Architecture Reading Techniques: A Feasibility Study

Jeffrey Carver and Krystle Lemon
Mississippi State University
Department of Computer Science and Engineering
carver@cse.msstate.edu, kdl18@msstate.edu

Abstract

In order to correctly determine if the software architecture for a system complies with the user requirements, it is necessary to review the software architecture description. There are different means available to perform these reviews ranging from checklists to detailed step-by-step protocols. In this paper we propose the ARTs, a new set of reading techniques focused on software architecture. We provide an overview of these new techniques and report on an initial feasibility study. The results of the feasibility study showed that the techniques were useful and seen by the subjects to provide benefit over a checklist-based approach. Another interesting result requiring additional research is that the checklist seemed to focus reviewers on defects of commission while the reading techniques seemed to focus reviewers on defects of type omission.

1. Introduction

A software architecture document is the high-level description of a system that ignores implementation detail. It provides a means to model the structure, behavior, relationships, and constraints among the components of a system [7], [8]. The process of creating the software architecture document includes both making the architecturally significant decisions and recording those decisions in the document. This process is often included as part of the software design phase. Recently it has become a more independent discipline because of the benefits isolating software architecture decisions provide to the software [2]. Software architecture is of particular concern to implementers of a system because it serves as a blueprint for construction. At the software architecture level, it is assumed that the user requirements are documented and understood so that the system can further defined [12]. The architecture will eventually be refined into a more detailed design. Therefore, it is imperative that the software architecture be thorough, well understood, and as correct as possible to prevent defects from slipping to subsequent phases of the lifecycle where they are more expensive to repair [2].

Specific techniques such as checklists or reading techniques can facilitate the inspection of software architecture documents. In this paper we introduce the Architecture Reading Techniques (ARTs). The techniques in this set contain well-defined steps and questions to help an inspector find potential defects in the artifact. Because the ARTs are new they will evolve based on the results of empirical studies. This paper reports on an initial ARTs feasibility study, conducted in a university course. The main purpose of this study was to better understand the use of the ARTs and compare them to a checklist.

This paper is organized as follows. Section 2 provides an overview of the techniques and related work. Section 3 describes the feasibility study. Results and discussion are presented in Section 4. Finally, Section 5 discusses the conclusions and future work.

2. Architecture Reading Techniques

Previous work has shown the value of scenario-based reading techniques for inspecting various software artifacts [3, 9-11]. These findings led us to create a family of scenario-based reading techniques tailored for defect detection in software architecture documents. Our belief is that additional guidance provided by these techniques will focus reviewers on different defects than the checklist. The ARTs were developed using an approach consistent with that used to develop other scenario-based reading techniques.

Much of the information encoded in a software architecture document is captured in a series of diagrams. These diagrams contain different shapes that are connected by various types of lines, each having a specific meaning depending on the diagram. Additionally, the software architecture document contains textual descriptions that accompany and explain the diagrams. Due to the similarity between software architecture and software design documents,

i.e. each have multiple viewpoints, a set of Object Oriented Reading Techniques (OORTs) for design [11] were chosen as the inspiration for the development of the ARTs. The OORTs contained two types of techniques, horizontal and vertical. *Horizontal Techniques* focus on ensuring consistency among documents from the same lifecycle phase. Conversely, *Vertical Techniques* focus on ensuring traceability to a previous lifecycle phase.

The software architecture document contains information about three basic concepts accompanied by additional documentation. First, information about the logical structure (the code modules) is included. Second, information about the communication patterns (the run-time interactions) is included. Finally, information about the physical structure (code teams, hardware) is documented. Along with these three types of information, the architecture document also includes additional supporting information necessary to fully understand the system [5]. This understanding was used to create the four ARTs (one horizontal technique and three vertical techniques).

The first vertical technique focuses on ensuring that the logical decomposition of the modules in the software architecture is realistic based on the information provided in the requirements document. The second vertical technique focuses on ensuring that the communication patterns are accurate based on the requirements. The third vertical technique focuses on comparing the information about the stakeholders, architectural concerns and architectural rationale to the information contained in the requirements. The horizontal technique focuses on internal consistency within the document [4]. The technique includes three specific comparisons:

1. Logical structure vs. communication patterns
2. Logical structure vs. physical structure
3. Communication patterns vs. physical structure.

3. The Study

The main goal of the study was to evaluate the feasibility of the ARTs. Along with this goal, we also wanted to begin understanding the relationship between the ARTs and a more traditional checklist. The checklist was adapted from one described in the course textbook [5]. We were interested in understanding whether the ARTs and the checklist focused reviewers on different types of defects. The checklist provided detailed questions to help the reviewers determine if the architecture document was consistent with: the stakeholders, itself, good form, the requirements, and the underlying architecture it described.

The study was conducted at Mississippi State University in the Fall 2004 CSE 4233/6233 Software Architecture course. This course is offered to senior level undergraduate students and graduate students. The purpose of this course was to teach the students the fundamental software architecture concepts and how to document that information. As part of the course, the students incrementally developed a software architecture document for a software system. Twenty-three students participated as subjects in this study (18 undergraduates and 5 graduates).

Two software systems were used in this study. The first was the Tactical Software Aircraft Flight Evaluation (TSAFE) architecture. It was designed to assist air traffic controllers in detecting and resolving short-term conflicts between aircrafts [6]. This document was created by the researchers, and it was seeded with relevant defects based on previous experiences. The second was the Computer-Aided Dispatch System for the London Ambulance Service (LAS). The purpose of this system was to aid in the dispatch and scheduling of ambulances to respond to emergencies [1].

The study was performed in-vitro using the four homework assignments of the course. The first three assignments involved both inspection of an architecture document and creation of an architecture document. In each of these three assignments, the students inspected different aspects of the TSAFE document (i.e. nothing was re-inspected) created by the researchers using the checklist. Then they created a portion of their own architecture for the LAS. The fourth assignment involved only the inspection of an architecture document. In this assignment, the subjects used the ARTs to inspect a version of the LAS document provided by the researchers. This document was not the same one the students had created in the previous assignments. Table 1 provides a summary of the study details.

Assignment	Artifact	Technique
1	TSAFE - **Logical**	Checklist
2	TSAFE - **Runtime**	Checklist
3	Complete TSAFE	Checklist
4	Complete LAS	ARTs

Table 1 – Overview of Study

We collected an assortment of data across during all of the assignments to evaluate the techniques. The quantitative data included the number and type of defects found and the amount of time taken. This data was collected via a defect report form. To ensure that the defect data collected during the inspections was realistic, the subjects were given their homework grade based on completion of the assignment rather

than number of defects reported. We were more interested in the subjects following the instructions given and reporting only those items they truly believed were defects. We did not want the number of defect reports to be inflated by a false perception that more defects would equate to a higher grade.

We also collected qualitative data to help us better understand the use of the techniques. Qualitative data was extracted from written assignment reports in which the subjects provided a self-analysis of their performance during the inspection. At the conclusion of all four assignments, each subject completed a questionnaire to gauge their overall perception.

4. Results and Discussion

The quantitative data helped us to understand the types of defects found using the different techniques (checklist and ARTs). No statistical tests were run to compare checklist to ARTs. Because the study was designed primarily to gauge feasibility, it was impractical to compute such statistical analysis.

4.1 Summary of Quantitative Data

Our analysis was based on defect type, omission vs. commission. Defects of *omission* occur because some necessary information was left out of the document. Defects of *commission* occur because information was recorded incorrectly in the document.

The results from Assignment 1, focused on the logical portion of the TSAFE system, indicated that the subjects were more likely to find defects of commission than defects of omission. In Assignment 2, focused on the communication patterns of the system, the subjects were equally likely to find defects of omission and defects of commission. In Assignment 3, focused on the stakeholders, rationale and other information not yet inspected, the subjects were again more likely to find commission defects than omission defects. Finally, in Assignment 4, using the ARTs, the subjects were more likely to find defects of omission rather than defects of commission.

4.2 Summary of Qualitative Data

In addition to the quantitative results, we collected qualitative data to help us understand the feasibility and usefulness of the ARTs. Twenty out of the twenty-three subjects thought that the ARTs were more effective and structured than the checklist. Furthermore, the subjects reported that they preferred the ARTs to the checklist because the ARTs:

- focused their attention in specific areas
- gave specific instructions on how to inspect and explained the types of things that were likely defects

In order to understand the feasibility of the ARTs the subjects were asked if they needed any knowledge outside of basic software engineering. Eighty-five percent of the subjects indicated that they did not need any additional knowledge to use the ARTs. This qualitative data supports the idea that the ARTs are feasible and easy to understand and use.

4.3 Discussion of Results

These results suggest that the ARTs and the checklist have different effects on a software architecture inspection. The qualitative data gives insight into the inspector's perception that the ARTs provide much needed guidance. Furthermore, the quantitative data indicates that the ARTs may help inspectors find more defects of omission while the checklist may help find more defects of commission. The data from the four assignments is summarized in Figure 1. For assignments 1-3 (checklist-based) the subjects were generally more likely to find commission defects while on assignment 4 (ART-based) the subjects were more likely to find omission defects. No conclusions should be drawn based on the higher overall rate of the ARTs compared with the checklist. This result is likely due to the different artifacts and other experimental conditions.

4.4 Threats to Validity

As a feasibility study, there were multiple threats to

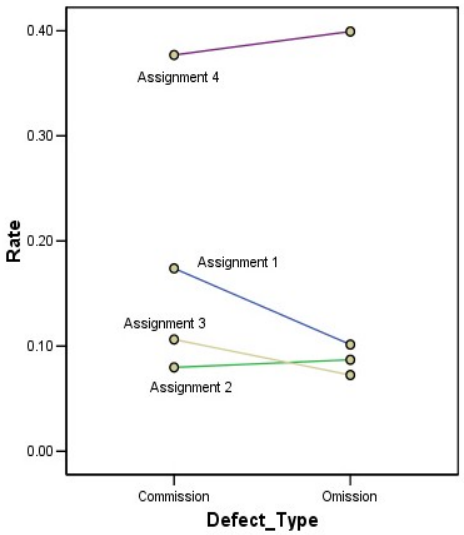

Figure 1 – Summary of Data

validity that must be addressed in future studies. These threats include, but are not limited to:

- Different artifacts used for each technique, (both the domain and the defect profile)
- Learning effects across the assignments

5. Conclusions and Further Work

This paper presented a new family of software architecture inspection techniques, the ARTs, and described a feasibility study to evaluate these techniques. The use of a checklist-based inspection was compared to the use of the ARTs. The results of the inspections were analyzed to determine the type of defects that were found more often using each technique. In the checklist-based inspections, the subjects were more effective at finding defects of commission than defects of omission. While in the ART-based inspection, the subjects found more defects of omission than commission. The ART's provided more guidance for the inspectors to identify information that was missing from the document such as modules, components, and connections. The subjects were able to effectively use the ART's because of the detailed guidance provided.

Our future work will include further investigation of the types of defects that can be uncovered by the ART's as compared with other techniques. In addition, we will examine characteristics of defects, other than omission or commission that may differentiate the checklist and the ARTs. We will also run a more controlled study using the techniques on the same (or similar) artifacts. Finally, the checklist and the ARTs appear to be complementary in the defects found. This result needs further investigation to determine the most effective way to combine the two approaches in an inspection.

Acknowledgements

This work supported in part by NSF grant CCF-0438923. We would like to thank the students of CSE 4233 at Mississippi State University for serving as subjects. We would also like to thank Byron Williams for providing feedback and comments. We would also like to thank the reviewers for their helpful comments.

References

[1] Allen, E. B., *Computer-Aided Dispatch System for the London Ambulance Service: Software Requirements Specification*, in *Technical Report*. 2003, Mississippi State University.

[2] Baragry, J. and Reed, K. "Why Is It So Hard to Define Software Architecture?" *Proceedings of Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific*. 1998, 28-36.

[3] Basili, V., Green, S., Laitenberger, O., Shull, F., Sorumgaard, S., and Zelkowitz, M., "The Empirical Investigation of Perspective Based Reading". *Empirical Software Engineering - An International Journal*, 1996. **1**(2): p. 133-164.

[4] Carver, J., *Architecture Reading Techniques*, in *Technical Reports*. 2005, Mississippi State University Department of Computer Science and Engineering.

[5] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J., *Documenting Software Architectures*. 2003: Addison-Wesley.

[6] Dennis, G., *Tsafe: Building a Trusted Computing Base for Air Traffic Control Software*, *Ph.D. Dissertation*, Department of Electrical Engineering and Computer Science. Massachusetts Institute of Technology. 2003

[7] Eden, A. H. and Kazman, R. "Architecture, Design, Implementation". *Proceedings of Software Engineering, 2003. Proceedings. 25th International Conference on*. 2003, 149-159.

[8] Erickson, R. L., Griffeth, N. D., Lai, M. Y., and Wang, S. Y. "Software Architecture Review for Telecommunications Software Improvement". *Proceedings of Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*. 1993, 616-620.

[9] Laitenberger, O., Atkinson, C., Schlich, M., and El Emam, K., "An Experimental Comparison of Reading Techniques for Defect Detection in Uml Design Documents". *Journal of Systems and Software*, 2000. **53**(2): p. 183-204.

[10] Laitenberger, O., El Emam, K., and Harbich, T. G., "An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective Based Reading of Code Documents". *IEEE Transactions on Software Engineering*, 2001. **27**(5): p. 387-421.

[11] Travassos, G., Shull, F., and Carver, J. "Reading Techniques for Oo Design Inspections". *Proceedings of 24th NASA Software Engineering Workshop*. 1999. Greenbelt, MD,

[12] Weyuker, E. J. "Predicting Project Risk from Architecture Reviews". *Proceedings of Software Metrics Symposium, 1999. Proceedings. Sixth International*. 1999, 82-90.