are send to the ArchWiki Controller, which will delegate them to the CommandManager. The CommandManager use the CommandFactory to create an appropriate Command and then executes it. The Command is then persisted with the CommandPersister if it is a persistable Command.

Receiver The ArchWiki Controller maintains the logic of the Command. A Command can perform up to three different functions:

- Perform verification of the validity of the Command and add meta data to the Command. Verification can be made to ensure that changes to the artifacts are possible within the ArchWiki model. Meta data such as the time of execution and the executing user can be recorded to connect the Command to the ArchWiki model. Also information need for undoing operations can be added.
- Make changes to the ArchWiki model and its artifacts. Examples of these are Commands that map implementation or add textual documentation.
- Request for information in the form of input to a View. Some Commands are request for input to a View. For example, selecting an architectural element in Archipelago or opening a piece of source code, will result in the ArchWiki Controller showing the Element View by sending the GuiInput object to the Element View.

4.4.3 Publishing Notifications

Figure 4.8 shows the data flow diagram for the Notifications. ArchWiki notifications are created from persistable Commands in the CommandStore. When the CommandStore is changed the NotificationManager first filters the Commands based on Filters specified by the User. Because the Commands are part of the ArchWiki model the filters select Commands based on the model. For example, Users can filter Commands

based on the User that executed the Command, or filter Commands involving elements based on how close they are in the ArchWiki model. Because of this filtering mechanism ArchWiki offers a personalized notification mechanism. After the Filters are applied the NotificationManager will turn Commands into Notifications by using the NotificationFactory. It then can create a personalized Atom/RSS feed of Notifications.

Chapter 5

Evaluation

5.1 Goal

The primary goal of the evaluation is to assess whether the ArchWiki tool is able to support software architecture knowledge management, and how it supports the architecture centric approach. The evaluation method is based on feedback from users of ArchWiki. The method consists of two parts.

The first part interviews participants about their views on the current state of software architecture and architecture knowledge management. In particular their own experiences and problems with software architecture are of great interest. This is to see if the participants' problems match with the problems that ArchWiki tries to address.

ArchWiki's approach is to support architecture knowledge management by using Web 2.0 concepts. This approach is chosen, because the identified problems with current architecture knowledge management practice appear to match with solutions Web 2.0 concepts can provide.

We divide the research question into three sub research questions and focus our evaluation study to answer each of these:

RQ1 Can ArchWiki support architectural knowledge retrieval?

RQ2 CanArchWiki support architectural knowledge documentation?

RQ3 Can ArchWiki support collaboration in architecture knowledge management?

The second part of the evaluation consists of presenting the tool and all its features to participants and gauge how the ArchWiki tool fulfills its goals. This is achieved by asking the participants their thoughts on the current state of the tool and how they feel to what degree ArchWiki fulfills its goals.

5.2 Evaluation method

The ideal method of evaluation would consist of evaluating the ArchWiki tool in a real life setting. However, it is difficult to find such a real life setting for a number of reasons. First, this tool is intended to support the architecture centric view and

this approach is not a widely used approach. Second, the ArchWiki tool is primarily a Web 2.0 tool and those tools in general have the property that the functionality is determined by the users, their work flow, and underlying data. An ideal evaluation study would take place in a setting where a lot of users have settled in working with the tool and have build up an extensive code base and accompanying documentation. This, however, would require a large study.

A large study would require observing the users using the ArchWiki tool in an actual large project, with a large code base, during a long time. Such a study would probably be resulting in more interesting results, because software architecture management is more of an issue in larger projects, because obviously larger code bases tend to also have more complex architectures. The longer time would enable the users to incorporate the ArchWiki tool into their work flow and to build up the artifacts with ArchWiki. Also during such a large evaluation study, users would leave the company and new people would join it. This would provide additional interesting data about new developers learning a code base with ArchWiki (as opposed to without or other means). Also interesting observations could potentially come from ArchWiki documentation by users that are no longer with the company and compare that to the situation if those ex-users would not have documented it or documented it with a different method.

A larger study would also face some difficulties. The users should have a positive attitude towards the architecture centric method and the ArchWiki tool. We feel that forcing users to adopt this approach and the ArchWiki tool would yield very different results.

Due to time and resource constraints we did not chose to do a larger study. Also the tool is not ready to be used in a real production setting, so doing a larger study would require us to invest an additional amount of time and resources to make the tool production ready.

We chose to do a small preliminary study to evaluate the ArchWiki tool. In this study we opted just to acquaint the participants with the ArchWiki tool and its features. We did not asked the participants to do certain tasks and assess those tasks, because this would be in a setting unsuitable for ArchWiki: a single user, short duration, small code base setting. Instead, by performing the tutorial and getting to know its features, users could give feedback based on their own ideas and imagination how ArchWiki would work.

5.3 Participants

Ideally the participants of the evaluation study are software developers and software architects that already use architecture knowledge management tools, such as ArchStudio and UML tools, to document their software architecture and use the architecture centric view. Also experience with the Web 2.0 technologies will allow participants to more effectively use the Web 2.0 technologies incorporated in ArchWiki. These participants will be able to share their opinions about how well ArchWiki supports the architectural tasks they currently perform without ArchWiki.

In practice, performing architectural tasks, such as documenting the architecture are not done very often and usually only in fixed stages of a project and not continu-

ously. Participants that fit the profile are hard to find. However, participants that do not document architecture are also valid participants, because they can provide feedback if and how ArchWiki can overcome the reasons they have for not performing architectural tasks.

5.4 Evaluation Setup

The evaluation was done in a group setting and started with an introduction to software architecture and that we created a tool called ArchWiki to support software architecture. The participants were then introduced to ArchStudio and ArchWiki. After the introduction the participants performed a tutorial in which they build a small mock client-server project with ArchWiki. After the tutorial the participants filled in a questionnaire consisting of 2 parts: the first part questioned them about their current views and practices on software architecture and the second part questioned them about their opinion about ArchWiki. After all participants had filled in the questionnaire we held a group discussion. In the next subsections we will describe the individual steps of the evaluation study in detail.

5.4.1 ArchStudio and ArchWiki Introduction

The evaluation study started by giving the participants an introduction about software architecture and its current state. We then introduced them to the ArchStudio tool, which is used to create software architecture artifacts such as structural diagrams based on the Xadl2.0 architecture description language (ADL). After that ArchWiki was presented as an extension for ArchStudio, which uses Web 2.0 features to support software architecture knowledge management. They were told that they would be performing a tutorial to present them to the features of ArchWiki and would fill in a questionnaire after which would follow a discussion with the whole group. After the introduction the participants proceeded to perform the tutorial individually.

5.4.2 ArchWiki Tutorial

The participants created a simple project using ArchWiki to demonstrate its features. The project started as a normal ArchStudio project in which the architectural structure was created along with some implementation artifacts that corresponded to components in the structure. The user then initialized the project for use with ArchWiki.

The first feature presented was the ArchWiki view, which is the main GUI element of ArchWiki. This view showed the element view when an architectural element was selected in the Archipelago editor of ArchStudio. This element view showed the architectural element as the main artifact in the model to support the architecture centric approach. This element view not only provides the structural information about the architectural element, such as its name and connections, but also the implementation mappings, wiki pages and tags.

The participants were shown how to navigate the artifacts of the model. ArchWiki aims to provide optimal navigability by accessing the model through the underlying software artifacts and navigating to other model elements by showing hyperlinks to other model elements in the element view. Before this navigability is possible, the

different artifacts first have to be connected in the model. ArchWiki aims to simplify this task by integrating these operations into the tools of the respective artifacts and using the context to determine the connecting element. For example, to connect a Java class to an architectural element, the user only had to open the element view of the corresponding architectural element and open the context menu of the class in the package explorer. ArchWiki then had an option to link that class to the architectural element in the model. After building up the model, ArchWiki enables the user a high degree of navigability with the architectural element as the central artifact.

The participant gains entry to the model through the respective editors of either the architectural elements and the source code: the structural editor Archipelago and text editors. When an architectural element is selected in Archipelago its element view is shown in the ArchWiki view. Also when a piece of source code is opened, the user will be shown the corresponding architectural element. This feature, which shows architectural documentation of source code automatically, aims to address the "findability" and traceability issues of architectural documentation.

Upon entry of the model the participant had access other model elements through the element view. It's possible to navigate to other architectural elements, but it is also possible to navigate to other types model elements such as source code by selecting it, which will open the source code in the text editor. Next to a high degree of navigability and usability ArchWiki also embraces the Web 2.0 way with some Web 2.0 technologies which where shown in the tutorial.

For example, the tagging of architectural elements is supported in the elements view. With tags participants categorized elements in an informal and a bottom up way by adding key word meta data. Tags should help with the comprehension of an architectural element by making important concepts associated with the element concrete. An autocompletion feature presented the user with already used tags when adding a new tag. This feature serves two purposes: to increase the usability of the tagging feature and to decrease the number of similar tags.

Another Web 2.0 technology used in ArchWiki is the use of wiki pages. Users created a wiki page for architectural elements in which they contributed unstructured textual architectural information. Following the Web 2.0 way, they connected this wiki page to the architectural element within the ArchWiki model. The wiki page of an architectural element was then embedded in the element view of the corresponding architectural element. ArchWiki provides an easy way to create, link, and show a wiki page for an architectural element by adding a "create wiki page" option in the context menu of an architectural element in Archipelago. Selecting this option created a new wiki page with the same title as the architectural element's name and linked it automatically to the corresponding architectural element in the ArchWiki model. ArchWiki not only provides a connection between an architectural element and a wiki page, but it can also infer connections to other architectural elements from the contents of a wiki page. The contents of a wiki page often refers to other wiki pages in the form of hyperlinks. When such a hyperlink is selected from the Wiki view it is analyzed by ArchWiki to find corresponding wiki pages in the ArchWiki model. If such a wiki page is found not only the wiki page will be changed, but the entire element view will be changed to show full information on the architectural element corresponding to the wiki page. To make it easier to establish links between wiki pages (and their corresponding architectural elements) it is possible to add links to the wiki page of an

architectural element by selecting "add link" from the context menu of the respective architectural element from Archipelago.

Because the ArchWiki model can become rather large and unmanageable a bookmarking feature is present in ArchWiki. Participants used this feature to bookmark architectural elements to make them more accessible. Bookmarked elements are shown in the user view to provide easy access.

Awareness is an important aspect of software development in teams and ArchWiki supports this by generating an RSS feed which shows the activity of users in ArchWiki. It does so by recording certain user interactions within ArchWiki. These interactions get filtered and translated to an RSS feed based on the user's preferences. Examples of filters are: show only actions of other users and show only actions on bookmarked elements.

5.4.3 Questionnaires

After the tutorial the participants were asked to fill in a questionnaire consisting of two parts. The questionnaire consisted of 52 questions. The questions were a mix of rating questions, using a five point Likert scale[1], yes or no questions, and open questions. A large number of questions also included, next to a section to fill in their own answer, a list of potential answers, of which the participants were asked to indicate to what degree the answers applied to them. The questionnaire we used is included in the appendix.

Software Architecture Views-and-Practices Questionnaire

The first part of the questionnaire covered the background, views and practices of the participant. It started with questions (2-6) relate to their background in software development and software architecture. The next questions focused on their practices and views on the different aspects of software architecture. Questions 7 through 11 were about software architecture consumption and program comprehension. Documenting the software architecture was covered in questions 12 to 17. Their views on collaborating on software architecture were handled in questions 18 to 21.

ArchWiki Questionnaire

After the participants had performed the tutorial they were able to get an idea of ArchWiki and its features. The participants filled in the second part of the questionnaire in which they gave feedback on ArchWiki and how they feel to what degree ArchWiki was able to reach its goals. Questions 22 to 24 first questioned the participants about their experiences with Web 2.0 and how they think it could help with software architecture management. In questions 25 to 34 the participants were asked about their general impressions about ArchWiki. The next questions were geared towards individual features. Questions 35 to 39 covered the implementation mappings of ArchWiki, questions 40 to 42 handled the use of a Wiki as documentation, questions 43 to 44 handled the tags, questions 45 to 46 were about bookmarks, and questions 47 to 52 handled the notifications.

[1] http://en.wikipedia.org/wiki/Likert_scale

5.4.4 Group Discussion

After all participants had filled in the questionnaires we had a group discussions about the topics covered in the questionnaires and their answers to the questionnaires. This discussion was used to gain more insight in the answers given by the participants. Answers that were unclear or not expected were potential subjects of discussion.

5.5 Pilot Study

Before we performed the evaluation study we first performed a pilot study with a researcher from our research group. We used this study to assess the duration and structure of the actual study and the strength and comprehensibility of the questions. There was a difference between the pilot study and the actual study, because the pilot study was performed on a one-on-one basis and not in a group setting like the actual evaluation. The feedback received during the pilot included bugs encountered during the tutorial and suggestions for certain questions. Overall, the participant agreed with the set of questions and found the tutorial easy to perform and adequate in its ability to demonstrate ArchWiki's features. The participant did notice that the tool performed sluggish in certain instances. We believed this was partly due to the use of a remote web server. For the actual evaluation study we used a web server hosted on the university's computer, and added concurrency and caching to speed up the ArchWiki tool.

5.6 Evaluation study

For the evaluation study we invited 8 participants to partake in 3 group sessions. The participants were master's or PhD students, but most were also active in the industry. We performed the study on 3 different days with respectively 2, 2, and 4 participants.

5.7 Results

In this section we will present the results from the evaluation study. The results come from both the questionnaires and group discussions, which followed the questionnaires. The results are accompanied by box plots that graphically depict the data retrieved from the Likert scale questions, which range from 1 (strongly disagree) to 5 (strongly agree).

5.7.1 Background Information

The participants were developers with experience ranging from 1 to 13 years. Also most of them worked within a team, but were not in a designated software architect position. The architects in their projects determined the architecture by choosing the architectural patterns and frameworks to be used and created high level designs resulting in artifacts such as a UML-diagrams.

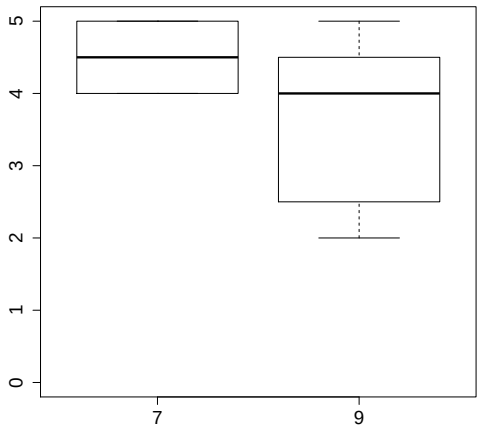

Figure 5.1: Program comprehension and software architecture documents

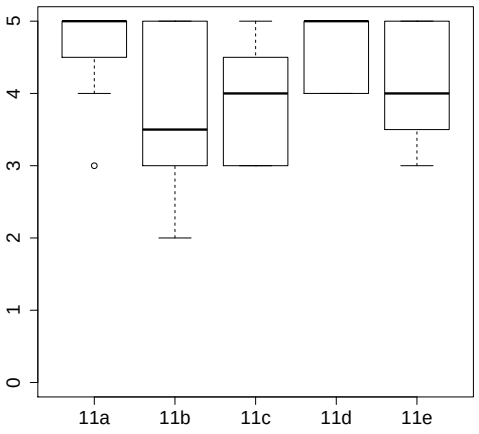

Figure 5.2: Problematic items with consumption software architecture

5.7.2 Software Architecture and Program Comprehension

The participants all agreed that program comprehension is a time consuming task as can be seen from Figure 5.1 Question 7. For this process they used both, top down and bottom up analysis, and often a combination of both. Software architecture documents are regarded as useful and are used when available (Figure 5.1 Question 9). They found the following problems software architecture documentation in order of importance (Figure 5.2):

1. No confidence that the architecture documents are up to date and in sync with other development artifacts such as source code (Question 11d).

2. Locating the software architecture documents and relevant information within the documents (Question 11a).

3. Low participation in software architecture documentation by other team members (Question 11e).

4. Lack of traceability between the software architecture documents and other software development artifacts such as source code and requirements documents (Question 11c).

5. The use of different tools to process the architecture documents, source code and other documents (Question 11b).

Other problems mentioned were the lack of documentation of the design decisions, lack of test cases that describe certain functionality and the incompleteness of the documentation.

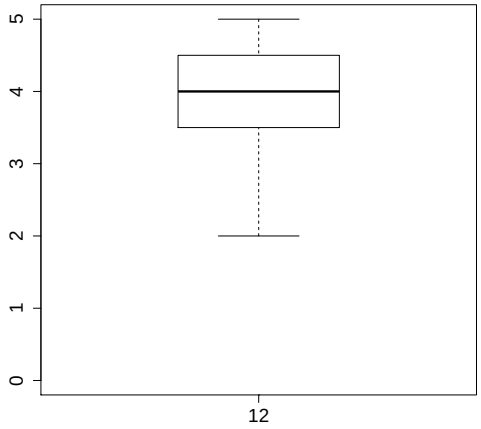

Figure 5.3: Importance of documenting the software architecture

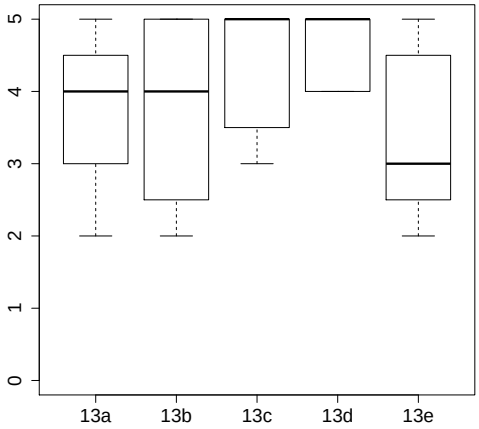

Figure 5.4: Reasons for documenting the software architecture

5.7.3 Documenting the Software Architecture

As can be seen from Figure 5.3 it was found important to document the architecture once it is in your head.

The participants found it was important to document the software architecture for the following reasons in order of importance (Figure 5.4):

1. To help project team members understand the system faster and to provide common concepts to assure everyone has the same mental picture of the software architecture (Question 13d).

2. To document the design decisions and rationale (Question 13c)

3. To provide a high level abstract view of the system (as in (UML) diagrams) (Question 13b).

4. To acts as a personal reminder of how the system is constructed (Question 13a).

There was no consensus on the importance of the documentation to act as blueprint and help achieve that the implementation stays as true as possible to the intended architecture (Question 13e): 3 Participants found it important, 3 participants were neutral, and 2 participants found it unimportant. Other mentioned reasons were:

- To help with designing new features.
- Because the documents are easier to consume than code.
- Evaluation of the architecture.
- Talking to stakeholders.

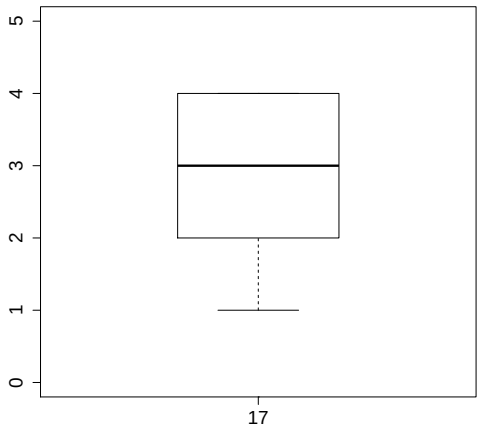

Figure 5.5: Satisfaction with current means of documenting

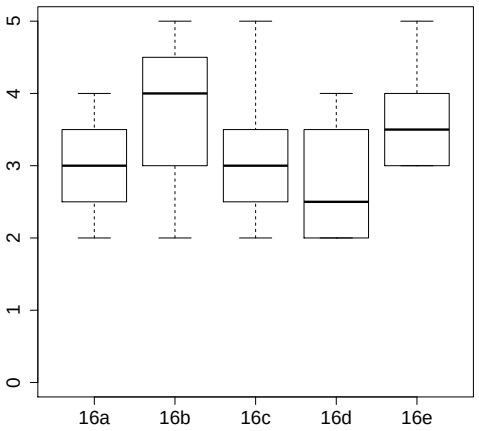

Figure 5.6: Problematic items with documenting the software architecture

They document the architecture in various ways. These range from low level documentation such as comments in the code and Javadoc to intermediate documentation such as test cases to higher level documentation, such as diagrams. The diagrams were often created in an informal way such as drawing on a whiteboard during a meeting. To capture these diagrams they simply made a picture of it with a camera. They also created documents with textual documentation, which was usually done in a text editor such as Word , but also wiki's were used. Opinions about the current means of documenting vary: Some were satisfied with the current means of documenting and others were not (Figure 5.5).

Documenting was done both in the beginning as in the later stages of development. Some created high level documents in the beginning of development when they were designing the architecture. Low level documentation such comments and Javadoc are typically done during coding phases. Also in the later stages high level architecture documents are also created/modified to document the architecture.

There are two main problems with documenting the architecture (Figure 5.6):

- It takes too much time (Question 16b).
- The current tools are inadequate (Question 16e).

Other mentioned reasons were:

- It will most probably get out of date.
- Small contribution require disproportionate effort (Question 16c).
- Just forgetting to do (Question 16a).
- Partial contributions are not possible or undesirable (Question 16d).

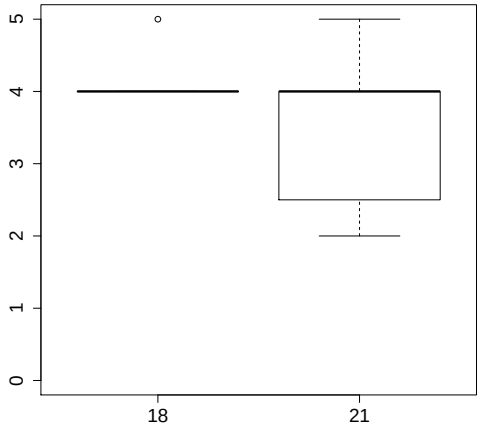

Figure 5.7: Collaboration

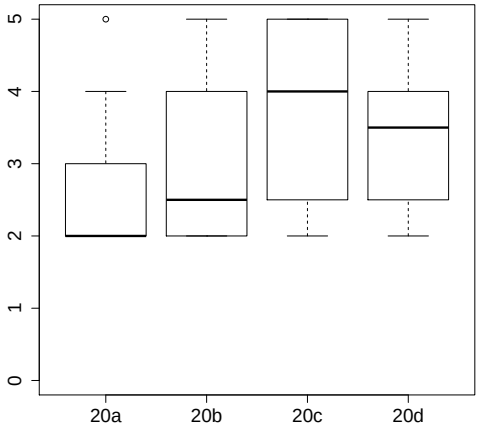

Figure 5.8: Problematic items with collaborating on software architecture

5.7.4 Collaborating on Software Architecture

All participants agreed that collaborating on software architecture with other team members can help with the problems related with the use of software architecture as can be seen from Figure 5.7 Question 18. Most participants collaborate using face-to-face communication such as physical meetings or coding together, but also Skype calls. One participant primarily participated in open source projects where face-to-face communication is more uncommon and collaborated through a wiki and a bug tracking system. He also used mailing lists sometimes. Another participant also used a wiki (Confluence[2]) in his company to collaborate.

The following problems were identified with collaborating on software architecture (Figure 5.8):

1. Team members are not aware when changes are made to the architecture documents and when the changes are of interest to them (Question 20c).

2. The participation of other team members with collaboration on software architecture is low (Question 20d).

Other mentioned problems were:

- No specific tools for collaborating on software architecture documents (Question 20b).

- Sharing of the physical architecture with other team members is difficult (Question 20a).

- No standard tool set for a project.

[2]http://www.atlassian.com/software/confluence/overview

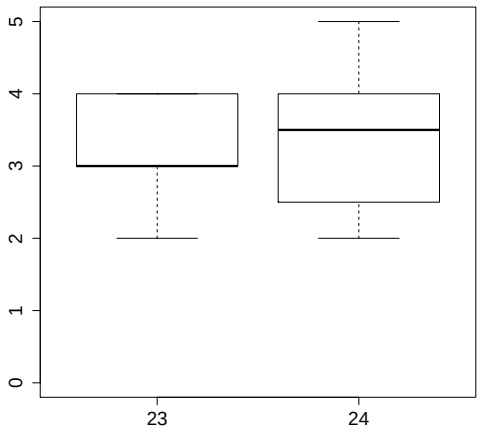

Figure 5.9: Web 2.0 and software architecture

- No standard location for getting and putting documents.

In Question 21 (Figure 5.7) most participants answered that they were more inclined to produce software architecture documentation if team members were made aware of the contributions.

5.7.5 Web 2.0 and Social Media Use

All participants used Web 2.0 and social media applications such as Wikipedia, Twitter, Facebook and Youtube. There was no real consensus on the opinion if Web 2.0 technologies could help in overcoming the problem related with the use of software architecture. In Question 23 on Figure 5.9 you can see that 3 participants thought it could and 1 participant did not think it could help. 4 Participants thought Web 2.0 could help to lower the threshold to document the architecture as can be seen in Question 24.

5.7.6 In-depth ArchWiki Features

Implementation Mapping

Figure 5.10 shows data regarding the implementation mapping. The participants agreed that the implementation mapping addresses the "traceability" issues of architectural documentation (Question 35) and that the mapping and the context view lowers the threshold to document the architecture when making changes to the source code (Question 36). Most of the participants agreed this mapping helps keep the software architecture and source code synchronized and thus help prevent architectural degradation (Question 37). Most participants also think this feature is useful and helps with program comprehension and documenting the architecture (Question 38).

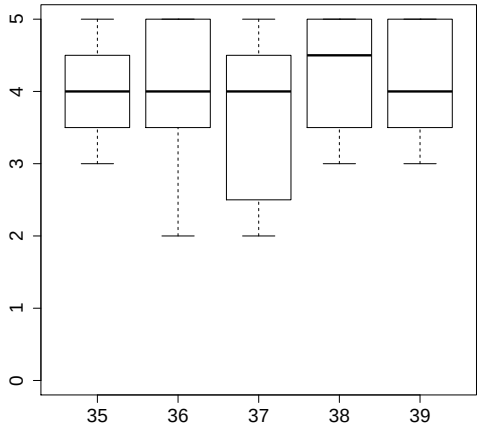

Figure 5.10: Implementation mapping

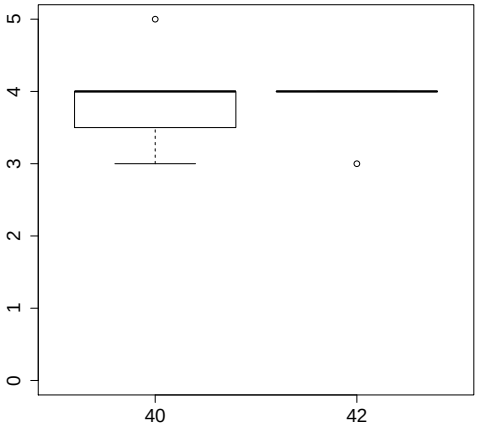

Figure 5.11: Wiki

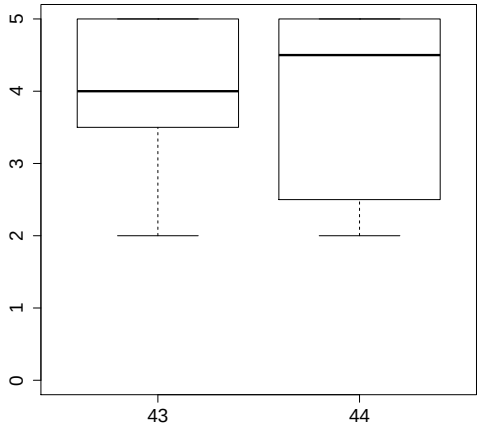

Figure 5.12: Tags

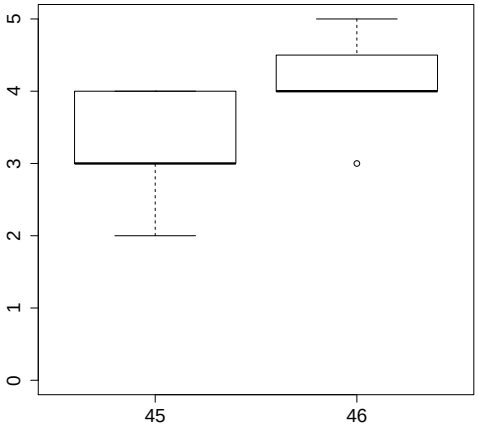

Figure 5.13: Bookmarks

Wiki

The participants found using a Wiki a good way for documenting the architecture as can be seen in Question 40 in Figure 5.11. Some users would also like to see wiki pages for other elements such as: tags, links, types, higher level components (library/jar, package, service), view points. In Question 42 the participants agreed that the use of a wiki lowers the threshold to document the architecture.

Tags

In Question 43 (Figure 5.12) most participants agree that tags created by themselves or others can help with program comprehension and also in Question 44 that tagging

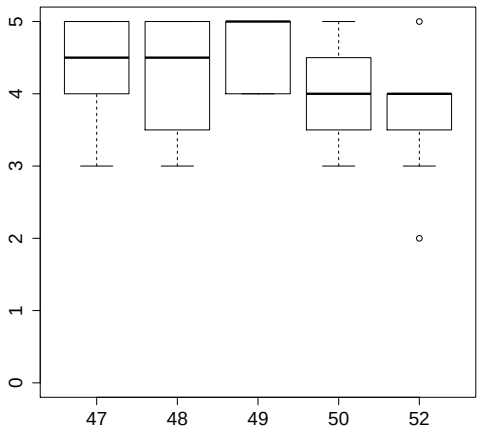

Figure 5.14: Notifications

is such a lightweight tool that it lowers the threshold to document the architecture.

Bookmarks

Figure 5.13 shows the data regarding the bookmarks. Some participants state in Question 45 that they think that bookmarks can help with managing the complexity of large architectural models. In Question 46 most state that they also like the feature to filter notifications based bookmarked elements.

Notifications

Data regarding the notification feature is shown in Figure 5.14. The participants answered in Question 47 that they liked the idea of being notified of actions of other team members and in Question 48 all liked the idea that other users would be made aware of their actions. Most stated in Question 52 that they liked the use of an RSS feed to show the notifications. They all stated in Question 49 they would like to be able to filter the notifications and in Question 50 most liked the currently available filters. Other desired filters were:

- Filter by tag.
- Filter by user or user group.
- Filter by project, package, type hierarchy.
- Custom filter.
- Filter on changes in documentation linked to files that the user recently worked on.

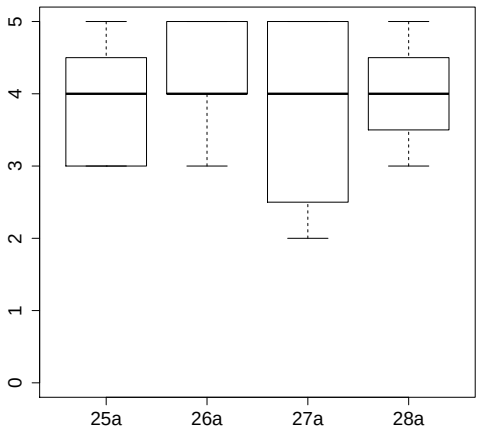

Figure 5.15: ArchWiki goals

5.7.7 ArchWiki Goals

Figure 5.15 shows the data about the participants' general impressions. Most participants found that ArchWiki helps with program comprehension as can be seen from the data from Question 25a. They mentioned the following features which helped them:

- Links between the source code, architectural components and documentation
- Integration and navigability between the different artifacts in the IDE
- Using the Wiki for documentation
- Modeling the structural architecture using xadl2.0 with Archipelago
- Tagging
- Notifications
- Showing incoming/outgoing connections
- Having a shared/centralized repository of artifacts
- The whole combination of things

In Question 26a they also agreed that ArchWiki helps with documenting the architecture. They mentioned the following features which helped them with that:

- IDE integration
- Linking the source code, architectural components and documentation
- Navigation between artifacts
- Making it a team effort (collaboration)

- Use of a wiki to document the architecture and the ease of adding rich meta data to components with it
- Tagging and autocompletion
- Modeling the structural architecture using xadl2.0 with Archipelago

In Question 27a most participants thought ArchWiki helps in keeping the architectural documentation up to date and in sync with other development artifacts such as source code.

The features they think help with this are:

- Linking the source code, architectural components and documentation and having the code and documentation alongside each other
- Documenting the architecture is now part of the work flow and integrated in the IDE
- More people collaborating on the documents and showing that other developer are also maintaining the documents
- Notifications and one can filter notifications and pay more attention to filtered elements
- The ease of updating the wiki

Some participants noted that there were no direct features to keep the architectural documentation up to date and in sync.

In Question 28a they agreed that ArchWiki supports collaboration on software architecture. The features that promote collaboration on software architecture are:

- Notifications; , the "continuous integration" of changes by others it allows and the social pressure it gives
- Ease of use
- IDE Integration
- Making documenting the software architecture a social experience and having access to other users
- Using a wiki and the awareness it raises with other developers and its informal nature
- Tagging

One participant also wanted to see why changes were made in the notifications.

5.7.8 ArchWiki General Impressions

The participants generally had a positive impression of ArchWiki or saw potential but think it required more features. They liked the concept of integrating documentation, architectural diagrams and code. Also integrating it in the IDE and centralizing the artifacts is well liked.

They also favored the usability apart from some minor issues. One participant noted: "In general it was very easy and intuitive to use". Some noted that the wiki screen was small, due to lack of screen space. Another participant mentioned that creating a new design would probably take some time, but maintaining it would be probably easier.

Most liked about the tool was:

- The linking and integrating of the various artifacts into a single tool
- The use of a wiki itself
- The use of a standard way of documenting with a centralized document location
- It lowers the barrier to contribute/collaborate on documentation
- Notifications with RSS output
- Tagging
- Visualization

There was also some aspects of ArchWiki that were not liked by the participants:

- Forced to be using ArchStudio and Eclipse
- Not being able to document the type.
- Retrieving wiki pages from the server was somewhat slow, compared to other interface interactions.
- Notifications do not give reasons for changes.
- Use of a simple browser to show the wiki.
- Documentation is not stored on the RCS.
- The fact that developers were now easily be able to edit the architecture, because software architectural tasks should only be done by a designated software architect.

Participants also mentioned several features they would like to see included into ArchWiki:

- Improved visualization
- An automatic mapping between code and design

- A karma feature, which awards points for adding pages, tags, etc to keep users involved
- The ability to bundle the wiki as part of a distribution
- Wiki pages for tags
- Wiki pages for higher level parts of the system
- Store the tags in the wiki
- Availability of templates for wiki pages
- Ability to comment on types
- Link and integrate mailing lists
- Mandatory comments for changes

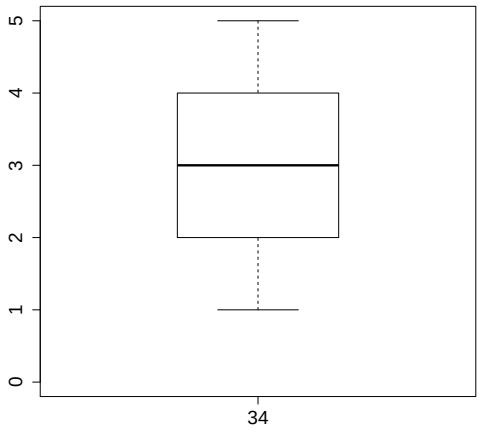

Figure 5.16: ArchWiki general impressions

In Figure 5.16 you can see that some some users would use it right now, another was held back by its current state as a prototype, and some preferred their current work flow for small projects with regular face-to-face meetings with whiteboard drawings. They, however, stated that ArchWiki might be an option in other circumstances such as increasing project size.

5.8 Threats to Validity

5.8.1 Threats to Internal Validity

There are a number of threats, which could influence the reliability of the data we gathered.

Implementation

The evaluation study only covered our current tool implementation to support our approach for supporting software architecture management with Web 2.0 technologies. Ideally we would have implemented the tool more closer to our vision. Examples of unimplemented features and current limitations in the implementation can be found in respectively Section 4.2.3 and Section 7.4.1.

The tool is currently implemented as an extension to ArchStudio and Eclipse. Opinions about the tool are therefore based on this integration and also on their experiences ArchStudio and Eclipse. Using ArchWiki with a different IDE and a different modeling architecture modeling tool might yield a different user experience

Also some parts could have been differently. We for example made the architectural element the centralized artifact in our model. We could also connected wiki pages to other artifacts, such as tags.

It is hard to determine whether the data we gathered stems from the approach or the current tool implementation. The data we gathered may therefore apply to the evaluated implementation of the tool and not to the ideal approach we envisioned.

Evaluation Study Setup

The way the evaluation study was set up also poses some threats to the validity:

Baseline No specific baseline to which the ArchWiki tool is compared to. The results from the participants are probably affected by what they are currently doing and this hidden personal baseline is probably different for every participant. Finding a proper baseline tool would be hard since ArchWiki would have a data advantage since it reliant on its own (meta) data. Because there is a different baseline for every participant, we included many questions about their views and practices on software architecture in the questionnaire to assess their baseline and if there are obvious results that could be attributed by their baseline.

Study Size The small amount of participants makes it very hard to do a quantitative analysis.

Selection Bias The participants were all PhD and master's students, although most of them were also working as developers in the industry. It is possible that ArchWiki provides different degrees of utility to different groups of users. It is, for example, plausible that ArchWiki provides more utility to open source developers who have limited options for face-to-face communication.

Study Method The study consisted only of a simple tutorial with predetermined steps, although participants were allowed stray from it. Because of this tutorial, ArchWiki wasn't really used in a real setting and imagination was required from the participants to envision how it would be used in a real setting.

Collaboration The tutorial did not include collaboration with other users and also did not build on an already existing ArchWiki model to keep it simple. The users had to envision how ArchWiki would work with collaboration and a more complex ArchWiki model with more content.

Experimenter Bias The questionnaire created by us and the discussion was led by us. Bias in the questionnaire and during the discussion could have led to different responses by the participants. The questionnaire was reviewed first by others and the pilot study was performed to prevent such bias.

Lack of Anonymity The group discussion prevented the participants from giving anonymous feedback. This could possibly have influenced participants' answers.

Web 2.0 Threats

Being a Web 2.0 application also comes with its own problems regarding threats to the validity of the data.

Users, Usage, and Content In Web 2.0 applications most of the utility is provided by the users themselves and the application only provides a framework. For example, the MediaWiki software supporting Wikipedia provides no immediate benefit to its users. Only when users add content to the Wikipedia it becomes of utility to the users. The utility of the tool is thus highly dependent on the amount of users and the amount of content they provide. Web 2.0 applications often do not prescribe how they are used, but their usage is determined by the users themselves. The usage often evolves with time also. The way a tool is used also determines its utility. Even with the single instance of an application the ways the tool is used can differ. For example, the German and Japanese Wikipedia use it in a different way. Because ArchWiki is just an experimental tool it is not used yet and therefore it is hard to determine the full utility of the tool. Participants could only assess features with immediate benefits such as implementation mappings and context views and had to mostly use their imagination to form an opinion about the tool. The users, the way they use the tool, and the content they provide become independent variables we would have liked to measure, but we were unable to do that, because these users and content do not exists at this moment.

Usability Usability of an application always plays a role in the utility. However because Web 2.0 tries to leverage the long tail, usability is of more importance. However it is unclear to what a degree this applies to ArchWiki, because it is not aimed at the general public, but more at software developers. We evaluated the usability, but the relation between the usability and the utility is not clearly defined.

5.8.2 Threats to External Validity

The external validity refers to the generalizability of the experiment's results.

Participants

The participants are not the ideal representation of the intended user base, because they are all volunteers and with an academic background, which could make them more receptive to new technology. Also the tool is intended for users who support the architecture centric approach which is not used in practice. The intended user base is also not completely specified and subject to change. The user base can, for example, be further narrowed to online open source project developers.

External Web 2.0 Factors

There are also external Web 2.0 factors not influenced by the tool itself. Such as that the potential users have to be ready to use such a tool for it to be adopted. For example, Google Wave was a Web 2.0 collaboration tool that was introduced as the next big thing, however users did not adopt it and Google pulled the plug fairly quickly. The reasons for not adopting it are not very clear. It could have been bad timing, not being able to find a specific use for it, and/or not being able to invite other users. So even with our participants it is possible they provide different data if we conduct the experiment again at a later moment when they have a different mindset.

5.8.3 Ecological Validity

The experiment was performed in a artificial setting and it is hard to generalize the results to real life settings.

Tasks The task of the participants of the users consisted of performing a tutorial, which is is not comparable how the tool would be used in real life.

Content The participants worked with an empty project with no initial content. Apart from the very early stages, users of ArchWiki would be working with projects that already have substantial content and this content would only increase over time.

Collaboration The participants did not collaborate with ArchWiki although it is a collaboration tool. In real life ArchWiki would be continuously used by the users to collaborate on the software architecture.

5.9 Conclusion and Discussion

The data gathered from this experimental evaluation study has given us insights on the current views and practices in the area of software architecture. The preliminary findings on the usage of ArchWiki, which uses Web 2.0 features to support the software architecture centric view, indicate that the approach has potential to support architectural knowledge management.

In the next three sections we will answer each of the three research questions with the data we gathered from our evaluation study.

5.9.1 Architectural Knowledge Retrieval

Program comprehension is for our participants an important, time consuming task. They try to use architecture documents whenever possible, but they are often not available or easily found. Also the design decisions made while making the architecture documents are often not documented. They also did not have confidence that the architecture documents were correct or in sync with other artifacts such as the source code. This is further complicated by the low participation of team members in maintaining the documents and the lack of traceability between development artifacts, such as source code and requirements documents on one side and the architecture documents on the other side. Also using different tools for each type of artifact does not help.

We do think that ArchWiki can help with addressing some of these problems and this is supported by the answers of the participants. ArchWiki supports program comprehension in the following ways:

- The integration with the IDE gives the users easy access to the architecture documents and consistent way to store software architecture documents.
- The mapping between the implementation and architectural documentation provides support for bottom up analysis by showing the corresponding architectural component when editing a certain piece of source code and it provides support for top down analysis by linking to the corresponding pieces of source code when viewing an architectural element.
- The wiki enables an easy way to provide textual documentation, in which, for example, the design decisions regarding a certain architectural element can be further elaborated.
- Tags provide an easy way to identify to which categories an architectural component belongs to makes it easy to locate other elements of the same categories. Also it helps to provide a common vocabulary among the users.
- Notification of changes to the architecture helps keeping users up to date with the software architecture. Without notifications users would not be aware that the architecture has changed at all. Even if they would be made aware that a change has been made (by for example a change in modification date of a file), the user would have to compare the old architecture to the new architecture and synthesize the changes from the differences. People learn easier when information is presented to them in small chunks, and notifications provide just that.
- Bookmarks can help to manage the complexity of large architectural models. By bookmarking elements users can keep tabs on architectural elements which are of more importance to them.
- The tight coupling between the source code and architecture will help to keep them synchronized and thus help prevent architectural degradation.
- The tool also adds meta data to the artifacts, such as when changes were made and by who. This allows for a lot of possibilities such as finding the right person to help you with the comprehension of a certain architectural element.

Following these results we can answer research question RQ1 that ArchWiki supports architectural knowledge retrieval.

5.9.2 Architectural Knowledge Documentation

Software architecture documenting practices of our participants currently consists of writing architectural design in Word documents and the occasional (UML) structural diagram and writing low level documentation such as comments in the source code and Javadoc. Comments and Javadoc are easily shared because they are tied to the source code, but the sharing of higher level documentation is more problematic. There is usually very little high level documentation and if there is, it is usually not easily found. The documents are not frequently used and updated so the confidence in the correctness of the documents is usually low.

The reason for documenting the software architecture was not for the user itself, but more to help with the comprehension of other team members and to assure everyone has the same mental picture of the software architecture. This suggests that there is a collaborative element to software architecture. Also the documenting of design decisions is found important as is the creation of high level abstract view. The two main problems with documenting the architecture are that it takes too much time and the current tools are inadequate.

We think that ArchWiki provides some of the features that the participants would like, while it addresses the problems they have with the current tools:

- The integration of the tool with the IDE makes it easy to create new architectural documentation, because you do not have to context switch between different tools. It also takes away the hassle of storing it and sharing it with other users.

- When editing a piece of source code the implementation mapping presents the user with the corresponding architectural documentation. The tool saves the user time by preventing the user from having to search for the corresponding software architecture and lowers the threshold to document the architecture. Also the tool makes it easy to navigate within and among the different types of artifacts by showing appropriate links.

- The wiki provides a good way to document the architecture and also lowers the threshold to document the software architecture, because of its ability to input informal, unstructured data. The wiki is also a good location to document design decisions.

- By tagging you can help to categorize architectural elements, which can help other users with program comprehension. It is also such a lightweight tool that is lowers the documentation threshold.

- Notifications help with other users' program comprehension by keeping them up to date of the changes you make.

Following these results we can answer research question RQ2 that ArchWiki supports architectural knowledge documentation.

5.9.3 Collaboration

The ways on collaborating on software architecture are very minimal and informal. Collaboration often consists of organizing a physical meeting with the developers and then discuss the software architecture. The resulting artifacts are often diagrams on whiteboard and Word documents. The whiteboard diagram can be captured by taking a picture of the whiteboard. Often there are no specific collaboration tools for software architecture. The Word files and the picture files containing the diagrams are often shared through a revision control system.

Two often mentioned problems were:

- Team members were not made aware when architecture documents changed and whether the changes were of interest to them.
- The participation of other team members with collaboration on software architecture is low.

These problems of not having people to collaborate with and not knowing when to collaborate are fundamental problems and indicate that the level of collaboration on software architecture is currently low. This is supported by the fact the currently used tools are not designed specifically for software architecture and even if they are, then the tools are designed for a single user. This is in contrast with the other development tasks such as programming and testing, which become increasingly more integrated and collaborative. An example of this is IBM Rational Team Concert in which source code, bug reports and test cases are all integrated in a collaborative tool. Perhaps this is caused by the perception that software architecture is such an infrequent task that it could be accomplished with only physical meetings.

ArchWiki on the other hand is focused on the software architecture centric approach in which the software architecture plays an important role throughout the development process and is continually evaluated and changed. Participants agree that collaboration can help with software architecture so supporting collaboration is an important aspect of the ArchWiki tool. ArchWiki supports collaboration in a number of ways. The most obvious collaboration feature of ArchWiki is the inclusion of the Wiki, which is a collaborative text editor. Also ArchWiki provides features to integrate software architecture into the current work flow:

- The integration of the tool into the IDE makes sure that every user has the right tool set to view and edit the architecture. There is no more splintering of different tools.
- Instead of explicitly editing a certain file or multiple files for the different artifacts, ArchWiki intends to only have the user specify the architecture once, and then have full access to all relevant artifacts from within ArchWiki. These artifacts range from the architectural artifacts such as structure diagrams, Wiki pages and tags to other software development artifacts such as source code. Also the user always uses the current version of the architecture and does not have to search for the right version of each of the documents.

Another collaborative area is awareness. ArchWiki features a user model to enable features in this area:

- ArchWiki provides a notification feature to let other users know what changes you made.
- Users can receive notifications from other users to be aware of their work. They can specify which notifications they receive by using the filtering feature.

Following these results we can answer research question RQ3 that ArchWiki supports collaboration in architecture knowledge management.

5.10 Challenges

The feasibility of the ArchWiki tool relies on a number of factors. First the tool itself is in an alpha state is not yet ready to be used in production. The selected Web 2.0 features could be increased, decreased and also implemented differently. A small set of features could provide too little benefit for users, a large set of features could scare users off and affect usability, and features not implemented to a user's liking could also prevent a user from using it. The features also have impact on other features. For example, Google Maps may only appeal to geography enthusiasts, and not to car drivers. The car driver may be interested in traffic information. If you integrate traffic information with Google Maps the utility for car drivers will increase to more than what it would be for only textual traffic information. Web 2.0 features often enhance each other and the utility of the combination of two features is often greater than their sum. Thus, it is a challenge to find the best mix of features. We plan to further extend Archwiki in a number of ways:

- The collaboration model is currently not ideal. The data (except wiki pages) are currently stored locally and this data will only be shared once the user commits the data and another user retrieves that data from the RCS. This also introduces the problem of conflicting data. A more ideal solution would be to centralize access to the data. This approach has not been done, because of the dependency on ArchStudio which uses a local data, single user approach.
- The user model is very simple and could be extended with, for example, user groups and a point feature.
- The amount of filters is very limited and could be extended with, for example, filters that show only notifications of team members that you are working with.
- The visualization should be more polished. It can, for example, be optimized to work with multiple monitors.
- Some functionality does not work as expected.
- The connection to the wiki pages is done by simply showing the wiki page as it would be done in a normal browser. The wiki page shows now some unnecessary elements could be hidden.
- The connection to the source code is done by simply specifying a relative path. A more advanced method which will automatically link source to architectural elements with pattern matching might be warranted.

- A number of features could be added to increase its utility such as linking more artifacts such as mailing lists etc.

Also factors outside of the tool itself impact its feasibility and one of the most important is the factor of user adoption. In general ArchWiki's features do not provide immediate value by itself in a way like an IDE's language tool that gives an error message if a user compiles bad code. The way of ArchWiki, which also applies to software architecture and Web 2.0 in general, is that it provides users the ability to provide benefit to themselves in the long run. A lot of the tool's feasibility therefore lies in its users and its willingness to adopt it. The case of Google Wave is an example in which a tool laden with features perished, just because users would not adopt it. There are a numerous obstacles in adopting a tool like ArchWiki:

- The users should be willing to embrace the software architecture centric approach in which software architectural tasks are part of the work flow just a coding is. In the evaluation study one feedback we got was that normal developers should not be able to do architectural tasks. For users who think the same this tool is not an option.
- We also got some feedback that some users will not be willing to do architectural tasks regardless ArchWiki's attempts to make architectural tasks easier and more fun to do.
- Company culture and practices might have to change to adopt ArchWiki and include it in the work flow. For example, some users prefer face-to-face meetings with whiteboard drawings over the usage of a tool.

Low adoption has several consequences for the usefulness of the tool:

- If not enough users or incapable users use the tool the architecture might become incomplete or incorrect.
- Low participation count might affect if new users are willing to use the tool.
- Incomplete or incorrect architecture might impact the willingness of a user to adopt the tool.

Chapter 6

Related Work

In this chapter we discuss research and technologies related to our thesis. We have divided this chapter into two sections: related technologies and related research.

6.1 Related Technologies

One of the most important aspects of ArchWiki is collaboration. In the next section we will discuss a number of collaboration tools for software development. This section is followed by a sections about ADL tool suites, AKM tools, and other tools. In the concluding section we will position ArchWiki in the context of these related technologies.

6.1.1 Collaboration Tools

Revision Control Systems

A vital part of collaboration is the ability to share artifacts such as source code and other documents. Revision control systems (RCS) [72], which form an important aspect of Software Configuration Management (SCM) [2], store files in a central repository and allow developers to share source code with other developers and enable them to work concurrently on the same files. Because they also maintain all the versions of the files it is even possible to access and work on a different version of a file. Committing a file shares it with other developers by storing it on the repository and introducing a new version of the file.

Revision control systems were available as early as 1975 with SCCS [60] and have continued to play an increasingly important role in software development, especially in Open Source. Today, version management is essential to software development and is considered the most critical component of any development environment [17]. Two of the most widely used revision control systems are Concurrent Versions System (CVS [30]) and Subversion (SVN [58]). The newest generation of revision control systems are the distributed revision control systems (DRCS) [52], such Mercurial [68] and Git [69]. These systems operate in a peer-to-peer manner in which each user operates on his own copy of the repository and separates committing changes from publishing them.

Issue Tracking Systems

An issue tracking system (or bug tracking system) is a database that maintains and manages a list of issues. These can be issues reported by customers, but also by members within an organization. The main benefit of a bug-tracking system, such as BugZilla,[1] is to provide a clear centralized overview of development requests (including both bugs and improvements, the boundary is often fuzzy), and their state.

IDE Integrated Collaboration Tools

Jazz [13] started as a collaboration tool, which focused on team awareness and communication from within the IDE. It provided features, such as instant messaging and whiteboard sharing abilities. Since then, the Jazz platform has evolved to a much broader scope [26], providing classic asynchronous collaboration for work items, synchronous collaboration in the context of the work, and RSS feeds for notification of changes, approvals, reviews of changes, and so on. It also integrates bug reports, change sets, builds, tests and is capable of linking these artifacts to each other.

Cheng et al. [14] discuss the extensibility of the Jazz platform. They used Jazz to implement research prototypes that promote developer communications, management of decision rights and decision points during software development processes, and the scheduling of software projects. They describe their experiences extending and using Jazz and Rational Team Concert (RTC)[2], an agile collaborative integrated development environment based on Jazz, and provide some recommendations for other researchers considering extending the Jazz platform.

Mylyn started in 2005 as Mylar [37], a degree-of-interest model for IDEs. The idea for this model originates from the way IDEs present elements of interest to the programmer. When performing a task only a relative small number of elements are of interest to programmer. As the size of the system increases the elements relevant to the current task become only a small subset of those shown in the IDE views. Especially in enterprise-scale systems the large number of elements unrelated to the current task occludes the relevant information. It is often not possible for the programmer to show all relevant elements in one view and he is forced to work with long scrolling lists to find the elements. As a result, programmers spend more time looking for related information than they do performing the task.

The Mylar tool was developed to help programmers focus and work on the code related to a task. Mylar monitors a programmers' activity and captures the relevance of code elements to their task in a degree-of-interest (DOI) model. The model is created and used by the Mylyn Eclipse plugin. This plugin modifies several Eclipse views to view interesting elements more clearly and to hide non-interesting elements. Also, tasks are now explicit elements and it is possible to change between tasks or schedule the task in the Mylyn task list. Every activated task builds up a DOI model called a task context. To support collaboration it is possible to integrate Mylyn tasks with existing team collaboration tools such as issue tracking systems and revision control systems. Tasks can be stored locally or in remote task repositories such as Bugzilla. Using existing issue tracking systems increases both the effectiveness of tasks, by

[1] http://www.bugzilla.org

[2] https://jazz.net/products/rational-team-concert/

making it possible to collaborate on them, and issue tracking system, by integrating them into the IDE. It is now, for example, possible to create a task (bug report) from the error log view and attach stack traces and task contexts to them. Collaboration is also improved by integrating tasks with revision control systems. Mylyn automatically groups changes made during a task into a change set. This way tasks are also explicit on the RCS repositories. Every change in the revision history can then be linked to a particular task.

Validation of the tool was done by conducting a field study to assess the impact on productivity [38]. To approximate the productivity, Kersten defined the "Edit ratio" as:

> The relative amount of edit vs. selection events.

The results indicated that the use of the tool indeed increased the edit ratio.

Both, Jazz and Mylyn, are great examples of successful task integration into the IDE to increase collaboration. Mylyn is now used by thousands of programmers.

6.1.2 Architecture Description Language (ADL) Tool Suites

AcmeStudio [62] is an architecture development environment, written as a plugin for Eclipse. With the tool architectural models can be created in the Acme ADL. Its main screen is a visual editor in which architectural models can be created by connecting visual representations of architectural elements defined in Acme. A key aspect of AcmeStudio is the ability to easily and quickly create new Architectural Styles. A style defines the available vocabulary in a particular domain. Examples of such styles are Pipe-and-Filter and Client-Server. Rules can be assigned to a style to define the correct composition of an architecture in this style. AcmeStudio allows different analysis tools to be integrated, and applied to appropriate architectural styles.

Dashofy et al. [19] present Archstudio 4, an architecture development environment very much alike AcmeStudio. The architectural models are created in the xADL 2.0 ADL. Archstudio comes with a visual box-and-arrows editor called Archipelago. It also provides a tree and table-based editor called ArchEdit. It also ships with a plug-in-based framework for integrating analysis tests called ArchLight. It provides a unified user interface in which various tests can be applied to and run on architectural models. The key feature of Archstudio and the underlying modeling language xADL 2.0 is their extensibility. Tools can read the xADL 2.0 schema that defines the syntax of the ADL and automatically generate libraries used for building tools that interact with those new features of the ADL. We extended ArchStudio 4 in our approach to view and edit the structural diagrams.

6.1.3 Architecture Knowledge Management (AKM) Tools

Architectural Knowledge (AK) is seen as an increasingly important asset in software development. AK does not only include the architecture design but also the decisions, rationale, assumptions, context, alternatives and other factors that together determine architecture solutions. Architecture Knowledge Management (AKM) tools provide means of managing this knowledge. The next paragraphs describe several AKM tools.

Liang et al. [45] proposes the Knowledge Architect tool suite for architecture knowledge management. Plug-ins created for individual tools, such as Word, can share

their AK for specific activities through a central knowledge repository. The Knowledge Explorer provides various visualizations to inspect AK entities and their relationships.

Babar et al. [6] developed a framework for managing AK: Process-centric Architecture Knowledge Management Environment (PAKME). Aman-ul-haq and Babar [74] created the Automatic Architecture Knowledge Extraction Tool (AAKET) to extract AK from documents, which is expected to perform most of the time-consuming tasks semi-automatically with minimum human intervention.

Capilla et al. [11] present a web-based tool called ADDSS based on the decision view [23]. In this tool design decisions can be captured and connected to requirements. Users can navigate the design decisions and the architecture. Users assisted by the tool perceived that building and visualizing the architectures iteratively, eased the construction process and the understandability became higher.

Jansen and Bosch [36] propose their Archium approach that is a combination of the traditional ADLs, in which an architecture is modeled to produce the component-and-connector view, and the architectural management tools, which aims to manage architectural knowledge, such as design decisions, rationale, design rules, design constraints and solutions.

6.1.4 Other Tools

Hipikat [77] is a tool that forms an implicit group memory from the information stored in a project's archives, and then recommends artifacts from the archives that are relevant to a task that a newcomer is trying to perform. Their approach has two parts. First, they form the implicit group memory from the artifacts and communications stored in a project's history. Second, they present to the developer artifacts selected from this implicit group memory that may be relevant to the task being performed.

While ArchWiki relies on the users to connect the different artifacts in the group memory together, Hipikat uses an automated system. In an initial qualitative study subjects implemented changes to a medium-size software system. The subjects reported that Hipikat helped them to start the assigned task. The recommendation system of Hipikat is similar to ArchWiki as the connections and hyperlinks to other artifacts can also be seen as a recommendation system.

Happel et al. [32] propose the TEAM approach towards knowledge sharing in distributed development teams. The TEAM approach is based upon the concept of a Software Engineering Semantic Web, which itself is based upon the concept of the Semantic Web. Ontologies are used to capture the internal structure of software development artifacts as well as different semantic interrelations among them. For the easy acquisition of semantic knowledge, TEAM provides user-friendly lightweight semantic authoring and annotation. Besides explicit knowledge from development artifacts and written documents, TEAM also captures implicit knowledge knowledge from the developer's interactions inside the IDE. To access the knowledge inside the Software Engineering Semantic Web TEAM provides semantic search and recommendation functionality.

6.1.5 Insights from Current Related Tools

Looking at the approaches in collaboration in software development we observe that the early collaboration tools were one-dimensional and intended for a specific purpose, such as revision control and bug tracking. New collaboration tools such as Mylyn and Jazz focus on integrating the old tools and connecting their artifacts together. Other experimental tools such as Hipikat and TEAM assemble a model containing these artifacts and use this model to provide features such as recommendation features which can suggest interesting artifacts. Unfortunately, these collaboration tools in general do not support the integration of architectural tools and artifacts.

Early software architecture tools, such as the tool suites created to support ADLs, focused primarily on the “end product”, which in general is the component-and-connector style diagram. Recently software architecture has seen a shift from the component-and-connector diagrams to a more broader view. In this view software architecture is seen as a set of design decisions and the focus has shifted from focusing solely on the "end product" to also give more attention for the process of creating the “end product”. This resulted in the introduction of architecture knowledge management tools. These focus on new artifacts such as design decisions and rationale and linking the artifacts in a meta-model.

The direction of general software development tools has gone in the direction of Web 2.0, with the integration of different artifacts and tools with current generation tools such as Jazz and Mylyn. Experimental tools such as Hipikat and TEAM exploit connections between those artifacts. In the field of software architecture there also has been a shift to a more Web 2.0 way of thinking with the rise of AKM tools and the promotion of design decisions and rationale to first class entities, and interconnecting them to provide traceability.

However, there is no integration between the different kinds of software architecture tools, such as ADL tool suites and AKM tools, let alone integration between the software architecture tools and the general software development tools in the IDE. It is very hard to realize our Web 2.0 infused software architecture centric based work flow without those integrations. The direction the current tools are evolving too is certainly compatible with ArchWiki. ArchWiki expands on the current tools by integrating more artifacts and tools. Additionally, ArchWiki places high value on Web 2.0 concepts such as *Users*, *Usability*, and *Collaboration* by offering features such as: users as first class entities, personalization, and awareness.

6.2 Related Research

6.2.1 Web 2.0 in Software Development

Jazz also includes Web 2.0 features such as the tagging of work items. Treude et al. [73] have researched the adoption of these tags by software developers, and the role of tagging in team-based software development. They conclude that the research has shown how the social computing mechanism of tagging has been adopted and adapted by a large software development team. The main advantages of using tags in software development are their flexibility and their lightweight, bottom-up nature.

This research backs up the idea that using Web 2.0 technologies like tagging can help with the collaboration on software architecture artifacts as well.

Calefato et al. [10] tries to reduce socio-cultural distances within distributed development teams coping with geographical distance by extending the Jazz platform to embed social information collected from social networks into the Jazz.

The linking of artifacts and storing them in a single repository provides interesting opportunities for data mining the repository, because artifacts are now linked explicitly and in the past they were linked using heuristics. Nguyen et al. [50] report on their experiences on mining the Jazz repository. For example, they examined if the Jazz team experienced communication and task completion delays due to the large geographic distance between the teams. Their analysis suggests that the Jazz team did not experience as much delay as reported in previous literature.

Herzig et al. [35] also report on their experiences in mining the Jazz repository. They mentioned that the advantage of Jazz is based on *work item edges* that connect different artifacts together. The resulting data contain many new insights into development processes and artifact dependencies that cannot be retrieved using other project repositories.

Chau and Maurer [12] report on an exploratory case study on the use of a Wiki-based experience repository (MASE). MASE is a modified Wiki implementation focused on general knowledge management. It offers, for example, a chat function and integration with Microsoft NetMeeting.[3] ArchWiki uses the standard MediaWiki implementation, but can possibly benefit from using a Wiki implementation tailored for knowledge management. In this case study they found evidence from the industry that supports the use of an informal knowledge sharing, such as ArchWiki. For example, they found it important for such a tool to support easy collaboration and communication. They also found a need to support both structured and unstructured knowledge. Other goals of ArchWiki such as making contributing easy and more focus on the day-to-day software developers are also found important.

Farenhorst et al. [24] also used a wiki to document the architectural knowledge. They report on their experiences with creating and using a wiki environment in the architecture department of NPK, a large Dutch software development organization. The wiki was built using the Confluence Enterprise[4] wiki software. An important issue was the adoption of the wiki. Therefore, they used several wiki patterns [47] to spur adoption such as the Critical Mass pattern, the Champion pattern (in which on of the author acted as wiki champion), the Moderator pattern (in which three architects were appointed Moderators of specific parts of the wiki), and the FAQ pattern (they made a page with frequently asked questions to help wiki users with basic functionality and features). Their efforts resulted in wide-spread enthusiasm throughout multiple departments within the organization. One of the problems they encountered was that they found the inclusion of diagrams cumbersome, because the diagram would change frequently and had to be updated manually. In ArchWiki the diagrams are shown in their native editor Archipelago and therefore don't have to be embedded in the wiki.

Solis et al. [66] recognize the strength of using a lightweight tool such as a wiki to manage architectural knowledge. They present Spatial Hypertext Wiki (ShyWiki),

[3] http://en.wikipedia.org/wiki/Microsoft_NetMeeting

[4] http://www.atlassian.com/software/confluence/overview

an AKM tool in the form of a wiki which provides spatial hypertext capabilities. The difference with a normal wiki is that it allows pages to contain a set of notes, which a user can manipulate (add, remove, move, etc). Notes can also contain links or other notes. Transclusion is also possible, which allows the inclusion of a note defined inside another document. Notes can be moved freely in the interface with drag-and-drop.

6.2.2 Insights from Current Web 2.0 Experiences in Software Development

A number of tools, such as Jazz, have successfully introduced Web 2.0 technologies, such as tags. Wiki's have also been used to share knowledge. One of the most important success factors for a wiki is its adoption, which will probably be also the case for ArchWiki. Spending considerable effort to increase adoption, such as using wiki patterns [47], will most likely be required to make a Web 2.0 tool such as ArchWiki successful.

The experiences with wiki's have also been in a standalone setting, without any integration with other artifacts and tools. This has lead to problems with wiki's, such as reported by Farenhorst et al. [24], in which they found the inclusion of diagrams cumbersome, because the diagram frequently changed. In this case, ArchWiki addresses this problem by integrating the ADL tool which creates the diagram with the wiki.

Chapter 7

Conclusions and Future Work

7.1 Summary

For this master's thesis we investigated whether Web 2.0 is able to support the field of architecture knowledge management. We did this by studying the current practice in software architecture knowledge management and observing the current applications of Web 2.0. From this we created the ArchWiki approach with an accompanying tool. We evaluated the potential of the ArchWiki approach and tool by performing an initial evaluation to gauge their potential. We conclude that ArchWiki was able to support architecture knowledge management and we answered all three research questions, stated in Section 5.1, positively.

7.2 Answers to the Research Questions

We summarize the three sub research questions:

RQ1 Can ArchWiki Support Architectural Knowledge Retrieval?

Participants of the evaluation study all found that program comprehension to be a time consuming task, and they indicated that they had problems retrieving the architectural knowledge.

In our evaluation study we found that ArchWiki supports architectural knowledge retrieval (Figure 5.15 Question 25a). ArchWiki supports program comprehension in a number of ways, such as: easy access, traceability, textual documentation, easy categorization, awareness, bookmarks, and meta data.

RQ2 Can ArchWiki Support Architectural Knowledge Documentation?

Participants found it important to document the architecture, especially to aid in the comprehension of other team members and to assure everyone has the same mental picture of the software architecture. They found that documenting the architecture takes too much time and that the current tools are inadequate.

In our evaluation study we found that ArchWiki supports architectural knowledge documentation (Figure 5.15 Question 26a). ArchWiki supports documenting the architecture in a number of ways, such as: tool integrating, context-sensitivity, traceability, tagging and notifications.

RQ3 Can ArchWiki Support Collaboration in Architecture Knowledge Management?

The view on collaboration of the participants strengthened our idea that collaboration can help with software architecture and that a collaborative view such as Web 2.0 can help support that. Currently participation on software architecture is low and is usually done face-to-face, without using specific collaboration tools for software architecture knowledge management. The participants agree that collaboration can help with software architecture.

In our evaluation study we found that ArchWiki supports collaboration in architecture knowledge management (Figure 5.15 Question 28a). ArchWiki supports collaboration in a number of ways, such as: tool integration, central single instance, traceability, awareness, wiki pages, and collaborative tagging.

From our evaluation of ArchWiki and the answers to our sub research questions we conclude that Web 2.0 concepts can be used to support software architecture knowledge management.

7.3 Contributions

The contributions we have made in this master's thesis are as follows:

- The ArchWiki approach, which uses Web 2.0 concepts to support software architecture knowledge management.
- The ArchWiki tool, which is developed to support the ArchWiki approach. The tool contributes, among other things, the following elements to software architecture management:
  - Integration between different development tools, such as source code editor, visual ADL editor, and textual documentation.
  - Context-sensitive Element View, which serves as the hub of architectural information.
  - Integrated wiki pages within the IDE.
  - A meta model that connects different software development artifacts.
  - Easy navigation through hyperlinks, based on the underlying model.
  - Collaboration support with explicit user entities and make AKM a social experience.
  - The use and persistence of meta data, such as user actions.
  - Awareness through RSS notifications.
  - Personalized views through tags, bookmarks and filters.
  - Contributions are of fine granularity and incremental.
  - Recommendations, based on the underlying model.
- An initial evaluation of the ArchWiki approach and tool.

7.4 Future Work

In the following section we compiled a list of future work divided in three categories: tool features, evaluation study, and research opportunities.

7.4.1 Tool Features

We think the following tool features have the potential to increase ArchWiki's utility:

- Integrate the meta data, such as actions, into model. This will make it easy to see the history of architectural components.
- Explore the history of the ArchWiki artifacts. Make it possible to look at a snapshot of the whole ArchWiki model at a particular moment.
- Add a diff visualization between two different snapshots.
- Extend actions to make those actions revertible. This might bring the rest of ArchWiki in line with MediaWiki pages, in which this is already possible, and can possible lower the barrier of making changes.
- Recommendation features based on data mining the model and usage patterns [79]:
  - Recommend related architectural components.
  - Recommend users that can provide additional architectural knowledge.
  - Recommend often accessed artifacts to all users.
  - Recommend to delete artifacts that are never accessed.
- Add more user and social features, such as governance structure, reputation, karma points, groups etc.
- Integrate more development artifacts such as requirements, use cases, test cases, benchmarks, and bug reports.
- Integrate forums and mailing lists.
- Add a view to directly visualize the underlying model.
- Add advanced search features, which can also use the underlying model and corresponding properties, such as distance in the model.
- Support customized and personalized views of architectural structures (configurations).
- Bookmark more types of elements, such as views, and enable users to share them.

7.4.2 More Extensive Evaluation Study

To better assess the utility of the tool would require the following:

- Use ArchWiki with a large and more complex software architecture.
- Longitudinal study to assess how the utility of the tool evolves in the long term. This will enable the integration of the tool into the current work flows to possibly stabilize to a (sub-)optimal state. Also the exit and entrance of participants can be examined.
- More participants to assess the collaboration features of the tool. Also different kinds of participants such as 'champions' who lead in using the tool and users that only consume data.
- Evaluate the quality of the notifications. Too many, too few, too much information, too little, more integration?
- Conduct an evaluation study to assess the compatibility of ArchWiki, with decreased overhead, with methodologies such as agile, which moves away from comprehensive software architecture documentation, partly because of high overhead.
- Quantify how ArchWiki affects participation in software architecture and what effect participation has on architecture knowledge management.
- What is the difference in effect of ArchWiki on (globally) distributed software development versus co-located software development.
- What is the right granularity to create and connect artifacts. Is it better to connect the implementation to the architecture at the block or line level?

7.4.3 New Research Opportunities

Insights from the the ArchWiki approach and tool lead to the following research opportunities:

- New data mining opportunities, introduced by the model which interconnects artifacts, users, and actions.
- Study the evolution of the architecture over time.
- What constitutes collaborative architecture?
  - Is it different from individually created architecture?
  - Is it better or worse?
  - Is it easier to comprehend?
- What value does each different artifact type add?
- What kinds of actions should be recorded?

- Study the effects of incremental documentation and architecture recovery.
  - Is the documentation itself different?
  - Is is easier to document?
  - Is it easier to comprehend?

Bibliography

[1] Mark S. Ackerman, Volker Wulf, and Volkmar Pipek. *Sharing Expertise: Beyond Knowledge Management*. MIT Press, Cambridge, MA, USA, 2002.

[2] Robert Aiello and Leslie Sachs. *Configuration Management Best Practices: Practical Methods that Work in the Real World*. Addison-Wesley Professional, 1st edition, 2010.

[3] Chris Anderson. The long tail. `http://www.wired.com/wired/archive/12.10/tail_pr.html`. Accessed: 15/05/2012.

[4] K Arnold and J Gosling. *The Java Programming Language*, volume 2nd of *The Java Series*. Addison-Wesley, 1996.

[5] Muhammad Ali Babar, Torgeir Dingsyr, Patricia Lago, and Hans van Vliet. *Software Architecture Knowledge Management: Theory and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[6] Muhammad Ali Babar and Ian Gorton. A tool for managing software architecture knowledge. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, SHARK-ADI '07, pages 11–17, Washington, DC, USA, 2007. IEEE Computer Society.

[7] F. Bachmann and F. Merson. Experience using the web-based tool wiki for architecture documentation. Technical report, CMU/SEI-2005-TN-041, Carnegie Mellon University, Software Engineering Institute, 2005.

[8] Frederick P Brooks. *The Mythical Man-Month*. Addison Wesley, 1995.

[9] E. Brynjolfsson, Y. Hu, and M. Smith. From niches to riches: anatomy of the long tail. *Sloan Management Review*, 47(4):67–71, September 2006.

[10] Fabio Calefato, Domenico Gendarmi, and Filippo Lanubile. Embedding social networking information into jazz to foster group awareness within distributed teams. In *Proceedings of the 2nd international workshop on Social software engineering and applications*, SoSEA '09, pages 23–28, New York, NY, USA, 2009. ACM.

[11] Rafael Capilla, Francisco Nava, Sandra Pérez, and Juan C. Dueñas. A web-based tool for managing architectural design decisions. *SIGSOFT Softw. Eng. Notes*, 31(5), September 2006.

[12] Thomas Chau and Frank Maurer. A case study of wiki-based experience repository at a medium-sized software company. *Proceedings of the 3rd international conference on Knowledge capture KCAP 05*, pages 185–186, 2005.

[13] Li-Te Cheng, Susanne Hupfer, Steven Ross, and John Patterson. Jazzing up eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, eclipse '03, pages 45–49, New York, NY, USA, 2003. ACM.

[14] P. Cheng, S. Chulani, Y. B. Ding, R. Delmonico, Y. Dubinsky, K. Ehrlich, M. Helander, T. Klinger, A. Kofman, P. Matchen, V. T. Rajan, G. Saadoun, A. Sempere, P. Tarr, C. Williams, P. F. Xiang, A. Yaeli, S. X. Yang, and A. Ying. Jazz as a research platform: experience from the software development governance group at ibm research. In *Proceedings of the 1st International Workshop on Infrastructure for Research in Collaborative Software Engineering (iReCoSE)*, November 2008.

[15] Eric Clayberg and Dan Rubel. *Eclipse Plug-ins*, volume 2006 of *The Eclipse Series*. Addison-Wesley Professional, 2008.

[16] Alistair Cockburn. *Agile Software Development: The Cooperative Game (2nd Edition) (Agile Software Development Series)*. Addison-Wesley Professional, 2006.

[17] CollabNet. Rapid Subversion Adoption Validates Enterprise Readiness and Challenges Traditional Software Configuration Management Leaders. `http://www.open.collab.net/news/press/2007/svn_momentum.html`, 2007.

[18] James O. Coplien and Gertrud Bjørnvig. *Lean Architecture: for Agile Software Development*. Wiley Publishing, 2010.

[19] Eric Dashofy, Hazel Asuncion, Scott Hendrickson, Girish Suryanarayana, John Georgas, and Richard Taylor. Archstudio 4: An architecture-based meta-modeling environment. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, ICSE COMPANION '07, pages 67–68, Washington, DC, USA, 2007. IEEE Computer Society.

[20] Remco C. de Boer and Rik Farenhorst. In search of 'architectural knowledge'. In *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, SHARK '08, pages 71–78, New York, NY, USA, 2008. ACM.

[21] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, SIGDOC '05, pages 68–75, New York, NY, USA, 2005. ACM.

[22] Alan Dix, Tiziana Catarci, Benjamin Habegger, Yannis loannidis, Azrina Kamaruddin, Akrivi Katifori, Giorgos Lepouras, Antonella Poggi, and Devina Ramduny-Ellis. Intelligent context-sensitive interactions on desktop and the web. In *Proceedings of the international workshop in conjunction with AVI 2006 on Context in advanced interfaces*, CAI '06, pages 23–27, New York, NY, USA, 2006. ACM.

[23] Juan C. Dueñas and Rafael Capilla. The decision view of software architecture. In *Proceedings of the 2nd European conference on Software Architecture*, EWSA'05, pages 222–230, Berlin, Heidelberg, 2005. Springer-Verlag.

[24] R. Farenhorst and H. van Vliet. Experiences with a wiki to support architectural knowledge sharing. In *Proceedings of the 3rd Workshop on Wikis for Software Engineering (Wikis4SE), Porto, Portugal, 2008*, 2008.

[25] Martin Fowler. *Patterns of Enterprise Application Architecture*, volume 48 of *The Addison-Wesley signature series*. Addison-Wesley Professional, 2002.

[26] Randall Frost. Jazz and the eclipse way of collaboration. *IEEE Softw.*, 24(6):114–117, November 2007.

[27] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*, volume 47 of *Addison Wesley Professional Computing Series*. Addison-Wesley, 1995.

[28] David Garlan. Software architecture: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 91–101, New York, NY, USA, 2000. ACM.

[29] O Gruber, B J Hargrave, J McAffer, P Rapicault, and T Watson. The Eclipse 3.0 platform: adopting OSGi technology. *IBM Systems Journal*, 44(2):289–299, 2005.

[30] Dick Grune. Concurrent versions system, a method for independent cooperation. Technical report, IR 113, Vrije Universiteit, 1986.

[31] Carl Gutwin, Reagan Penner, and Kevin Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pages 72–81, New York, NY, USA, 2004. ACM.

[32] Hans-Jörg Happel, Walid Maalej, and Ljiljana Stojanovi. Team: towards a software engineering semantic web. In *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, CHASE '08, pages 57–60, New York, NY, USA, 2008. ACM.

[33] James D. Herbsleb and Rebecca E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*, ICSE '99, pages 85–95, New York, NY, USA, 1999. ACM.

[34] J.D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481 – 494, june 2003.

[35] Kim Herzig and Andreas Zeller. Mining the jazz repository: Challenges and opportunities. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, pages 159–162, Washington, DC, USA, 2009. IEEE Computer Society.

[36] Anton Jansen and Jan Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.

[37] Mik Kersten and Gail C. Murphy. Mylar: a degree-of-interest model for ides. In *Proceedings of the 4th international conference on Aspect-oriented software development*, AOSD '05, pages 159–168, New York, NY, USA, 2005. ACM.

[38] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, SIGSOFT '06/FSE-14, pages 1–11, New York, NY, USA, 2006. ACM.

[39] Philippe Kruchten, Rafael Capilla, and Juan Carlos Dueñas. The Decision View's Role in Software Architecture Practice. *IEEE Softw.*, 26(2):36–42, 2009.

[40] Philippe Kruchten, Patricia Lago, and Hans Van Vliet. Building Up and Reasoning About Architectural Knowledge. *Quality of Software Architectures*, 4214(4214):43–58, 2006.

[41] Philippe Kruchten, Patricia Lago, Hans van Vliet, and Timo Wolf. Building up and exploiting architectural knowledge. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 291–292, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Patricia Lago, Paris Avgeriou, Rafael Capilla, and Philippe Kruchten. Wishes and boundaries for a software architecture knowledge community. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, WICSA '08, pages 271–274, Washington, DC, USA, 2008. IEEE Computer Society.

[43] Vidya Lakshminarayanan, WenQian Liu, Charles L Chen, Steve Easterbrook, and Dewayne E Perry. Software architects in practice: handling requirements. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research (CASCON 06)*. ACM, 2006.

[44] J. Lee. Design rationale systems: understanding the issues. *IEEE Expert*, 12(3):78 –85, may/jun 1997.

[45] Peng Liang, Anton Jansen, and Paris Avgeriou. Collaborative Software Architecting through Knowledge Sharing. In *Collaborative Software Engineering*, pages 343–368. Springer, 2010.

[46] B P Lientz, E B Swanson, and G E Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978.

[47] Stewart Mader. *Wikipatterns*. Wiley Publishing, 2007.

[48] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse Rich Client Platform*. Addison-Wesley, 2010.

[49] Nature. Internet encyclopedias go head to head. `http://www.nature.com/nature/journal/v438/n7070/full/438900a.html`. Accessed: 15/05/2012.

[50] Thanh H. D. Nguyen, Adrian Schröter, and Daniela Damian. Mining Jazz: An Experience Report. In *Proceedings of the First International Workshop on Infrastructure for Research in Collaborative Software Engineering (IRCoSE) at FSE 2008*, 2008.

[51] Robert L Nord, Nanette Brown, and Ipek Ozkaya. Architecting with just enough information. In *Proceeding of the 6th international workshop on SHAring and Reusing architectural Knowledge*, SHARK '11, pages 9–12. ACM, 2011.

[52] Bryan O'Sullivan. Making sense of revision-control systems. *Commun. ACM*, 52(9):56–62, September 2009.

[53] Manoj Parameswaran and Andrew B Whinston. Social computing: An overview. *Communications of the Association for Information Systems*, 19(37):762–780, 2007.

[54] David Lorge Parnas. Software aging. In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[55] David Lorge Parnas and Paul C. Clements. A rational design process: How and why to fake it. *Software Engineering, IEEE Transactions on*, SE-12(2):251 –257, feb. 1986.

[56] D.E. Perry, N.A. Staudenmayer, and L.G. Votta. People, organizations, and process improvement. *Software, IEEE*, 11(4):36 –45, july 1994.

[57] Thomas M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[58] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[59] Alexander J Quinn and Benjamin B Bederson. Human Computation : A Survey and Taxonomy of a Growing Field. *Human-Computer Interaction*, 32457(113352):1403–1412, 2011.

[60] Marc J. Rochkind. The source code control system. *IEEE Trans. Software Eng.*, 1(4):364–370, 1975.

[61] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[62] Bradley Schmerl and David Garlan. Acmestudio: Supporting style-centered architecture development. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 704–705, Washington, DC, USA, 2004. IEEE Computer Society.

[63] Syed-Ikhsan Syed Omar Sharifuddin and Fytton Rowland. Knowledge management in a public organization: a study on the relationship between organizational elements and the performance of knowledge transfer. *J. Knowledge Management*, 8(2):95–111, 2004.

[64] Victoria E. Shipp and Peter Johnson. Supporting collaboration in the development of complex engineering software. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '11, pages 84–87, New York, NY, USA, 2011. ACM.

[65] Software Engineering Institute. Software architecture definitions. `http://www.sei.cmu.edu/architecture/start/community.cfm`.

[66] C. Solis, N. Ali, and M.A. Babar. A spatial hypertext wiki for architectural knowledge management. In *Wikis for Software Engineering, 2009. WIKIS4SE '09. ICSE Workshop on*, pages 36 –46, may 2009.

[67] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF : Eclipse Modeling Framework*. Eclipse series. Addison-Wesley Professional, 2009.

[68] Bryan O Sullivan. *Mercurial - The Definitive Guide: Modern Software for Collaboration*. O Reilly, 2009.

[69] Travis Swicegood. *Pragmatic Version Control Using Git*. Pragmatic Bookshelf, 2008.

[70] T. M. H. Reenskaug and J. O. Coplien. The DCI architecture: A new vision of object-oriented programming. `http://www.artima.com/articles/dci_vision.html`, March 20, 2009.

[71] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[72] Walter F. Tichy. Rcs -a system for version control. *Softw. Pract. Exper.*, 15(7):637–654, July 1985.

[73] Christoph Treude and Margaret-Anne Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 12–22, Washington, DC, USA, 2009. IEEE Computer Society.

[74] Aman ul haq and Muhammad Ali Babar. Tool support for automating architectural knowledge extraction. In *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, SHARK '09, pages 49–56, Washington, DC, USA, 2009. IEEE Computer Society.

[75] Hataichanok Unphon and Yvonne Dittrich. Software architecture awareness in long-term software product evolution. *Journal of Systems and Software*, 83(11):2211–2226, 2010.

[76] Jilles Van Gurp and Jan Bosch. Design erosion: problems and causes. *Journal of Systems and Software*, 61(2):105–119, 2002.

[77] Davor Čubranić and Gail C. Murphy. Hipikat: recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[78] Dennis M Wilkinson and Bernardo A Huberman. Cooperation and quality in wikipedia. *Proceedings of the 2007 international symposium on Wikis WikiSym 07*, pages 157–164, 2007.

[79] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.

Index

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

AK: Architectural Knowledge

AKM: Architecture Knowledge Management

Architectural knowledge: Architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is [41].

Architecture knowledge management: The way architectural knowledge is acquired and shared within an organization.

Architectural degradation: Architectural degradation occurs when the descriptive architecture diverges from the prescriptive architecture.

Architecture description language: A language to capture design decisions.

Context-sensitivity: The ability for ArchWiki to react to the context that the user is in.

Descriptive Architecture: The design decisions that are realized in the implementation.

Network effect: The value of a product or service is dependent of the number of others using it.

Prescriptive architecture: The design decisions that reflect the software architect's intent.

Software architecture: A software system's architecture is the set of principal design decisions made about the system.

Traceability: The codification of relationships between homogeneous and heterogeneous software development artifacts.

Appendix B

Questionnaire

Introduction

Software architecture is a relatively new term in the software development industry. It is often associated with a blueprint, as in a UML diagram, to guide the development process. The creation of this blueprint is often done by a single person, the software architect, without a real collaboration process with other stakeholders, such as developers, maintainers, testers and managers.

Recently, the view on software architecture has shifted to something that is done by multiple stakeholders and during all phases of development. Also the definition of software architecture has broadened to not only include high level structure diagrams, but all other sorts of design decisions and rationale. Because in this view the software architecture is created and used by different stakeholders during the whole development process, it is important to be able to collaborate on the software architecture. However the current tools are somewhat lacking in support of this new view, especially in the collaboration department.

Luckily there has been a surge of collaboration on the internet with the advent of Web 2.0 applications such as Wikipedia, Google maps, YouTube and Twitter.

Our approach is to support this new view on software architecture with the use of Web 2.0 technologies. To evaluate this approach we have created a software architecture management tool called ArchWiki, which uses a number of Web 2.0 technologies to manage the software architecture.

Purposes and objectives

We have invited you to participate in this evaluation study to gain a deeper understanding of the current software architecture views and practices and how ArchWiki supports software architecture management.

Study structure

The total study will last for approximately 2 hours.

We have chosen to evaluate ArchWiki by presenting you with a tutorial of ArchWiki and gaining feedback on ArchWiki by presenting you with a questionnaire. The tutorial will be performed individually for about 1 hour. In this tutorial you will use ArchWiki to create the software architecture of a small mock client-server application, during which you will use some features currently available in ArchWiki. This tutorial is only used to acquaint you with the features of ArchWiki and therefore files resulting from the tutorial will not be kept. You are free to ask any questions about ArchWiki to the researcher when performing the tutorial.

After the tutorial we will present you with a questionnaire about your views on and practices of software architecture and a questionnaire about our approach on software architecture management and ArchWiki. The questions will be discussed one by one with the other participants in a group discussion, but the questionnaire will be filled in individually. The questionnaires and group discussion will take another hour.

During this study your voice will be recorded. The information gathered during this study will be presented anonymously and is expected to be used for a master's thesis.

Thank you for participating in this study.

Software architecture views and practices Questionnaire

1. **Name:** _____

Background information

2. **How many years have you been active in software development?** _____
3. **Do you work within a project team?** □ Yes □ No
4. **What is your position within the project team?** _____
5. **Does your team have a software architect?** □ Yes □ No
6. **What does the software architect do?**

_____

_____

Software architecture consumption and program comprehension

7. **Program comprehension is very time consuming**
   strongly disagree □—□—□—□—□ strongly agree
8. **How do you analyze unknown software systems for program comprehension? (Top down, bottom up, directory structure, analysis tools, external documents, javadoc,etc.)**:

_____

_____

9. **Software architecture documents (Diagrams (UML), text documents etc.) help greatly with program comprehension**
   strongly disagree □—□—□—□—□ strongly agree
10. **Do you use software architecture documents (Diagrams (UML), text documents etc.) when available?**
    □ Yes □ No

I find the following items problematic with the consumption of software architecture documents

11a. **Locating the software architecture documents and relevant information within the documents**
   strongly disagree □—□—□—□—□ strongly agree

11b. **The use of different tools to process the architecture documents, source code and other documents**
   strongly disagree □—□—□—□—□ strongly agree

11c. **Lack of traceability between the software architecture documents and other software development artifacts such as source code and requirement documents**
   strongly disagree □—□—□—□—□ strongly agree

11d. **No confidence that the architecture documents are up to date and in sync with other development artifacts such as source code**
   strongly disagree □—□—□—□—□ strongly agree

11e. Low participation in software architecture documentation by other team members
strongly disagree □—□—□—□—□ strongly agree

11f. Other problematic items:

Documenting the software architecture

12. It is important to document the architecture once it is in your head
strongly disagree □—□—□—□—□ strongly agree

Documenting the software architecture is important...

13a. to act as a personal reminder of how the system is constructed
strongly disagree □—□—□—□—□ strongly agree

13b. to provide a high level abstract view of the system (as in (UML) diagrams)
strongly disagree □—□—□—□—□ strongly agree

13c. to document the design decisions and rationale
strongly disagree □—□—□—□—□ strongly agree

13d. to help project team members understand the system faster and to provide common concepts to assure everyone has the same mental picture of the software architecture
strongly disagree □—□—□—□—□ strongly agree

13e. to act as a blueprint and help achieve that the implementation stays as true as possible to the intended architecture
strongly disagree □—□—□—□—□ strongly agree

13f. Other important reasons:

14. How do you document the software architecture (comments in code, javadoc, (UML) diagrams, text diagrams, architecture discription languages (ADL))?

15. Is there a difference in phases within the project? Do you document more in the beginning than during the later stages?

I find the following items problematic when documenting software architecture

16a. Just forgetting to do
strongly disagree □—□—□—□—□ strongly agree

16b. Takes too much time
strongly disagree □—□—□—□—□ strongly agree

16c. Small contributions require disproportionate effort
strongly disagree □—□—□—□—□ strongly agree

16d. Partial contributions are not possible or undesirable
strongly disagree □—□—□—□—□ strongly agree

16e. Current tools are inadequate
strongly disagree □—□—□—□—□ strongly agree

16f. Other problematic items:

17. I am satisfied with the current means of documenting the architecture
strongly disagree □—□—□—□—□ strongly agree

Collaborating on software architecture

18. Collaborating on software architecture with other team members can help with the problems related with the use of software architecture
strongly disagree □—□—□—□—□ strongly agree

19. How do you collaborate on software architecture?

I find the following items problematic when collaborating on software architecture

20a. Sharing of the physical architecture documents with other team members is difficult
strongly disagree □—□—□—□—□ strongly agree

20b. No specific tools for collaborating on software architecture documents
strongly disagree □—□—□—□—□ strongly agree

20c. Team members are not aware when changes are made to the architecture documents and when the changes are of interest to them
strongly disagree □—□—□—□—□ strongly agree

20d. The participation of other team members with collaboration on software architecture is low
strongly disagree □—□—□—□—□ strongly agree

20e. Other problematic items:

21. I am more inclined to produce software architecture documentation if team members can be made aware of my contributions
strongly disagree □—□—□—□—□ strongly agree

ArchWiki Questionnaire

Web 2.0 and social media use

22. I use Web 2.0 and social media applications such as Wikipedia, Twitter, Facebook and Youtube
□ Yes □ No

23. Web 2.0 technologies can help in overcoming the problems related with the use of software architecture
strongly disagree □—□—□—□—□ strongly agree

24. Web 2.0 features lower the threshold to document the architecture
strongly disagree □—□—□—□—□ strongly agree

ArchWiki general impressions

25a. ArchWiki helps with program comprehension
strongly disagree □—□—□—□—□ strongly agree

25b. Give the top 3 features that help the most with program comprehension?

_____

_____

_____

26a. ArchWiki helps with documenting the architecture
strongly disagree □—□—□—□—□ strongly agree

26b. Give the top 3 features that help the most with documenting the architecture?

_____

_____

_____

27a. ArchWiki helps in keeping the architectural documentation up to date and in sync with other development artifacts such as source code
strongly disagree □—□—□—□—□ strongly agree

27b. Give the top 3 features that help the most with preventing architectural degradation

_____

_____

_____

28a. ArchWiki supports collaboration on software architecture
strongly disagree □—□—□—□—□ strongly agree

28b. Give the top 3 features that promote collaboration on software architecture the most

_____

_____

_____

29. What is your general impression about ArchWiki?

30. What do you think about the usability of ArchWiki?

31. What do you like the most of the ArchWiki tool?

32. What do you dislike the most of the ArchWiki tool?

33. What additional features would you like to see the most:

34. I would personally use ArchWiki
strongly disagree □—□—□—□—□ strongly agree

In-depth feature questions

Implementation mapping

35. This feature addresses the "traceability" issues of architectural software documentation
strongly disagree □—□—□—□—□ strongly agree

36. The mapping and the resulting context view lowers the threshold to document the architecture when making changes to the source code
strongly disagree □—□—□—□—□ strongly agree

37. The mapping helps keeping software architecture and source code synchronized and thus help prevent architectural degradation
strongly disagree □—□—□—□—□ strongly agree

38. I find the feature of linking source code to software architecture useful
strongly disagree □—□—□—□—□ strongly agree

39. This feature helps with program comprehension and documenting the architecture
strongly disagree □—□—□—□—□ strongly agree

Wiki

40. Using a Wiki is a good way to document the software architecture
strongly disagree □—□—□—□—□ strongly agree

41. Would you want to see Wiki's for other elements in the model? (Tags etc)
□ Yes , please specify which elements □ No

42. The use of a Wiki lowers the threshold to document the architecture
strongly disagree □—□—□—□—□ strongly agree

Tags

43. Tags created by me or others can help with program comprehension
strongly disagree □—□—□—□—□ strongly agree

44. Tagging is such a lightweight tool that it lowers the threshold to document the architecture
strongly disagree □—□—□—□—□ strongly agree

Bookmarks

45. Bookmarks can help with managing complexity of large architectural models
strongly disagree □—□—□—□—□ strongly agree

46. I like the feature to enable notifications based on the bookmarked elements
strongly disagree □—□—□—□—□ strongly agree

Notifications

47. I like to be notified of actions of other team members
strongly disagree □—□—□—□—□ strongly agree

48. I like the idea that other users are aware of my actions
strongly disagree □—□—□—□—□ strongly agree

49. I want to be able to filter the notifications
strongly disagree □—□—□—□—□ strongly agree

50. I like the currently built-in filters
strongly disagree □—□—□—□—□ strongly agree

51. What other filters would you like to see?

52. I like the use of an RSS feed to show the notifications
strongly disagree □—□—□—□—□ strongly agree