Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development

William Kim

Adviser: Sam Chung, Ph.D. (Chair)
Institute of Technology,
University of Washington, Tacoma, WA, USA

Adviser: Barbara Endicott-Popovsky, Ph.D.
Center for Information Assurance & Cybersecurity
University of Washington, Seattle, WA, USA
{willkim, chungsa, endicott}@uw.edu

Abstract

Popular Open Source Software (OSS) development platforms like GitHub, Google Code, and Bitbucket take advantage of some best practices of traditional software development like version control and issue tracking. Current major open source software environments, including IDE tools and online code repositories, do not provide support for visual architecture modeling. Research has shown that visual modeling of complex software projects has benefits throughout the software lifecycle. Then why is it that software architecture modeling is so conspicuously missing from popular online open source code repositories? How can including visual documentation improve the overall quality of open source software projects? Our goal is to answer both of these questions and bridge the gap between traditional software engineering best practices and open source development by applying a software architecture documentation methodology using Unified Modeling Language, called 5W1H Re-Doc, on a real open source project for managing identity and access, MITREid Connect. We analyze the effect of a model-driven software engineering approach on collaboration of open source contributors, quality of specification conformance, and state-of-the-art of architecture modeling. Our informal experiment revealed that in some cases, having the visual documentation can significantly increase comprehension of an online OSS project over having only the textual information that currently exists for that project. We recommend more rigorous evaluation of these claims as future work.

Keywords: Open Source Software Development, Software Architecture Documentation, Model-Driven Software Engineering

1. INTRODUCTION

Open Source Software (OSS) development allows for distributed collaboration on software projects that can sometimes compare in the size and scope of traditional enterprise applications. Mainstream online OSS development hubs such as GitHub[1], Google code[2], and Bitbucket[3] have evolved to fit the needs of a globally dispersed population of developers. All three of these sites

[1] GitHub, https://github.com/
[2] Google Code, https://code.google.com/
[3] Bitbucket, https://bitbucket.org/

incorporate an Agile-like iterative development approach with a system of continual code changes using one or more of the major version control systems, allowing for highly flexible change control and release cycles, a necessity in today's constantly changing tech landscape. Project management is performed using a system of issue tracking, tasking, and comment system, which encourages the social angle of the online development process.

The general OSS development pattern may not necessarily map directly to the stages of traditional software development life cycle—requirements, design, implementation, testing, and maintenance. We can see that the common lifecycle of an OSS iteration—1) Report bug. 2a) Developer is tasked or 2b) code pull request is submitted. 3) Change is committed to the code base—fits tightly to the highly iterative implementation, testing, and maintenance loop. But what happens further back at the beginning of the engineering process, namely the requirements and the design process?

Currently, the vast majority of design and requirements manifest in the form of developer documentation [1]. These take various forms such as like public wiki pages, getting-started articles, reference documentations, engineering specifications like IETF RFCs, code comments, etc. At this point, we would like to note that this documentation is essentially all textual information and move on to the second key focus of this paper.

It is a foundational principle of software engineering that a good design is critical towards the success of a complex and large software project. Visual modeling of software architecture as part of the design efforts can have significant benefits to the software maintenance process. In particular, Unified Modeling Language (UML), which has become the industry standard for software modeling, has been shown to significantly improve correctness of changes by those who used it while adding an insignificant overhead of maintaining the UML documentation [2].

The study referenced above [2] did not deal with the OSS community, which may add other variables such as the technical qualifications of current and potential contributors. However, we assert that having visual documentation in the form of UML models can have a positive impact on OSS development. At the very least, additional meaningful documentation may draw in more newcomers who choose open source projects based strictly on the availability of good starter documentation to break into the project [3].

Despite the opportunity for highly distributed participation, many OSS projects are highly centralized (See Table 1). This reflects a challenging situation where very few people (sometimes a single individual) is managing the development and health of any given large OSS project. Perhaps not surprisingly, the quality of OSS project has been shown to go down as the number of minor contributors (low expertise) goes up [3]. Improvements can be made to increase the effectiveness of distributed development in OSS, both for the managers of such projects and the distributed contributors to those projects.

Table 1. OSS project participation

*Top three most starred GitHub projects. †Top contributors measured by number of code commits. Data gathered on May 6, 2014.

Project Name	Total # of Contributors	Top Contributor† % of Commits	Top 3 Contributors % of Commits
twbs/bootstrap*	597	57.6	77.2
jquery/jquery*	223	30.6	49.1
joyent/node*	571	30.5	61.1
openssl/openssl	47	29.9	64.7
mitreid-connect/OpenID-Connect-Java-Spring-Server	18	56.6	82.4

UML is at least part of the answer for development in the traditional enterprise software. Software giant IBM champions its use and develops products for implementing UML processes[4]. However, UML is a massive framework and may not all be appropriate for the fast-paced, iterative development needs of the OSS community. That is why we propose the use of a re-documentation methodology known as 5W1H Re-documentation, based loosely on the journalistic 5 W's and 1 H of Who, What, When, Where, Why, and How [4]. This approach strips down the UML framework to apply the core needs of re-documenting a legacy system. We hope to apply this methodology to the OSS development processes and maintain the benefits observed for more traditional enterprise software endeavors.

The next section of this paper will discuss previous work relating to the issues described above. Section 3 will give an overview of our three-part research approach including the problem statements that drive them. The following three sections will discuss each approach in detail with results. Finally, we end the paper with our conclusions and suggestions for future work.

2. PREVIOUS WORK

[4] IBM - Rational Unified Modeling Language - UML Resource Center. www.ibm.com/software/rational/uml/. May 2014

In Table 2, we identify the three key papers that we use as the groundwork of our research. One of the goals of our research is to combine the contributions of each of these authors into our own research approach.

Table 2. Key Topics in Previous Work Papers Compared to This Paper

Paper	OSS Development	UML	Architecture Modeling Best Practices
A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance [2]	no	yes	no
Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors [1]	yes	no	no
Service-Oriented Reverse Reengineering: 5W1H Model-Driven Re-Documentation and Candidate Services Identification [4]	no	yes	yes
Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development	yes	yes	yes

2.1 A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance

This paper by Wojciech James Dzidek contains an empirical study of the effect of UML on the maintenance of real software projects is the basis of two aspects of our research. First, it provides the motivation for considering the benefits of using UML and software architecture modeling in general for OSS development. Second, it provides us with a rigorous example of how we can empirically test whether using UML benefits software development.

The main conclusion of this paper is that additional overhead of using UML did not have a significant impact on the time it took for software maintenance tasks while having a statistically significant positive impact on the functional correctness of those changes. This clearly shows some benefits to using UML under the conditions set in this paper's experiment.

This paper also has the usefulness of demonstrating how the benefits of using UML can be

actually tested. One of problems is concerning exactly that, except in the online OSS environment (see Section 3 for exact problem statement).

2.2. Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors

This paper by Barthélémy Dagenais explores various types of documentation (albeit all textual forms of documentation) and their effects on decisions of open source contributors. These decisions include whether or not to contribute to certain projects in the first place, and the extent and quality of code contributions, and the amount of effort invested in documentation creation and maintenance.

One of the key findings of this paper was how different types of developer documentation had different effects (both positive and negative) on certain factors such as quality of contributions and additional maintenance overhead. Even though all the types of documentation in this study were textual forms, it goes to show how not just the existence and quantity of documentation is important, but the type as well.

2.3 Service-Oriented Reverse Reengineering: 5W1H Model-Driven Re-Documentation and Candidate Services Identification

As new developers and implementers encounter a new software project or system, there is a need to understand it as quickly and effectively as possible. Visual architecture modeling can be a great asset for the aiding comprehension of a new project. Phillip Kruchten's landmark paper on the 4+1 Views architecture modeling in 1995 gave us some high-level guidelines on how architecture of a software project can be abstracted [5]. However, we need more guidelines for best practices on what to do about an existing project that did not have this kind of architectural understanding from the start.

This is where re-engineering re-documentation comes in. By taking the perspective of the 5 W's (who, what, when, where, and why) and 1 H (how) and making parallels between those and the different views of the architecture, we can perhaps have clearer understanding of the software itself. Further empirical testing of this hypothesis is future work of this paper.

3. APPROACH

Based on our literature survey, we identified three main problem areas of our topic. Phrased as questions, they are as follows: 1) Why is software architecture modeling missing from popular online open source code repositories? 2) How can we model complex modern software projects that have multiple components and technologies as part of one software project? 3) How can including visual documentation improve the overall quality of open source software projects?

4. Architecture Modeling Features in Online OSS Development

In order to answer the question of why visual architecture modeling is apparently missing from the popular OSS development hubs, we first had to determine that visual architecture and other visual documentation features were in fact not part of the mainstream OSS development process. This was done through a case study of existing documentation features of three of the top online OSS repositories. The results are shown in Table 3. By virtue of being open source code repositories, all three sites offer storage and access to source code and commit history. The major project management mechanism is the issue tracking systems on each repository site. All three sites also host a project home page and "wiki" system for each software project hosted on their site.

Table 3. Open Source Software Repository Names of Links to Features

Repository Name	Home Page	Commit History	Issue Tracker	Wiki	Source Code
GitHub[1]	readme.txt	commits	Issues	Wiki	(default homepage)
Google Code[2]	Project Home	Changes	Issues	Wiki	Source
Bitbucket[3]	Overview	Commits	Issues	Wiki	Source

Altogether, documentation on these three sites exist as a combination of code comments in the source code, messages in the issue tracker and commit history, wiki pages, and the project descriptions on the home page. Not counting links to external resources, this documentation is completely textual.

Finding willing contributors for architecture documentation may not be easy. First, there is a problem of experience. Developers with specific domain experience of the project may not necessarily have architecture development training or experience required to do effective architecture development and those trained in software architecture may not have the appropriate domain knowledge to apply it well [6]. However, at least one benefit is that the type and quality of documentation, such as “Getting Started” articles, can be used as a marketing tool to attract new contributors to the project [1].

A further study of available UML tools shows very little in mature free open source software (FOSS) UML modeling tools. There are some enterprise-level UML tools available for a price, but this goes directly against the spirit of OSS development as the whole technology stack from platform, language, integrated development environments (IDE), version control systems, and repositories all have mature FOSS options.

One of the most popular IDEs for the Java platform is Eclipse. The Eclipse Modeling Project[5] is an effort to promote model-based development and provide a unified set of modeling frameworks, tooling, and standards implementations. Many of the tools support meta-modeling

[5] Eclipse Modeling Project. http://www.eclipse.org/modeling/

following UML specifications. These meta-models are developed and maintained by via syntactic representations in semi-structured text. An example of this is the Eclipse UML2 plugin[6]. Again, while the models are conceptually architectural in nature, the tangible existing artifact is still textual.

One of the main visual diagramming tools from the above suite of tools and frameworks is known as Papyrus[7]. By their own description, "Papyrus is graphical editing tool for UML2 as defined by OMG." Started in 2008, this project is not in a functional release state. At the time of this writing, the latest release version was 0.10.2.

This cursory analysis of the state of the art of FOSS graphical UML modeling tools with Papyrus as an example may indicate that FOSS options for graphical UML tools for visual architecture modeling may still be in immature stages. This could support the argument that the lack of tool support is one reason for why visual architecture modeling is not prevalent in projects on the major online OSS repositories.

Another reason may be that OSS technologies are evolving faster than the open source UML tool development effort. This claim not validated in this paper and is left as future work. However, in the next section, we discuss the considerations when modeling architecture of modern OSS applications.

5. Best Practices for Modeling Architecture of Modern Applications

Many modern applications are made up of multiple components and technologies. Any given software project may use many programming languages, especially in the case of web applications where client side and server side functionality have their own suite of platforms and languages. This complexity is a challenge of effectively modeling the software of these composite applications which may not fit neatly to the picture of stand-alone enterprise applications.

Because OSS is often in the category of new, leading edge software using the latest languages and frameworks, we have an increasing need in the open source for best practices of architecture modeling, given the premise that architecture modeling is important to OSS in the first place.

The mix of static and dynamic content from web applications plus the inherently distributed nature of interaction makes web applications not fit neatly into the categories of defined elements of UML. The official OMG UML specification includes extension points with this in mind [7]. Attempting to apply UML to web applications is not new, as a literature survey will reveal suggestions on how to adapt UML to web applications. UML Stereotypes seem like the obvious choice, while other less obvious suggestions may be to reduce the scope of the architecture to just pages, hyperlinks, and dynamic content--such as by modeling each page as its own class

[6] http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Getting_Started_with_UML2/article.html
[7] Papyrus, http://www.eclipse.org/papyrus/

(one class for server side page and one for the client side page) [8].

By using the Sparx Enterprise Architect[8] tool, we performed a case study of modeling an OSS project known as MITREid Connect[9] in order to generate our own best practices of visual architecture modeling. MITREid Connect is a reference implementation of a new web authentication protocol called OpenID Connect. There is an active community of developers and implementers for both the OpenID Connect protocol and the OAuth 2.0 protocol it is built on. This means that there is a wealth of textual information on the web about this project.

MITREid Connect was chosen for two reasons: the author's familiarity with the code base and the fact that the project is a combination of both front-end JavaScript and HTML code plus back-end Java code. This is an example of an application with multiple components and different technologies. Because UML is a massive framework requiring expertise of its own to be proficient, we rely on a methodology to simplify and streamline the re-engineering process of the architecture. Using the 5W1H Re-Documentation methodology, we developed all 4+1 views in diagrams using Enterprise Architect.

We discovered that despite multiple languages and technologies being used, the Use Case, Component, and Deployment Views were fairly straightforward to diagram. These views do not depend on any specific programming constructs so the fact that multiple languages were used did not affect the difficulty of this part (Figure 1).

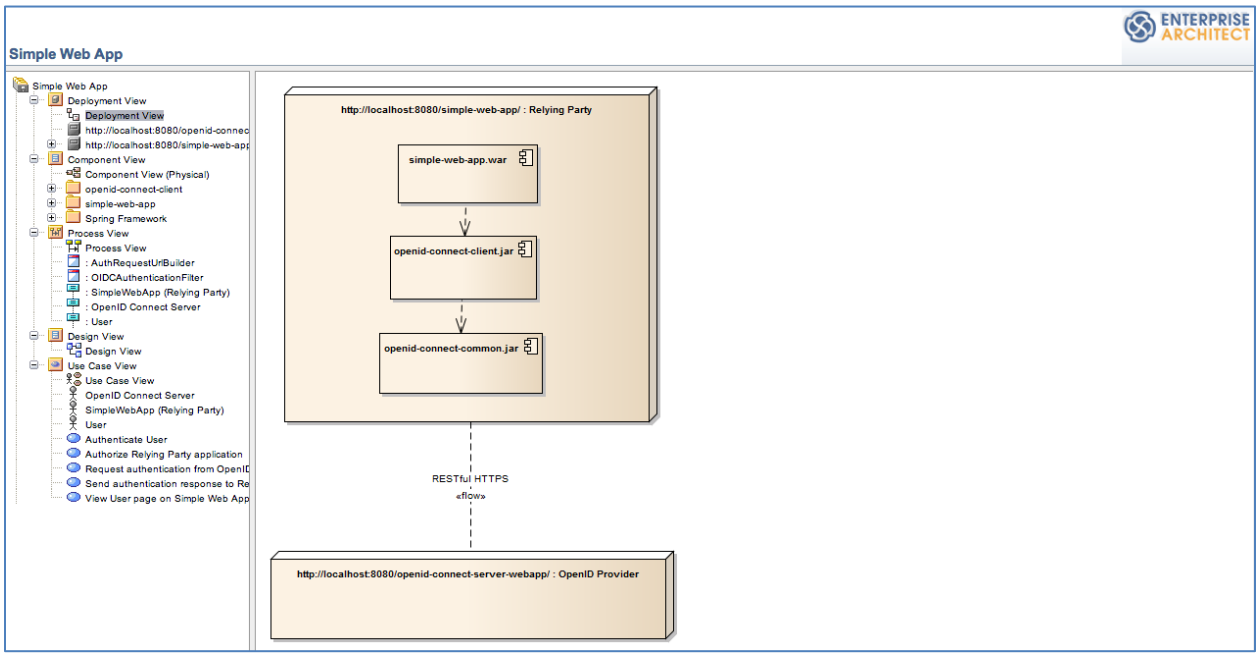

Figure 1. Deployment View

Design and Process Views were much more difficult. These views depend on certain

[8] Sparx Systems Enterprise Architect, http://www.sparxsystems.com/products/ea/index.html. May 2014.
[9] MITREid Connect, https://github.com/mitreid-connect/

assumptions about being able to abstract to classes, object instances, and message exchanges between objects (Figure 2). At the boundaries of where the Java code ends and other parts of the application begin, the ability to diagram the design becomes more challenging.

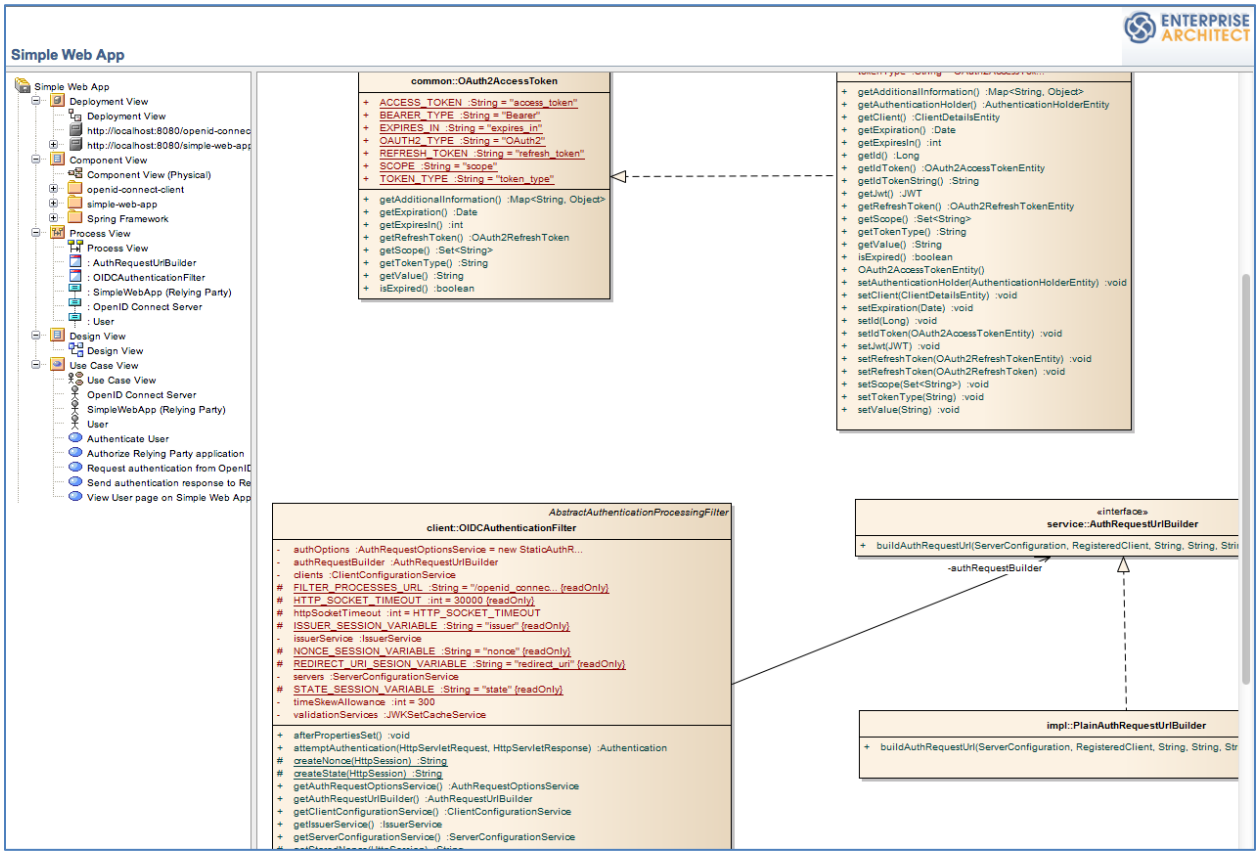

Figure 2. Design View

6. Effectiveness of Visual Documentation in Online OSS Development

The last component of this research was to explore how including visual documentation can improve the overall quality of open source software projects. First, we created a mock site of our UML diagrams (Figure 3). We used the publishing feature of Enterprise Architect tool to generate graphical diagrams viewable as HTML pages continuing to use the MITREid-Connect project as our case study. We then retrofit a local copy of a GitHub project page to include a link to the models, as a thought experiment of the implications of adding such a feature (the link to Models in Figure 2 does not exist on actual GitHub sites). We list a few of our observations below.

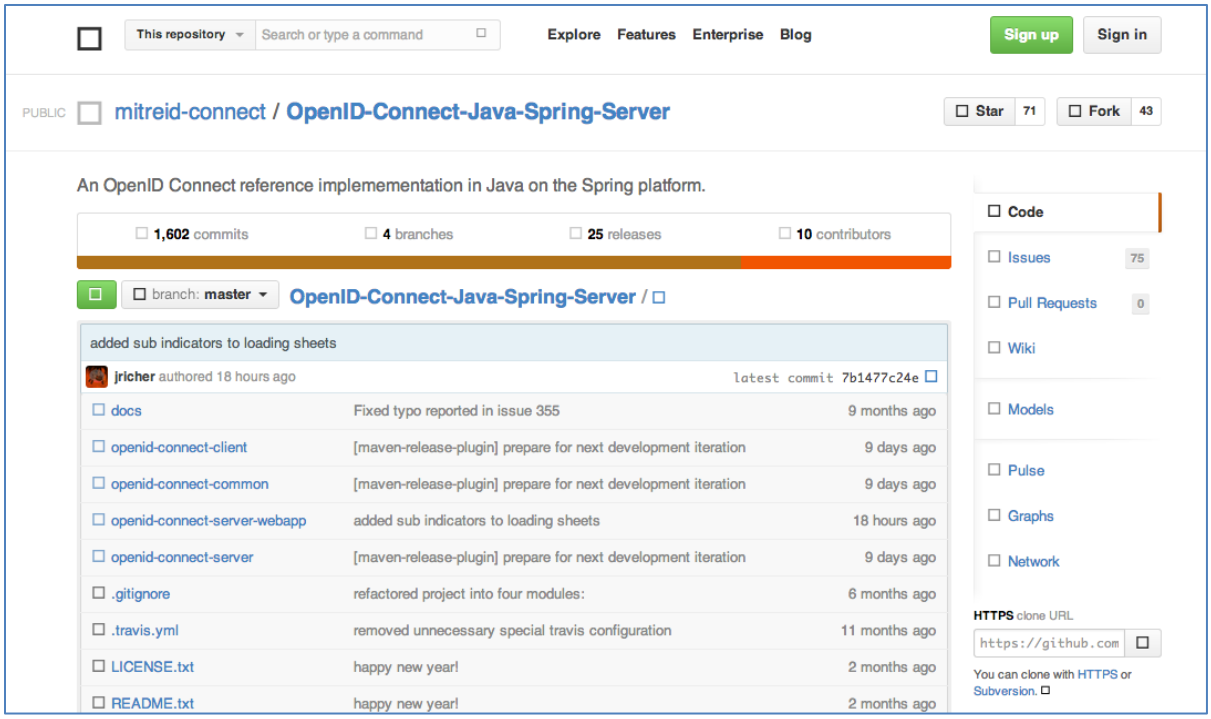

Figure 3. Mock Site of Modeling Functionality

As with program documentation of other forms, one of the main challenges that jump out here is the problem of maintaining and updating the documentation. Definitely the socio-technical dynamics of the project group and contributors affect the code contributions, but documentation updates may not be as related to social network effects [3]. In order to improve the social coding benefits of OSS, we propose that OSS diagrams could have a feature for posting comments to parts of the visual models, just as users can post comments on specific lines of code.

6.1 Experiment

As an initial effort to evaluate the effect of including visual documentation on OSS projects, we designed a small scale experiment using the model diagrams mock site prototype. We used two groups of three university students of various grade levels and taking either an Information Technology or Computer Science major. Their selection was through a voluntary participation upon request to a certain research group at the university. Their task was to answer a five questions about relating actual code of the MITREid-Connect project to the OpenID Connect specifications it is implementing (see Appendix A). The questions were free response answers, but there were explicitly correct answers for each problem. The control group was allowed access to any online resources they can find, including the official OpenID Connect specifications and the source code and documentation on the MITREid-Connect GitHub page. The experimental group had access to same resources plus access to the mock site including the diagrams we developed from earlier part of this research.

This experiment was not intended to be highly rigorous or scientific but should still achieve our goal of bringing to light some issues related to this subject. One very interesting result was that even a question as straightforward as what library dependencies are required by a the web application in question was answered incorrectly by all three participants in the control group while answered correctly by all of the experimental group (see Table 4). The reason was that the experimental group obtained their information from the Deployment View of the diagrams which showed a nested dependency of one library to another that the other group failed to catch from the source code alone. At least in this example, there was a clear benefit to the correctness of developer understanding in the group. Despite having all of the world wide web at their fingertips, these kinds of architectural details can be missed, especially without sufficient training in all of the technologies used and under a pressure situation.

Table 4 shows the results of this experiment. The questions are available in Appendix A. In order to be able to score partially correct answers, each question was scored out of 2 points--2 points for completely correct answer, 1 point for partially correct answer, and 0 points for incorrect answer. All subjects were either Information Technology Systems (ITS) or Computer Science & Systems (CSS) students. The CSS students had significantly more programming related coursework than the ITS students.

Table 4. Experiment Results.

Subject #	Experiment or Control Group	Major of Study	Question 1	Question 2	Question 3	Question 3a	Question 4
1	control	ITS	1	2	0	2	0
2	control	ITS	1	0	2	2	2
3	control	CSS	1	2	0	0	2
4	experiment	CSS	2	2	2	2	2
5	experiment	CSS	2	2	2	2	0
6	experiment	CSS	2	0	2	0	2

Based on the feedback from the test subjects, the other questions in the survey proved to be too demanding to be completed in the time frame allowed. The reasons varied from a lack of domain experience, programming experience, or both. The results from the other questions in the survey were highly inconclusive across both groups and therefore omitted from this study. Student #4 outperformed the rest of the group due to having prior knowledge of the MITREid Connect code and should be considered an outlier.

6.2. Experiment Validation

The test subjects were chosen through a voluntary participation request from a student research group. There was no expectation of random sampling and both the sampling group and sampling method has some bias. First, the control group was aware of the additional resources that the experimental group received. This may affect their level of motivation to complete the tasks. The test subjects and the tester are all familiar with each other, introducing other factors of potential bias. The sample questions were designed with the model diagrams in mind. This may limit the evaluation of the results to strictly within the conditions of this experiment.

7. Conclusion

In summary, we conclude that the additional overhead of adding a new type of documentation to OSS projects may be justified by the benefits that the existing textual documentation brings for OSS projects. The existence of easily digestible visual diagrams may attract new contributors to a project. And these newcomers can possibly become oriented with the functionality of the software more quickly. We have described some best practices of modeling modern web applications using a case study but discovered many challenges in this work described below. Although our experiment was quite informal and hardly can be considered rigorous, we still see that there is a potential for the quality of code (as measured by lesser number of faults) may increase due to better correctness of understanding by contributors that use models.

We identified some key challenges of including architecture modeling in OSS projects. Mainly, it appears that software development technologies are changing faster than OSS graphical modeling tool support can keep up with. While we can do our best to adapt these new applications to UML using stereotypes and other extension points, how long until next major programming paradigm renders it all obsolete? For instance, we proposed that regardless of the actual programming language at hand, we can design and re-engineer a system in an object oriented way. Already there were new programming paradigms such as aspect-oriented programming that create new issues for this kind of methodology.

Adding additional documentation infrastructure to any kind of project adds additional maintenance costs. Suggested for future work is developing mockups for web-based architecture diagramming with social coding elements as described in Section 7, which may inspire more architecture investment in OSS projects. Further rigorous empirical study of the costs and benefits are required in order to validate these claims.

References

[1] Dagenais, B. and Robillard, M. P. 2010. Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. *FSE-18*, November 7–11, 2010, Santa Fe, New Mexico, USA.

[2] Dzidek, W. J., Arisholm, E., and Briand, L. C. 2008. A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. *IEEE Transactions on Software Engineering*, Vol. 34, No. 3, May/June 2008.

[3] Bird, C. 2011. Sociotechnical Coordination and Collaboration in Open Source Software, *27th IEEE International Conference on Software Maintenance*, 2011.

[4] Chung, S., Won, D., Baeg, S. H., and Park, S. 2009. Service-Oriented Reverse Reengineering: 5W1H Model-Driven Re-Documentation and Candidate Services Identification. *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, Jan.14-15, 2009.

[5] Kruchten, P.B. 1995. The 4+1 View Model of architecture, *IEEE Software*. Vol. 12, Issue 6, Nov. 1995. p. 42 –50.

[6] Foote, Brian. 1997. Big Ball of Mud, *Fourth Conference on Patterns Languages of Programs* (PLoP '97/EuroPLoP '97). September 1997.

[7] OMG. 2011. Documents Associated With Unified Modeling Language (UML), V2.4.1, August 2011. http://www.omg.org/spec/UML/2.4.1/

[8] Conallen, J. 1999. Modeling Web Application Architectures with UML, *Communications of the ACM*, Vol. 42, No. 10, Pages 63-70, October 1999.

Appendix

A. Experiment Survey Questions

In all of the following questions, refer to the MITREid-Connect reference implementation as basis of all software related questions.

1. Which of the three libraries from MITREid-Connect (openid-connect-client, openid-connect-common, and openid-connect-server) are needed by a Relying Party (or Client) application?

2. Which software components (class, object field, basic data type, etc…) represent the OAuth Access Token in software and what packages are they located in?

3. How does an OIDC client web application (relying party) submit an authorization request to the server? In other words, what sequence of class(es) and method(s) does the program use to do this?

    a. Where does the authorization request URL get built?

4. Where is the client code that processes ID Tokens it receives from the OpenID Server? Show a few lines of code (copy/paste) from the class that shows processing of the ID Token.