Experience Using the Web-Based Tool Wiki for Architecture Documentation

Felix Bachmann
Paulo Merson

September 2005

Software Architecture Technology Initiative

Technical Note
CMU/SEI-2005-TN-041

Contents

Acknowledgements ..... **vii**

Executive Summary ..... **ix**

Abstract ..... **xi**

1 Introduction **1**

2 Wiki **4**

3 Wiki for Architecture Documentation **6**

3.1 What to Expect from a Documentation Tool 6

3.2 Use of Wiki as an Architecture Documentation Tool 6

3.3 Experiences Using Wiki 8

3.3.1 R1: Support for Documents that Contain Text and Diagrams 8

3.3.2 R2: Support for Multiple Views with Elements and Relations 8

3.3.3 R3: Support for Version Control 9

3.3.4 R4: Support for Easy Access 9

3.3.5 R5: Low Cost 10

3.3.6 R6: Support for Change Requests 10

3.4 Other Issues that Need to be Considered 10

4 Pros and Cons of Wiki for Architecture Documentation **12**

5 Configuring Wiki For Architecture Documentation **19**

6 Conclusion and Future Work **22**

Appendix Screen Shots from Siemens Project Wiki **23**

References **30**

List of Figures

Figure 1: Transclusion of the Source “PO” into the Target Page “View PetstoreWeb” 17

Figure 2: Wiki Page of the Java Pet Store SAD Showing Navigation Aids 18

Figure 3: Wiki Page Showing Architecture Document’s Table of Contents 21

Figure 4: Basic Layout of a Wiki Page 24

Figure 5: A Wiki Page with Included Picture 25

Figure 6: Editing a Section of a Wiki Page 26

Figure 7: Displaying the Differences Between Two Versions of the Same Wiki Page 27

Figure 8: Uploading an Image File into Wiki 28

Figure 9: Training Materials Included 29

List of Tables

Table 1: Comparing Wiki to Word and CVS for Architecture Documentation....... 13

Acknowledgements

We greatly appreciate the contribution of Team Sapphire in the Master of Software Engineering (MSE) program at Carnegie Mellon® University, especially Min Chen, Bharat Gorantla, Okeno Palmer, and Lutz Wrage. They put significant effort into creating the wiki system and did not hesitate to provide the valuable information included in this report. We also want to thank Siemens Corporate Research, Inc.—especially Matthew Bass, Zakaria El Houda, and Neel Mullick—for providing the materials for this experience report and deep insights into the issues encountered using wiki. We are also grateful to Luis Maya in the MSE program, who used wiki for architecture documentation and provided us with important reflections.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Executive Summary

In an organization that uses an architecture-centric development approach, the purpose of the software architecture, especially the produced documentation, is to guide all the stakeholders that contribute in one way or another to the development of the product(s).

Unfortunately, in many organizations, this documentation ends up on the shelves, unused and collecting dust. This happens in part because it is difficult

- to keep the architecture documentation current
- for nondevelopers to understand what the documents describe
- for nondevelopers to use the tools necessary to access the documentation

In this technical note, we describe two distinct experiences that students in the Carnegie Mellon[®] University Master of Software Engineering (MSE) program had with a wiki[1]-based collaborative environment for creating architecture documentation. Wiki enabled the team to create and maintain architecture documentation collaboratively, because everyone with a Web browser could read and change information.

Using wiki as a communication tool for software architecture documentation is a promising but risky approach. One significant concern arises from the need to adjust the responsibilities in an organization. Wiki still lacks flexible and robust access management functionality, creating an acceptance barrier that might be difficult to overcome. Another limitation is that many organizations and individuals prefer to work with printed documents, but wiki pages are not suitable for printing.

Overall, the advantage of being able to create and maintain architecture documentation in a dynamic and collaborative way seems to outweigh any disadvantages. The wiki approach is an alternative to the most common solution for architecture documentation, which is the pairing of an editing tool (e.g., Microsoft Word) with a configuration management tool (e.g., Concurrent Versions System). The key benefits of a wiki-based approach are: (1) the documentation becomes more granular—the authors edit each wiki page separately, which also reduces contention; (2) the documentation is accessible via a Web browser from any machine connected to the network; and (3) wiki provides a mechanism called *transclusion* that avoids repetition of information.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 The name *wiki* is based on the Hawaiian term *wiki wiki*, meaning “quick” or “informal.”

A software architecture document is a living document. It changes as more details are added to the design over time. Therefore, the use of a tool that promotes changes in a collaborative environment seems to be a good fit. Future work will show how scalable the wiki approach is, how it may affect an evolving architecture documentation, and what it might offer for managing distributed software development.

Abstract

In an organization that uses an architecture-centric development approach, it is the purpose of the software architecture, especially the product documentation, to guide all stakeholders who contribute in one way or another to the development of the product(s). Unfortunately, in many organizations, this documentation ends up on the shelves, unused and collecting dust. This happens in part because it is difficult to keep the architecture documentation current, hard for nondevelopers to understand what the documents describe, and challenging for nondevelopers to use the tools necessary to access the documentation.

This technical note discusses the benefits and challenges of using a wiki-based collaborative environment to create software architecture documentation. The findings are based on two experiences. The first was that of a team of Carnegie Mellon® University Master of Software Engineering (MSE) program students that used the wiki tool in a real-world software project. For its customer, the team had to produce and document the architecture of a system that will be developed by many geographically distributed teams. The second experience was a study conducted by another MSE student to reconstruct and document the architecture of a multitier enterprise application using the wiki tool and UML 2.0.

1 Introduction

Let us start with a little story—a real story from the past that represents a recurring theme in many organizations that develop midsize to large software systems.

> At one time, an organization that one of the authors of this technical note worked for started a project to develop a system in the telecommunications domain: a big software system with millions of lines of code. We and our management clearly knew that we could be successful only if we had the "right" architecture to control the development of the system and to distribute the work to more than 150 developers in different geographical locations. We got the job to create and describe the architecture that would make the product successful. We also were given the time, money, and right people for the job.
>
> We used a modeling tool to create the architecture. Early on, we discovered that we also had to create normal text documents for our own review and to show progress to management. Therefore, we produced the textual description using a text-editing tool and pasted the pictures from the modeling tool into that document.
>
> After about half a year of hard work, seemingly endless discussion, and review, we finished the documentation. We happily distributed our four volumes of materials to the next group of developers, so that they could start implementing the system. We all were exhausted but believed we had done a marvelous job. Everything was thought through and documented. The nights and weekends belonged to us again!
>
> What a surprise for us when the product managers showed up and asked for information—such as a roadmap for delivering features, estimates of how many people would be needed for the implementation, and so on. We wondered, "Didn't they read the documentation we gave them?" Everything was described there, at least to the degree that a product manager should be able to estimate the rest. They started complaining that it was too much to read, and they didn't understand those pictures anyway.
>
> It looked like we had more documentation work to do. But at least the developers had everything they needed—or so we thought. After a couple of weeks, we checked on how the development work was going. And yes, all the developers had our documentation available, nicely stacked on their shelves. They began to ask questions about aspects of the software that

> clearly had been documented. We responded to their questions with our own questions like, “Haven’t you read Section 4 in Volume II?” We discovered that no one really had looked at the documentation; it was put on a shelf the first day it was received and remained there.

Does that story sound familiar to you? If so, you should continue reading.

In an organization that uses an architecture-centric development approach, the architecture becomes a major communication tool for explaining to stakeholders the design decisions made and the consequences of those decisions. In particular, the architecture is the blueprint for implementation and has to be effectively communicated to the developers.

Architecture-centric development involves iteratively

- creating the business case for the system
- understanding the requirements
- creating or selecting the software architecture
- documenting and communicating the software architecture
- analyzing or evaluating the software architecture
- implementing the system based on the software architecture
- ensuring that the implementation conforms to the software architecture [Kazman 04]

Architecture documentation is created to facilitate this communication between stakeholders and used to plan the iterations. Documentation becomes especially important when more than a few people are involved in the software system development, when the person implementing the system is not the one who created the architecture, or when the development teams are geographically distributed.

It is still a common approach to produce documentation using an editing tool for text and a modeling or drawing tool for diagrams. Unfortunately, this documentation typically ends up on the shelves, collecting dust, after being used for an initial period to provide some insights into the architecture. Here are four good reasons why this happens:

1. It is very difficult to keep the architecture documentation current in a world of ever-changing requirements, especially for a large system. The architect(s) may not even get all the information about changes and new requirements. If stakeholders (e.g., developers) find just one thing out of date in the documentation, they’ll consider all of the documentation unreliable.
2. The available tool support may help architects design and evolve an architecture, but those tools are not widely accepted (and usable) by stakeholders other than developers. Even for developers, those tools might be cumbersome to use because they are not integrated into the implementation environment.
3. The process for changing the architecture (change control boards) might be too slow for developers. Once the implementation is in place, it is just easier to change the code;

changing the architecture is seen as overhead. Very often, code changes due to bug fixes, improvements, and refactorings are not reflected back into the architecture documentation.

4. The documentation is not effective because
    - The writers made too many assumptions about what the readers know.
    - The diagrams can't be understood because the notation is ambiguous.
    - The text is too verbose and repetitive.
    - The document is poorly structured and difficult to navigate through.

In this technical note, we describe two distinct experiences that students in the Carnegie Mellon® University Master of Software Engineering (MSE) program had using the Web-based documentation tool wiki to create architecture documentation. In the first experience, a team of four students got the job from its customer, Siemens Corporate Research, Inc., to develop an architecture for a system that will be implemented by teams at geographically dispersed locations. Wiki enabled the team to work collaboratively on the architecture documents and provide its customer with user-friendly access to the documentation. Wiki helped the team react to changing requirements and develop documentation that was useful to the customer—the top two problems mentioned above.

In the second experience, a student used wiki to record and organize multiple views of an architecture reconstructed from a multitier application—the Java Pet Store application [Singh 02].

In Section 2, we give an overview of the most important features of wiki as a tool to record technical documentation in a collaborative setting. We then describe in Section 3 how wiki can be used for producing and communicating architecture documentation and discuss the experiences of the students. Section 4 provides a point-by-point comparison of wiki and a Microsoft Word/Concurrent Versions System (CVS) combination. Section 5 has some recommendations for configuring wiki for software architecture documentation. In Section 6, we conclude this technical note by discussing further work needed to make a wiki-based architecture documentation approach more successful.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

2 Wiki

What is wiki? Wiki is in Cunningham's original description, *the simplest online database that could possibly work* [Leuf 01]. Wiki allows users to freely create and edit Web page content using any Web browser. According to *Wikipedia*, the first wiki Web site (March 25, 1995) was the WikiWikiWeb, part of the Portland Pattern Repository [Wikipedia 06a].

The power of wiki comes from its simplicity. Everyone who can use a Web browser and fill out Web-based forms can view **and** edit the content of a wiki page. Wiki supports hyperlinks and has a simple text syntax for creating new pages and links between pages "on the fly." If you can suppress your desire for fancy formatting,[2] Wiki is a fast-to-learn, easy-to-use, and intuitive editing environment. It allows novice users to produce fairly nice-looking Web pages that are immediately available to all other users.

Wiki also allows the reorganizing of content. Pages can be reordered, and new pages can be created to show the existing content in a different order.

So far, we've only described a pretty unsophisticated text-editing tool. So, what is the big deal with wiki?

While it's a simple concept, "open editing" has some subtle yet profound effects on wiki usage. By allowing everyday users to create and edit any page in a Web site, wiki encourages democratic use of the site and promotes content composition by nontechnical users. Together, those two factors lead to a paradigm shift in creating and maintaining documents. Because every user can (and is encouraged to) contribute, the workload is distributed—an effect that is typically seen as very positive. Also, because wiki encourages democratic use, every user is treated equally. The concepts of author (one who creates and maintains the document) and reader (one who only reads the document) don't really exist with wiki. Any reader can change the content of any document. Many organizations do not see this as a benefit, because it runs counter to the established view of document ownership.

To support discussion when two or more people make changes to the same page, wiki tracks changes at the page level. For every page, it is easy to look up earlier versions, display the differences between two versions, and revert to an older version. When anyone makes changes to a page, everyone can see what was changed.

A by-product of wiki is that all content is kept in a relational database, which is required for wiki to generate edited pages dynamically. Standard SQL-based database tools can be

[2] Wiki supports many kinds of formatting, such as creating tables, using special fonts, and using a rich set of HTML tags. But the more such features are used, the more complicated it becomes for a novice user to change anything.

employed to create all kind of queries, such as listing all changes performed by a given user in a period of time or extracting some information to be used at different sites or in other tools.

3 Wiki for Architecture Documentation

In this section, we discuss the experiences of the MSE student team with wiki. The screenshots mentioned throughout this chapter are shown in the appendix.

3.1 What to Expect from a Documentation Tool

To understand if any tool, not just wiki, is appropriate for creating and maintaining software architecture documentation, we need first to understand some important requirements for such a tool:

R1 Architecture documentation typically contains lots of text and diagrams to give an overview of the topic presented. A tool, therefore, has to support text editing and drawing functionalities.

R2 Architecture documentation is structured into views, and every view contains elements and relations. An architecture documentation tool should understand and support those concepts and therefore support managing views with elements and their relations.

R3 Typically the architecture documentation has more than one author; therefore, version control and merging capabilities would be helpful.

R4 The document(s) will be read by a diverse group of stakeholders, requiring the documentation artifacts to be easy to access and navigate.

R5 Because many stakeholders are potentially interested in the documentation, the costs for the tool environment and system administration should be kept to a minimum.

R6 Architecture documentation will evolve over time, and requests for changes will come from all possible sources. A quick turnaround of those change requests ensures that documentation will be kept current.

Today, it is very difficult to find a tool that fully supports all of those requirements for creating architecture documentation. In many development organizations, the architect has to use multiple tools, often a standard text-editing tool combined with a modeling or drawing tool.

3.2 Use of Wiki as an Architecture Documentation Tool

Prepared using a Web-based documentation tool, a document can typically be structured as linked pages—Web pages to be precise. Compared with documents written with a text-editing tool, Web-oriented documents typically consist of short pages (created to fit on one screen) with a deeper structure. One page usually provides some overview information and has links to more detailed information. When done well, a Web-based document is easier to use for people who just need to have some overview information. On the other hand, it can become more difficult for people who need detail. Finding information can be more difficult in multi-page, Web-based documents than in a single-file, text-based document, unless a search engine is available.

Using wiki as a Web-based documentation tool, we can see some solutions for the given requirements:

R1 **Support for documents that contain text and diagrams.** Web pages can contain text and multimedia; this capability should be sufficient for displaying architecture documentation. Wiki, though, does not offer drawing functionality such as creating UML or other diagrams. In addition, the fact that the documentation is structured as linked pages is adequate for browsing but not for producing a printed version.

R2 **Support for multiple views with elements and relations.** Views in wiki can be created as a set of pages that are linked to one another. It is also easy to create an overview page to help stakeholders select views. It would even be possible for users to create something like their own “bookshelf” with links to their favorite pages. Because wiki is a generic Web-editing tool, however, there is no support for creating specific types of views that contain specialized elements (e.g., classes, servers, tiers, queues) and relations (e.g., uses, is a, part whole).

R3 **Support for version control.** Wiki has version control for every page and provides the capability to show the differences between versions. Further, a page can be structured into sections, and any section can be edited. This capability to segment pages should minimize possible concurrent changes to the same document, supporting different team members editing the same document or even the same page. Although wiki supports collaboration, it has limited support for merging changes, baselining pages, or managing sets of pages.

R4 **Support for easy access.** The Web-based pages created with wiki should be easy for most stakeholders to read. For developers who are interested in seeing all the details in a single place, a wiki might be more difficult to use.

R5 **Low cost.** Wiki requires a server connected to an intranet or the Internet. In addition to network connectivity and the wiki software itself, these elements are required: a Web server (e.g., Apache HTTP Server), a relational database server (e.g., MySQL), and PHP. It is possible to install a complete solution using solely free and open source software. On the server side, the cost of installing and maintaining wiki is related to the administration of the server. Administration involves executing backups, installing version upgrades, setting user rights, customizing functionality, and configuring localization. In practice, little maintenance is required after installation. On the client side, there is no additional cost for the users of wiki, since the only tool they need is a Web browser. Therefore, the relative cost of wiki is inversely proportional to the number of users.

R6 **Support for change requests.** Wiki allows everyone to change the content. When developers discover a problem in the document, they can correct it immediately. Anyone interested in changes to a particular part of the documentation can display what was changed, when, and by whom.

When we rate wiki against the list of architecture documentation requirements, we conclude that wiki offers a pretty good solution.

3.3 Experiences Using Wiki

How did the team of students that used wiki determine whether it met their requirements and expectations? What pitfalls did they discover? What lessons from their experience can be applied to avoid problems in the future? To examine questions like those, we'll discuss the team's experiences with wiki in each of the six requirements.

3.3.1 R1: Support for Documents that Contain Text and Diagrams

As would be expected, the team had no problem creating text-based pages. Those pages were easy to create and, in most places, did not use fancy formatting. The editing interface is easy to use and the syntax for basic text formatting is very intuitive (see Figure 6 on page 26).

However, dealing with the architecture diagrams turned out to be very cumbersome. Wiki allows the uploading and inclusion of images in formats such as JPEG and GIF (see Figure 8 on page 28). Consequently, the students had to use a drawing tool to create diagrams, convert diagrams into images, and upload the images—in order to add diagrams to the text. Since this process had to be repeated for every new version of a diagram, it created a maintenance problem, especially early on when architecture diagrams changed very frequently.

As a result, too, the original diagram remained in the drawing tool where only the student team members could access it. This had a negative side effect on the capability for any stakeholder to respond quickly to change requests (Requirement 6). No one other than a student team member was able to make any text change that would also involve modifying a diagram—a situation which happened frequently. This inability of the wiki tool to deal with drawing is most likely its biggest drawback.

3.3.2 R2: Support for Multiple Views with Elements and Relations

To create views, the team structured the document so that the different views were on different pages (see chapters 4.1 to 4.3 in the Table of Contents shown in Figure 4 on page 24). This was an easy task that did not cause problems. The student team members did not create their own "bookshelf" or see the inability of wiki to understand architecture elements as a drawback. Their assessment was based on their recognition that text-editing and drawing tools—alternatives to wiki for describing the architecture—do not have any knowledge about software architectures.

3.3.3 R3: Support for Version Control

The wiki feature for version control and display of differences (see Figure 7 on page 27) was welcomed by the team at the beginning. After a while, though, the team encountered problems based on the dynamicity of wiki. It is just too easy to change a page. Whenever a team member responsible for a page checked for changes since the previous version, that team member had to check multiple previous versions, which made it very difficult to check for consistency.

To overcome this difficulty, the team started using another wiki feature—the discussion page. Every page in wiki also has a discussion page where team members can provide feedback without changing the page. In the beginning, this seemed to be a nice feature. But its use uncovered a further problem. A discussion page is not coupled with a page version. When reviewing comments, a team member could tell neither which version of the page was being discussed nor whether the comment was current or old. The team overcame this weakness by mentioning the page version number in their comments as a standard practice.

The lesson learned here is this: while page content is immature, letting everyone make changes to pages is not a good idea. At that stage, instead of allowing anyone to change pages directly, use the discussion page to describe the changes to be made. Later in the process, when fewer changes are expected, anyone can change the page directly without using the discussion page.

3.3.4 R4: Support for Easy Access

The team's architecture documentation has a fairly flat structure. Most views are represented in a few pages that typically do not feature links to other pages. Therefore, the team members and the customer, Siemens Corporate Research, Inc., can easily navigate through the document. It is realistic, of course, that the structure will become more complex as the amount of information grows. To date, we have no insights regarding which navigational features to use or how to use them in substantive architecture documentation. It is possible to create a sidebar with a partial or complete map of the wiki that looks like a table of contents with links. This sidebar can be displayed on the right-hand side of the screen for every page (see Figure 2 on page 18). Although it helps navigation tremendously, this feature is not added automatically when a wiki page is created, and the team did not implement a sidebar table of contents for its documentation.

A bigger problem was caused by the dynamic nature of wiki. The expectation is that everyone actually reads the information online to get the latest information. Wiki is an online tool, after all. There are some cases where this advantage is also a limitation. For example, the team held many teleconferences with its customer. In the team's conference room, however, there was no network connection. As a result, some participants were not able to discuss the online information. Instead, those participants printed information some days prior to a teleconference. By the time of the meeting, the information in the wiki pages had been changed, causing confusion as participants referenced different versions.

To avoid this problem, the definition of a baseline is required—an option which is not supported by wiki version-control functionality.

3.3.5 R5: Low Cost

The student team and customer were able to use the infrastructure with no problems and at very little cost. The tool is robust and did not crash. One incident, though, showed the strong dependency on a working infrastructure. For a couple of days, the team lost its server access and, with it, the ability to work on the documents. This problem is not specific to wiki; it applies to all server-based infrastructures (such as Web or mail servers) and can be solved with the usual measures, such as providing redundancy. But it shows that a working infrastructure is critical.

3.3.6 R6: Support for Change Requests

The student team did set up a feedback capability with its customer, allowing reviewers to comment on or change pages directly. Interestingly, the wiki-based feedback never happened. The customer provided feedback in emails and phone conferences. A feature that was intended to improve the communication between stakeholders was not used.

After interviewing the customer, it became obvious that the customer and student team had different expectations for use of the feedback mechanism. Although the student team allowed the changes, the customer’s understanding was that feedback would be provided as comments, not necessarily as direct changes. Wiki changes this traditional feedback paradigm, and, therefore, the users have to be reeducated.

Learning from the documentation development experience, the customer created video-based training materials when it took over the wiki infrastructure (see Figure 9 on page 29). These materials were designed to help new users understand and fully use the power of wiki.

3.4 Other Issues that Need to be Considered

Many different implementations of wiki are available. Some support smaller projects; others are well suited for large documentation projects. Some have very fancy formatting capabilities; others only offer basic ones. Some offer the ability to add functionality; others don’t. An organization has to understand its needs to be able to select the appropriate version.

Wiki provides an open environment, in which everyone who can access the site can read and change anything. In many cases, such an environment is not desirable. Organizations typically make a clear distinction on permissions, especially when it comes to subcontractors. Subcontractors should see and change only those parts assigned to them, along with some context information. The remainder of the information should be hidden from them. Wiki does not have a flexible permission system. As a result, it is not possible to show partial content. If permission-based access is really a must, a second wiki would have to be set up with scripts synchronizing the two sites.

It also can be very cumbersome to produce printouts for those who need to see the information but do not or cannot use wiki. Some wiki versions may offer limited scripting abilities that can be used to produce printouts. That scripting requires the user to identify a set of pages to be included and then to print that set. Be warned, though: if the structure of the documentation changes, the scripting has to be changed, too.

4 Pros and Cons of Wiki for Architecture Documentation

A software architecture document (SAD) consists of sections of prose intermingled with tables and figures. The figures are typically diagrams that follow a standard design notation (e.g., UML) or some informal notation with boxes and lines. The diagrams illustrate the different perspectives of the architecture, including the structure of implementation units, the hardware infrastructure, and the structure of runtime elements and their interactions. The prose and tables complement the diagrams to provide a description of the elements and relations depicted graphically—as well as other information, such as a system overview, architecture background, design rationale, mapping to requirements, and glossary.

Most SADs today are created using Microsoft Word. Typically, the diagrams are created externally and embedded in the Word document. Design diagrams are often created with a modeling tool, such as IBM Rational Software Architect or Omondo EclipseUML, but many times the architect simply uses a drawing tool such as Microsoft Visio or PowerPoint. An alternative to using a modeling or drawing tool that is now common in agile projects is creating and discussing diagrams on a whiteboard and then taking a digital picture of the board. In any case, diagrams are converted to an image format that is then inserted into the Word document. That document itself is usually stored in a configuration management tool (e.g., CVS), so that many people can access it and a revision and version-control process can be enforced.

Wiki offers an alternative to using an editing tool paired with a configuration management tool. Wiki, however, is not an alternative to modeling or drawing tools. Table 1 delineates the positive and negative aspects of using wiki versus Word and CVS for software architecture documentation. In the table, we've added graphic representations of the prose comments by inserting stylized icons:

- A smiling face icon (☺) symbolizes that the tool meets the need well.
- A frowning face icon (☹) means that the tool falls short of meeting the need.
- A plain face icon (😐) signifies that the tool might meet the needs of some users.

Table 1: Comparing Wiki to Word and CVS for Architecture Documentation

Feature	Word and CVS	Wiki
Granularity and concurrent changes	☹ Typically, the SAD would be contained in one document or a few documents. If multiple authors need to edit the SAD, there will be some contention. CVS allows different users to check out and commit changes to a Word document. However, Word documents are treated as binary files and cannot be compared or merged using CVS/diff tools. A user should lock the document when editing it.	☺ A wiki-based SAD[3] is typically much more granular and hence more suitable for documentation created collaboratively. When two or more authors edit the same page at the same time, the wiki server will try to merge the changes. In any case, a user may manually lock a page prior to editing it.
Repeating information in multiple places[4]	☹ If repetition is avoided by adding a cross-reference (e.g., “See Section X”) or hyperlink, the document is less readable because the user has to flip pages or follow links too often. ☹ If the information is copied to multiple places, the reader doesn’t have to flip pages, but the document becomes harder to maintain.	☺ Wiki provides a mechanism called transclusion that solves the problem caused by the repetition of information (see Figure 1 on page 17). Transclusion allows the embedding of a piece of text in different pages. When the text is modified in the source page, all target pages are updated. This is a major benefit of wiki over Word and other editing tools.
Working offline	☺ The user can edit documents regardless of network connectivity. However, to get the latest version or commit a new version to the repository, the user must establish a connection to the CVS server.	☹ Without a connection to the wiki server, it is not possible to read or change a wiki page.
User deployment	😐 In addition to Word, the user has to install a CVS client and configure access to the CVS server on the machine to be used to access the documentation.	☺ Nothing besides a Web browser is required on the user’s machine. Therefore the user can access the wiki from any computer that has network connectivity.

3 For this comparison, a wiki was created using MediaWiki. For more on MediaWiki, see http://en.wikipedia.org/wiki/MediaWiki [Wikipedia 06b].

4 Repeating content in a document should be avoided because it makes effecting changes more difficult. However, many times a piece of information—for example, the description of a key component of the architecture—is useful to the reader in many places and ideally should be visible in all these places.

Table 1: Comparing Wiki to Word and CVS for Architecture Documentation (cont.)

Feature	Word and CVS	Wiki
Document template[5]	☺ In Word, a user can create a document template (that has the extension ".dot") for the SAD; the template can be instantiated easily.	☹ There is no mechanism similar to the template for wiki. It is not possible to instantiate one wiki page based on the structure of another.
Search	😐 A user can press **Ctrl+F** to locate words or phrases in the document, but this Find feature doesn't work if the SAD spans several documents.	☺ A user can press **Ctrl+F** (Web browser option) to locate words or phrases in the current page. In addition, a search box is available on every page that allows searching the entire wiki.
Navigation	☺ If the SAD is a single document, Word can create a table of contents (TOC) based on the section and subsection headings. In addition, links to sections, tables, and figures can be added easily to the text. ☺ The user can select "View | Document Map" to see a TOC, from which the user can go to any section of the document. The vertical scroll bar also shows section headers and page numbers as the user slides the marker. 😐 If the SAD spans several documents, the user can create a master document and subdocuments. Even so, the only navigation aids are a centralized TOC and links from the master to the subdocuments.	☺ A TOC of the current page is created automatically for each page (see Figure 2 on page 18). Also, it's possible to configure a TOC of the entire SAD and manually add it to every page using transclusion (see Figure 1 on page 17 and Figure 2). ☺ In addition, wiki automatically creates a navigation menu (left-hand side of Figure 2), and it is very easy to create links from page to page. For instance, in the related views section of an architecture view, the user can create a link to page *View Xyz* by simply typing `[[View Xyz]]`. ☺ The user can also type the name of a page in the search box and click the **Go** button to go directly to that page.
Review	☺ The *Track Changes* option in Word allows visualization of edits in line with the document's text. It also indicates the author, date, and time of each change.	☹ There is no mechanism similar to *Track Changes*. The user can only add comments in the discussion page, which is available for each wiki page.

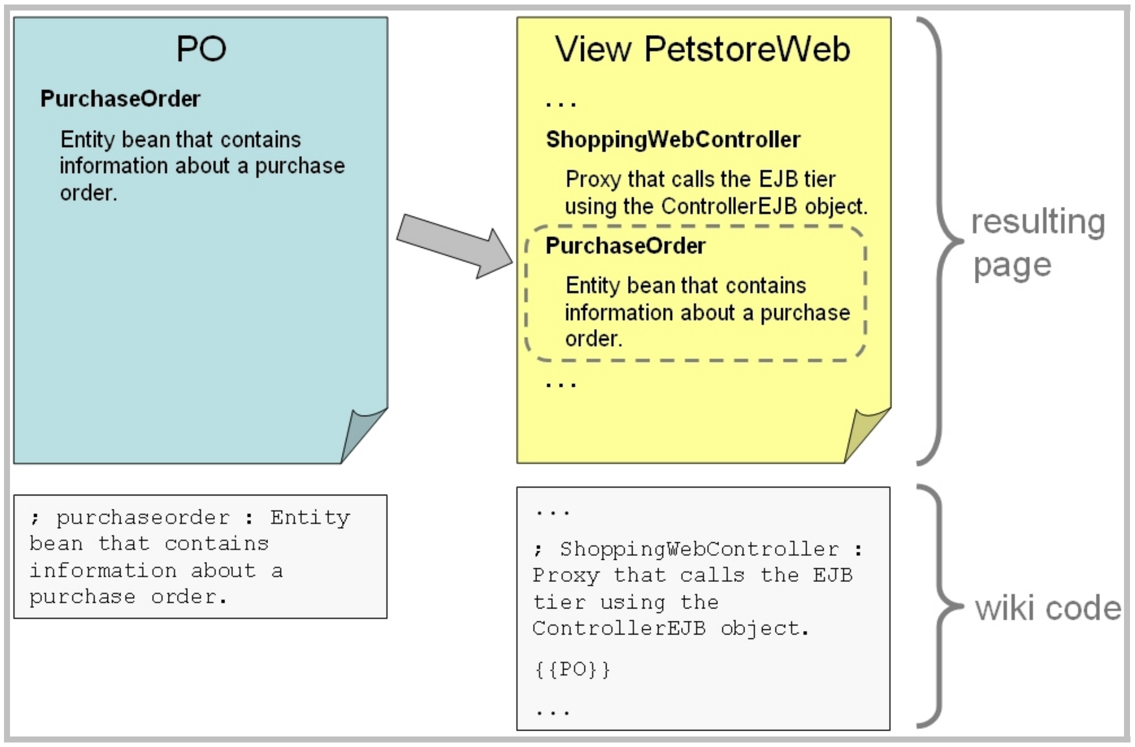
[5] SADs should follow a standard organization (i.e., a template). It's easier for the reader to navigate a familiar structure. Also, a template allows the writer to record information as soon as it is known and measure the work left to be done [Clements 02].

Table 1: Comparing Wiki to Word and CVS for Architecture Documentation (cont.)

Feature	**Word and CVS**	**Wiki**
History of changes	☹ CVS can display a history of all versions of the file, showing the author, date, and time of each version and comment. However, because Word is treated as a binary file, it is not possible to see what changed from one version to the next.	☺ On every page there is a tab named *history* that leads to a page where the user can see the author, date, and time of each change. The user can also compare any two versions in the history to see what has changed (see Figure 7 on page 27). ☹ Every time a page is saved, a history entry is created. As a consequence, there are usually many versions of a page that do not constitute major changes. A user can classify a change as minor or major when saving the page and minor changes can be omitted from the history by a manual process.
Notification of changes	☺ CVS provides a means to notify the users via email when artifacts are created, removed, or modified in the repository.	☺ On every wiki page there is a tab named *watch* that adds that page to the user's *watchlist*. At any time, the user can go to his or her *watchlist* page to see changes made to watched pages. The user can also opt to receive email notifications when a watched page is modified by someone else and when a new page is created.
Text formatting	☺ Very rich	😐 Limited to basic HTML formatting
WYSIWYG (what you see is what you get) editing	☺ Microsoft Word is a full-fledged WYSIWYG editor.	☹ Pages are edited in a regular text box. The author has to use wiki or HTML markup to format the text, create links, embed images, create tables, and so on. To see the result, the user has to click the **Preview** button. ☺ The user can add HTML WYSIWYG editors to MediaWiki manually [Wikipedia 06b].

Table 1: Comparing Wiki to Word and CVS for Architecture Documentation (cont.)

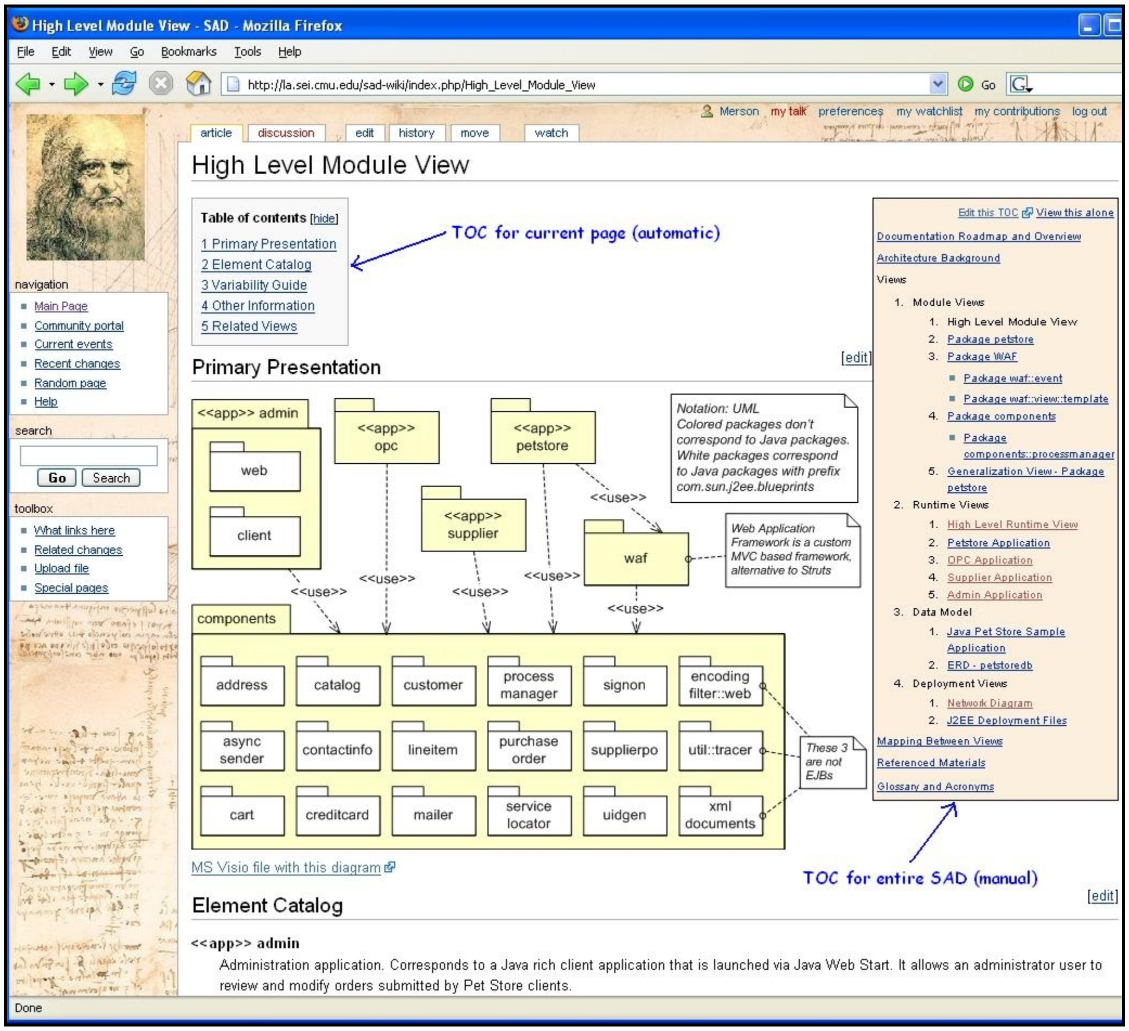
Feature	**Word and CVS**	**Wiki**
Printing	☺ The document is suitable for printing. The author can configure headers, footers, page numbering, and other characteristics of the printed pages.	☹ Not much can be configured when printing a Web page in a Web browser. Moreover, the documentation consists of several wiki pages. Scripts to print all pages need to be created and maintained manually.
Access control	☹ Access control in CVS is rudimentary. It requires user authentication, but authorization relies on permissions set in the file system where the repository files reside.	☹ Access control for MediaWiki is also very basic. User self-registration and authentication are available, but there are no means to set different permissions on separate pages to users or groups of users. The administrator can create a user group and assign permissions to that group, but the permissions apply to all pages. The administrator can also indicate what pages can be seen by users who are not logged in and whether these users can edit pages.
Documen-tation delivery	☺ In projects where the architecture documentation is a deliverable, the Word document is a simple artifact to be transferred to the contracting organization, either in soft or printed format.	☹ When a subcontracted organization hosts the wiki, the transfer of the architecture documentation is more complicated. The contracting organization has to set up a wiki infrastructure, and then the wiki database has to be exported and imported.


PO
PurchaseOrder
Entity bean that contains information about a purchase order.
View PetstoreWeb
. . .
ShoppingWebController
Proxy that calls the EJB tier using the ControllerEJB object.
PurchaseOrder
Entity bean that contains information about a purchase order.
. . .
resulting page
wiki code

```
; purchaseorder : Entity
bean that contains
information about a
purchase order.
```

```
...
; ShoppingWebController :
Proxy that calls the EJB
tier using the
ControllerEJB object.
{{PO}}
...
```

Figure 1: Transclusion of the Source "PO" into the Target Page "View PetstoreWeb"



High Level Module View

Table of contents [hide]

1 Primary Presentation
2 Element Catalog
3 Variability Guide
4 Other Information
5 Related Views

Edit this TOC View this alone
Documentation Roadmap and Overview
Architecture Background
Views
1. Module Views
   1. High Level Module View
   2. Package petstore
   3. Package WAF
      - Package waf::event
      - Package waf::view::template
   4. Package components
      - Package components::processmanager
   5. Generalization View - Package petstore
2. Runtime Views
   1. High Level Runtime View
   2. Petstore Application
   3. OPC Application
   4. Supplier Application
   5. Admin Application
3. Data Model
   1. Java Pet Store Sample Application
   2. ERD - petstoredb
4. Deployment Views
   1. Network Diagram
   2. J2EE Deployment Files
Mapping Between Views
Referenced Materials
Glossary and Acronyms

Primary Presentation [edit]

Notation: UML
Colored packages don't correspond to Java packages. White packages correspond to Java packages with prefix com.sun.j2ee.blueprints

Web Application Framework is a custom MVC based framework, alternative to Struts

These 3 are not EJBs

MS Visio file with this diagram

Element Catalog [edit]

<<app>> admin

Administration application. Corresponds to a Java rich client application that is launched via Java Web Start. It allows an administrator user to review and modify orders submitted by Pet Store clients.

Figure 2: Wiki Page of the Java Pet Store SAD Showing Navigation Aids

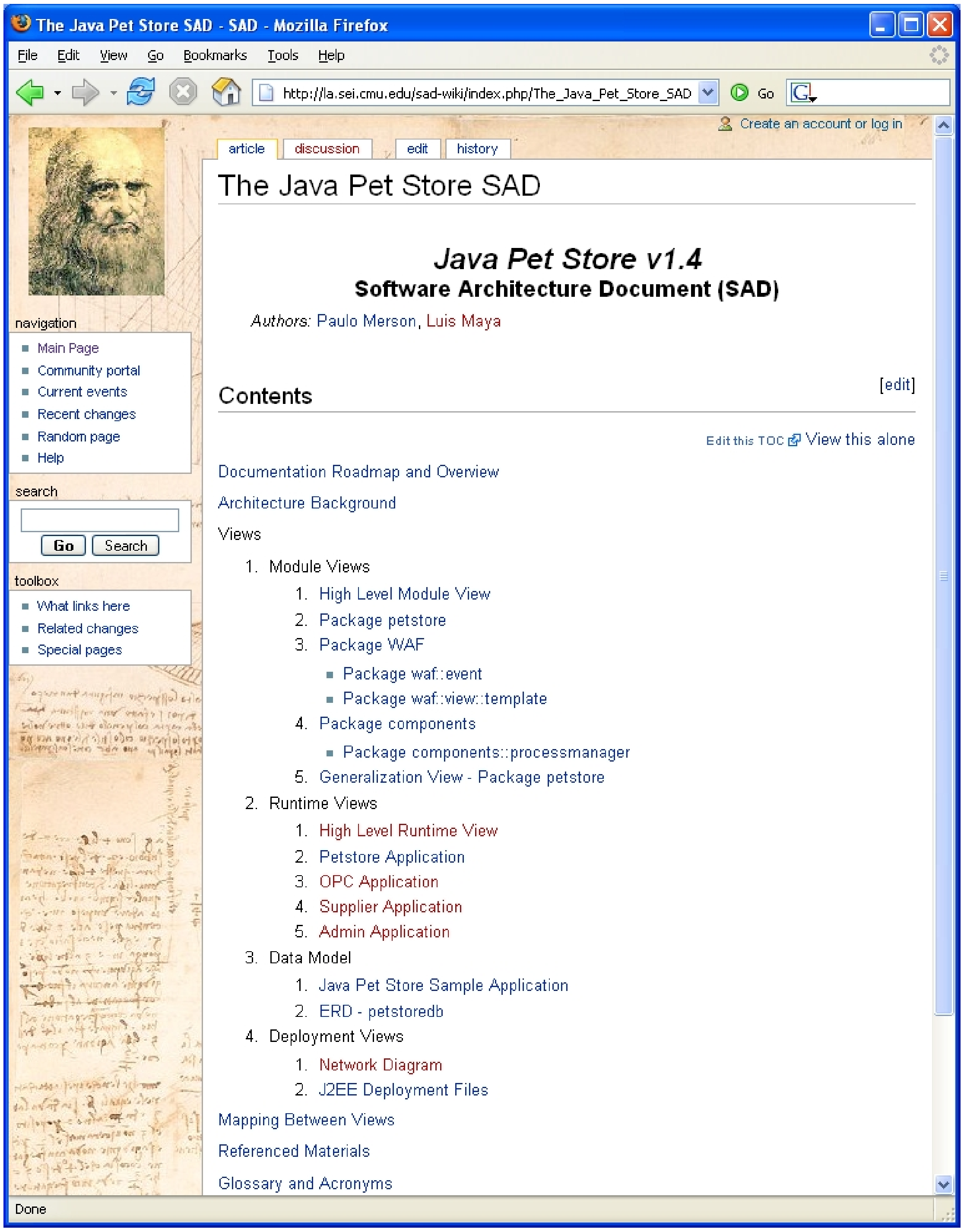
5 Configuring Wiki For Architecture Documentation

If you are going to use a wiki as the repository of your software architecture, there are some practical considerations and guidelines that may help. Below is a list of recommendations for the configuration and day-by-day use of your wiki-based SAD.

- The first step obviously is to create a new wiki or define a page for the SAD in an existing wiki. Although it is not possible to automatically enforce a specific structure for the new wiki, it is highly advisable to follow a standard organization—that is, a template.
- Create the initial page of the architecture documentation as a list of links to the main topics (see Figure 3). To facilitate navigation, you can manually add the main table of contents to every page of the documentation (see the right-hand panel in Figure 2). That is done using transclusion: insert the transcluded text into a table that occupies only a portion of the screen width (see Figure 1).
- Create one wiki page for each architecture view. Follow a convention to name the views, so that it is easy to remember the names when creating links (the view and its wiki page should share the same name).
- Create one wiki page for each mapping between views, so that each mapping can be edited independently.
- If you are using a drawing tool, such as Visio or PowerPoint, create one file for each diagram or one file for each architecture view. Give the file the same name as the view, replacing spaces with a standard character. For example, the diagrams used in the *Package petstore* view would be stored in Package_petstore.vsd. If you are not using a configuration management tool, such as CVS, to store the drawing tool file, here is an alternative: upload the file to the wiki and create a link to it in the wiki page that contains the corresponding architecture view.
- Diagrams need to be converted to an image format (e.g., JPEG) before they can be added to wiki pages. Give the image file the same name as the drawing tool file (use different suffixes in the name if the drawing tool file contains more than one diagram). For example, the diagrams in Package_petstore.vsd could be exported to
  - Package_petstore.jpg: structural view (primary presentation)
  - Package_petstoreSD1.jpg: first sequence diagram
  - Package_petstoreSD2.jpg: another sequence diagram
- Wiki does not provide an editorial feature similar to the *Track Changes* option in Word. The wiki option is to add comments to the discussion page. An alternative that has proven to be effective when reviewing a wiki page is this process:
  1. Copy the wiki page to a blank Word document.

2. Activate the *Track Changes* option.
3. Edit the Word document and add comments as needed.
4. Send the Word document to the author of the wiki page, who then can change the wiki page based on the edits and comments in the review.

- It is very common for an element in the architecture to appear in more than one view. Create the description of that element in a separate page and use transclusion in the element catalog of all pages that contain that element.
- If you already have documentation created in Word and want to migrate it to a wiki, there are macros/scripts that can help. To find them, go to http://www.google.com and type in “Word2Wiki” or “WordToWiki.”



The Java Pet Store SAD

Java Pet Store v1.4

Software Architecture Document (SAD)

Authors: Paulo Merson, Luis Maya

Contents [edit]

Edit this TOC View this alone

Documentation Roadmap and Overview

Architecture Background

Views

1. Module Views
   1. High Level Module View
   2. Package petstore
   3. Package WAF
      - Package waf::event
      - Package waf::view::template
   4. Package components
      - Package components::processmanager
   5. Generalization View - Package petstore
2. Runtime Views
   1. High Level Runtime View
   2. Petstore Application
   3. OPC Application
   4. Supplier Application
   5. Admin Application
3. Data Model
   1. Java Pet Store Sample Application
   2. ERD - petstoredb
4. Deployment Views
   1. Network Diagram
   2. J2EE Deployment Files

Mapping Between Views

Referenced Materials

Glossary and Acronyms

Figure 3: Wiki Page Showing Architecture Document's Table of Contents

6 Conclusion and Future Work

Using a collaborative tool such as wiki as a communication tool for software architecture documentation is promising. But this use of wiki still requires many workarounds—a circumstance that may prevent organizations from adopting a wiki-based approach. In addition, the need to adjust responsibilities in an organization and the lack of a permission system create acceptance barriers that also might be difficult to overcome.

However, the advantage gained by working collaboratively (at least in a small team) to create architecture documentation seems to outweigh those disadvantages. When asked whether they would use wiki again—knowing all its disadvantages—the student team members answered without hesitance, “Yes.”

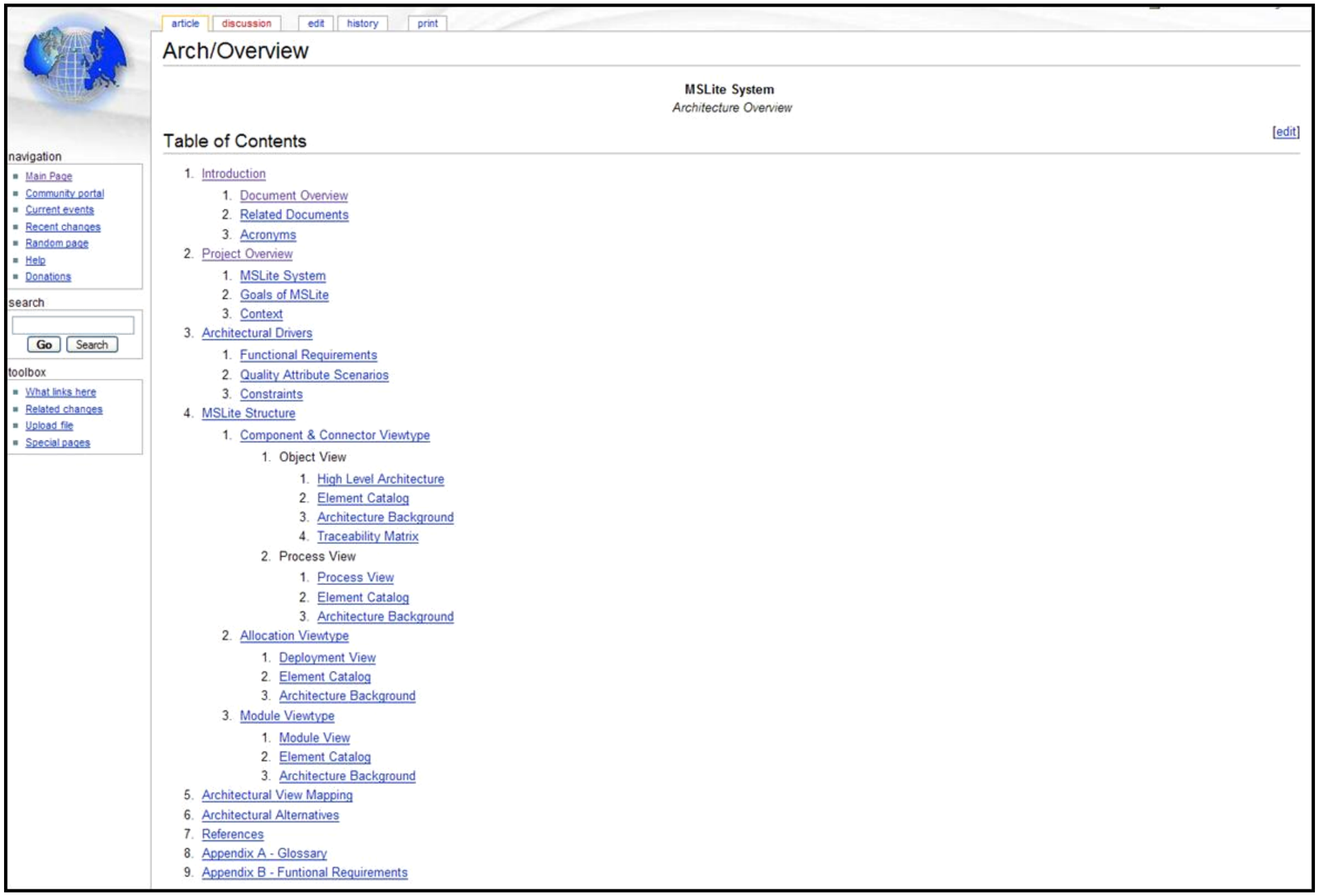
Siemens Corporate Research, Inc. also is convinced that it is at least worth testing the approach on a larger scale. The company took over the installation of the wiki infrastructure and plans to drive the implementation of the system by many geographically distributed teams.

We will follow this effort to identify benefits for the users (not just the producers) of the architecture documentation. In particular, we are interested to see whether wiki will support the unavoidable evolution of the architecture and whether the organization will be able to keep the architecture documentation current.

Another aspect we will observe is the support for distribution. The customer’s implementation of the system will be done by multiple teams distributed around the globe. We want to answer this question: “Would a tool like wiki reduce the risk for distributed software development?”

Appendix Screen Shots from Siemens Project Wiki

In this appendix, Siemens Corporate Research, Inc. provided some screen shots taken from the existing wiki.



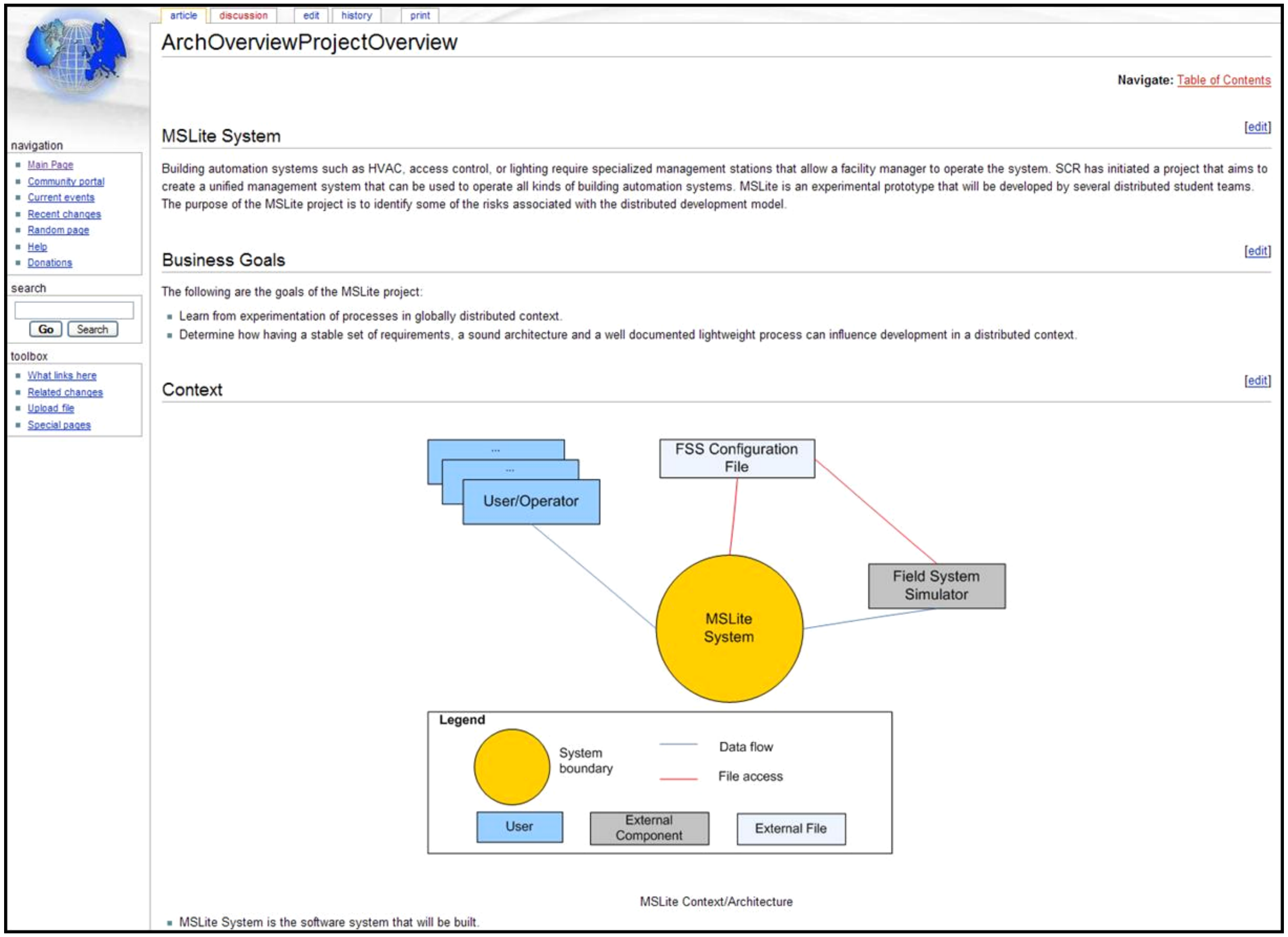
Arch/Overview

MSLite System
Architecture Overview

Table of Contents

1. Introduction
    1. Document Overview
    2. Related Documents
    3. Acronyms
2. Project Overview
    1. MSLite System
    2. Goals of MSLite
    3. Context
3. Architectural Drivers
    1. Functional Requirements
    2. Quality Attribute Scenarios
    3. Constraints
4. MSLite Structure
    1. Component & Connector Viewtype
        1. Object View
            1. High Level Architecture
            2. Element Catalog
            3. Architecture Background
            4. Traceability Matrix
        2. Process View
            1. Process View
            2. Element Catalog
            3. Architecture Background
    2. Allocation Viewtype
        1. Deployment View
        2. Element Catalog
        3. Architecture Background
    3. Module Viewtype
        1. Module View
        2. Element Catalog
        3. Architecture Background
5. Architectural View Mapping
6. Architectural Alternatives
7. References
8. Appendix A - Glossary
9. Appendix B - Funtional Requirements

Figure 4: Basic Layout of a Wiki Page



ArchOverviewProjectOverview

Navigate: Table of Contents

MSLite System [edit]

Building automation systems such as HVAC, access control, or lighting require specialized management stations that allow a facility manager to operate the system. SCR has initiated a project that aims to create a unified management system that can be used to operate all kinds of building automation systems. MSLite is an experimental prototype that will be developed by several distributed student teams. The purpose of the MSLite project is to identify some of the risks associated with the distributed development model.

Business Goals [edit]

The following are the goals of the MSLite project:

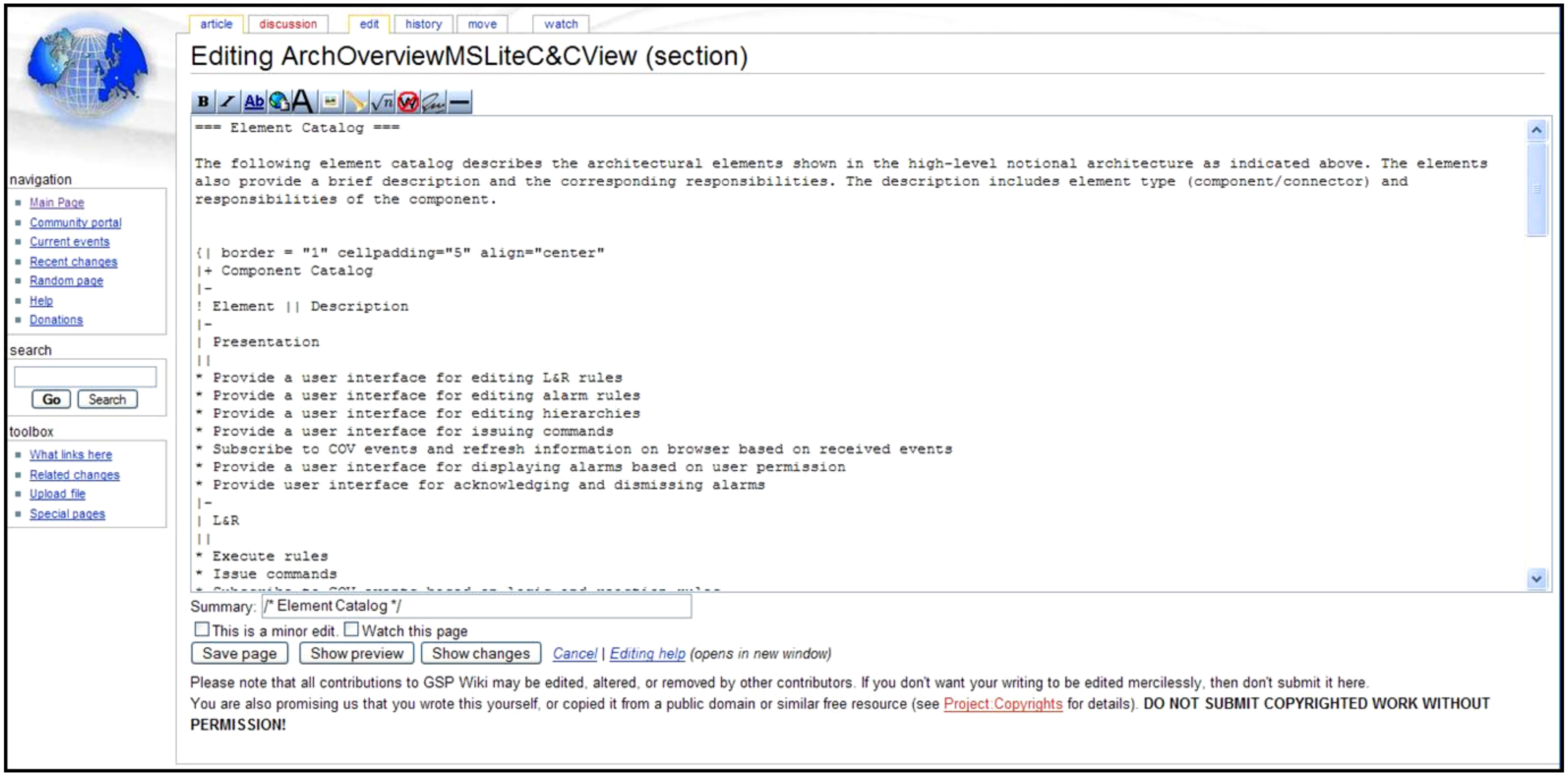
- Learn from experimentation of processes in globally distributed context.
- Determine how having a stable set of requirements, a sound architecture and a well documented lightweight process can influence development in a distributed context.

Context [edit]

MSLite Context/Architecture

- MSLite System is the software system that will be built.

Figure 5: A Wiki Page with Included Picture



Editing ArchOverviewMSLiteC&CView (section)

```
=== Element Catalog ===

The following element catalog describes the architectural elements shown in the high-level notional architecture as indicated above. The elements
also provide a brief description and the corresponding responsibilities. The description includes element type (component/connector) and
responsibilities of the component.


{| border = "1" cellpadding="5" align="center"
|+ Component Catalog
|-
! Element || Description
|-
| Presentation
||
* Provide a user interface for editing L&R rules
* Provide a user interface for editing alarm rules
* Provide a user interface for editing hierarchies
* Provide a user interface for issuing commands
* Subscribe to COV events and refresh information on browser based on received events
* Provide a user interface for displaying alarms based on user permission
* Provide user interface for acknowledging and dismissing alarms
|-
| L&R
||
* Execute rules
* Issue commands
```

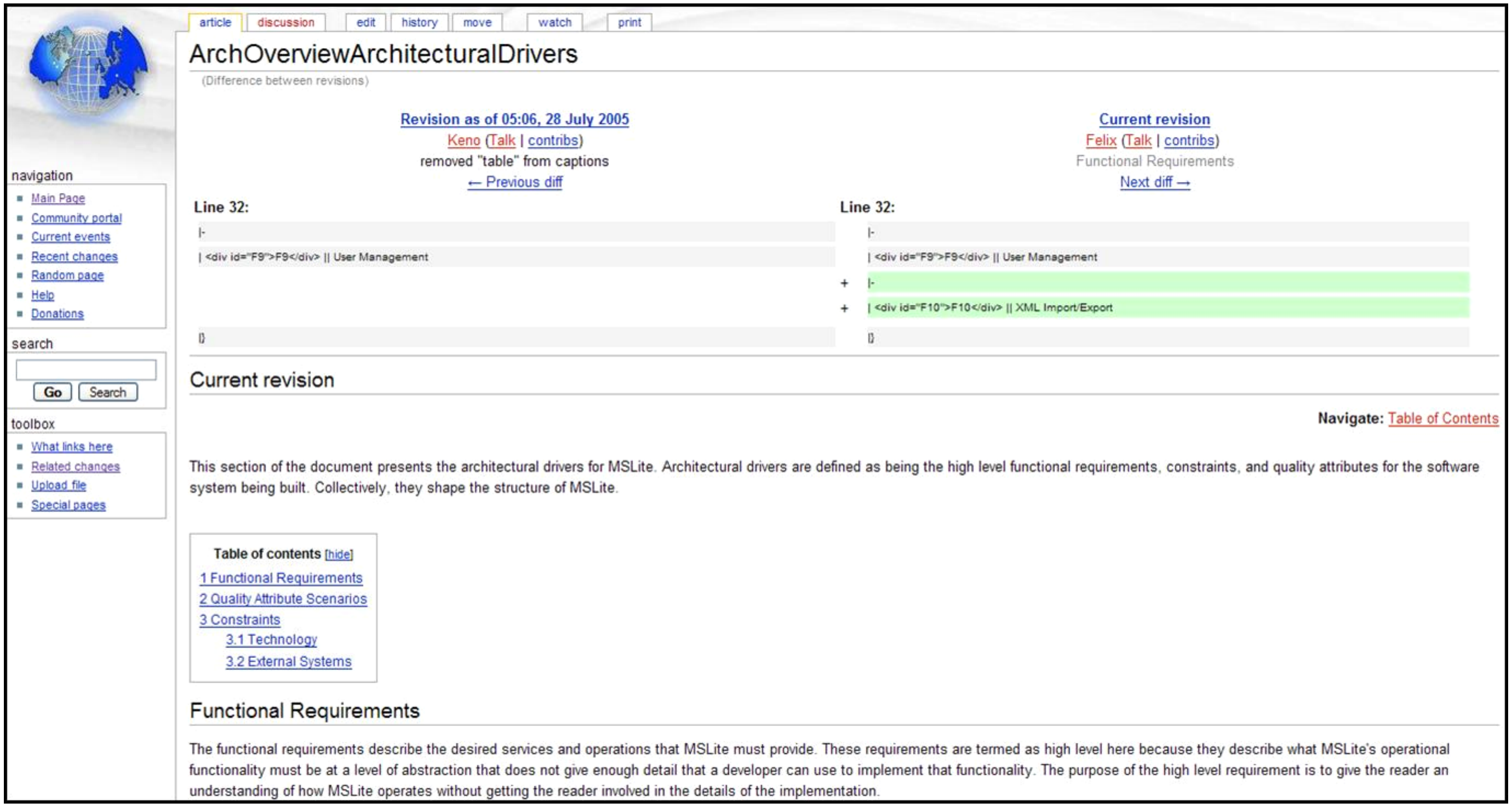
Summary: /* Element Catalog */

This is a minor edit. Watch this page

Save page | Show preview | Show changes | Cancel | Editing help (opens in new window)

Please note that all contributions to GSP Wiki may be edited, altered, or removed by other contributors. If you don't want your writing to be edited mercilessly, then don't submit it here.
You are also promising us that you wrote this yourself, or copied it from a public domain or similar free resource (see Project:Copyrights for details). **DO NOT SUBMIT COPYRIGHTED WORK WITHOUT PERMISSION!**

Figure 6: Editing a Section of a Wiki Page



ArchOverviewArchitecturalDrivers

(Difference between revisions)

Revision as of 05:06, 28 July 2005	Current revision
Keno (Talk | contribs)	Felix (Talk | contribs)
removed "table" from captions	Functional Requirements
← Previous diff	Next diff →

```
Line 32:                                      Line 32:
|-                                            |-
| <div id="F9">F9</div> || User Management    | <div id="F9">F9</div> || User Management
                                            + |-
                                            + | <div id="F10">F10</div> || XML Import/Export
|}                                            |}
```

Current revision

Navigate: Table of Contents

This section of the document presents the architectural drivers for MSLite. Architectural drivers are defined as being the high level functional requirements, constraints, and quality attributes for the software system being built. Collectively, they shape the structure of MSLite.

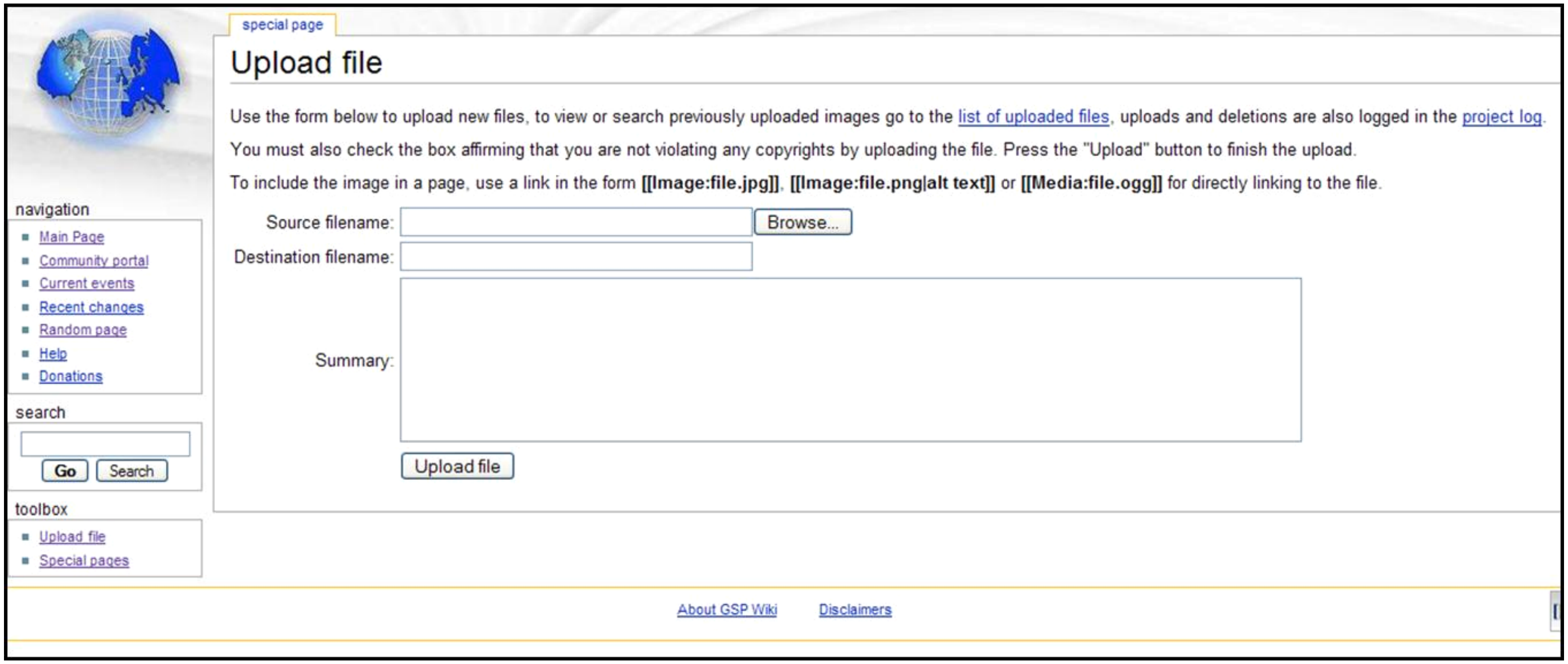
Table of contents [hide]

1 Functional Requirements
2 Quality Attribute Scenarios
3 Constraints
3.1 Technology
3.2 External Systems

Functional Requirements

The functional requirements describe the desired services and operations that MSLite must provide. These requirements are termed as high level here because they describe what MSLite's operational functionality must be at a level of abstraction that does not give enough detail that a developer can use to implement that functionality. The purpose of the high level requirement is to give the reader an understanding of how MSLite operates without getting the reader involved in the details of the implementation.

Figure 7: Displaying the Differences Between Two Versions of the Same Wiki Page



Upload file

Use the form below to upload new files, to view or search previously uploaded images go to the list of uploaded files, uploads and deletions are also logged in the project log.

You must also check the box affirming that you are not violating any copyrights by uploading the file. Press the "Upload" button to finish the upload.

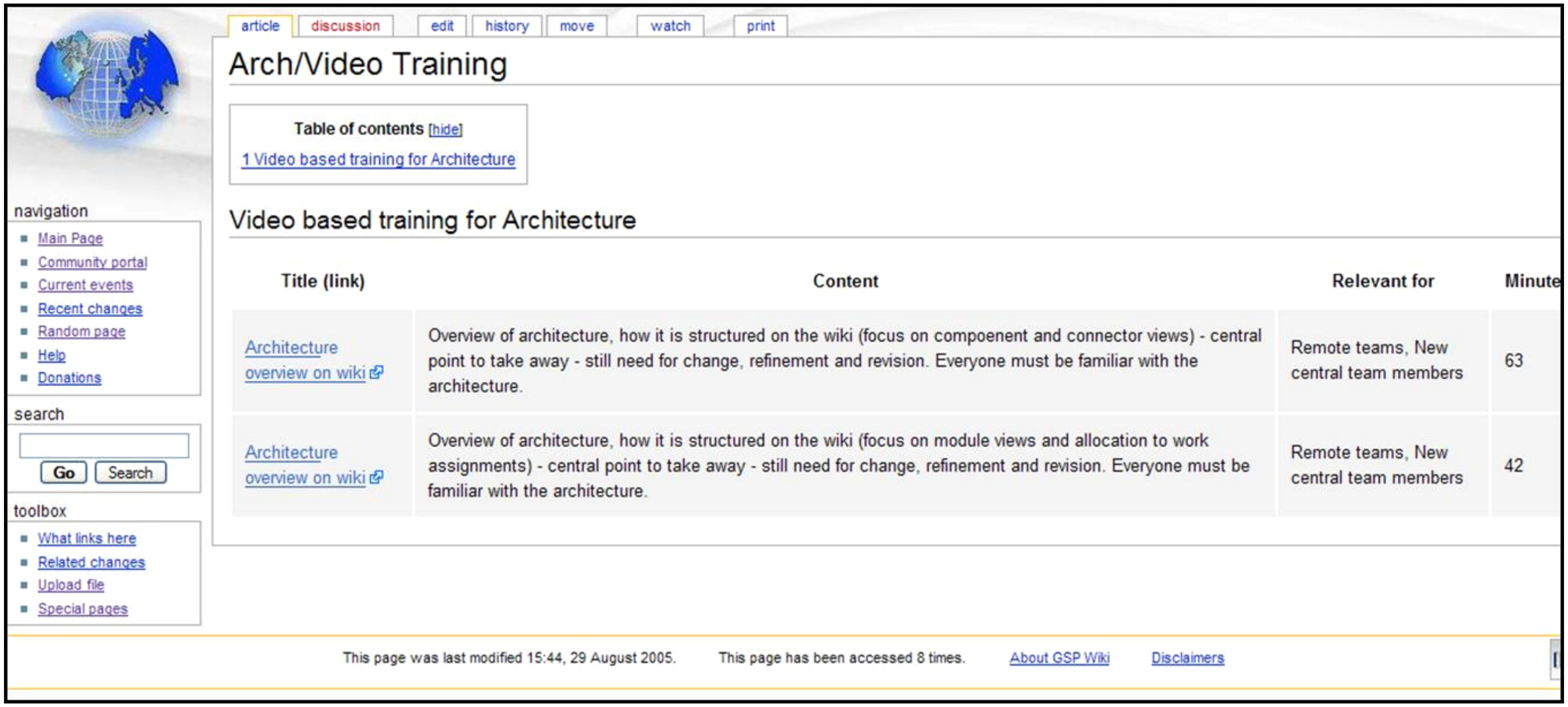
To include the image in a page, use a link in the form **[[Image:file.jpg]]**, **[[Image:file.png|alt text]]** or **[[Media:file.ogg]]** for directly linking to the file.

Source filename:

Destination filename:

Summary:

Figure 8: Uploading an Image File into Wiki



Arch/Video Training

Table of contents [hide]

1 Video based training for Architecture

Video based training for Architecture

Title (link)	Content	Relevant for	Minute
Architecture overview on wiki	Overview of architecture, how it is structured on the wiki (focus on compoenent and connector views) - central point to take away - still need for change, refinement and revision. Everyone must be familiar with the architecture.	Remote teams, New central team members	63
Architecture overview on wiki	Overview of architecture, how it is structured on the wiki (focus on module views and allocation to work assignments) - central point to take away - still need for change, refinement and revision. Everyone must be familiar with the architecture.	Remote teams, New central team members	42

This page was last modified 15:44, 29 August 2005. This page has been accessed 8 times. About GSP Wiki Disclaimers

Figure 9: Training Materials Included

References

URLs are valid as of the publication date of this document.

[Clements 02] Clements, P., et al. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

[Kazman 04] Kazman, R. & Nord, R. L. "Integrating Architecture Methods: The Case of the ATAM and the CBAM." http://www.sei.cmu.edu/news-at-sei/columns/the_architect/2004/1/architect-2004-1.htm (2004).

[Leuf 01] Leuf, B. & Cunningham, W. *The Wiki Way: Collaboration and Sharing on the Internet*. Boston, MA: Addison-Wesley, 2001.

[Singh 02] Singh, I., et al. *Designing Enterprise Applications with the J2EE Platform,* Second Edition. Boston, MA: Addison-Wesley, 2002.

[Wikipedia 06a] Wikimedia Foundation, Inc. *Portland Pattern Repository.* http://en.wikipedia.org/wiki/Portland_Pattern_Repository (2006).

[Wikipedia 06b] Wikimedia Foundation, Inc. *MediaWiki*. http://en.wikipedia.org/wiki/MediaWiki (2006).

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2005	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Experience Using the Web-Based Tool Wiki for Architecture Documentation		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Felix Bachmann, Paulo Merson		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-TN-041
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

In an organization that uses an architecture-centric development approach, it is the purpose of the software architecture, especially the product documentation, to guide all stakeholders who contribute in one way or another to the development of the product(s). Unfortunately, in many organizations, this documentation ends up on the shelves, unused and collecting dust. This happens in part because it is difficult to keep the architecture documentation current, hard for nondevelopers to understand what the documents describe, and challenging for nondevelopers to use the tools necessary to access the documentation.

This technical note discusses the benefits and challenges of using a wiki-based collaborative environment to create software architecture documentation. The findings are based on two experiences. The first was that of a team of Carnegie Mellon® University Master of Software Engineering (MSE) program students that used the wiki tool in a real-world software project. For its customer, the team had to produce and document the architecture of a system that will be developed by many geographically distributed teams. The second experience was a study conducted by another MSE student to reconstruct and document the architecture of a multitier enterprise application using the wiki tool and UML 2.0.

14. SUBJECT TERMS wiki, software architecture, architecture documentation, distributed development			15. NUMBER OF PAGES 44
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL