Applying Formal Methods to a Telecommunications System in a Commercial Setting

Andre Wong Marsha Chechik *

October 19, 1998

Abstract

Formal methods have been used mostly in safety-critical projects although some have industrial-strength implementations that can be used effectively in commercial settings. This paper describes a case study conducted in collaboration with Nortel to demonstrate the feasibility of applying formal methods to telecommunications systems. A lightweight formal method, SDL, was chosen by our qualitative evaluation to model a multimedia-messaging system. The model allowed us to locate errors in the software requirements document and to derive test suites, shadowing the existing development process and keeping track of a variety of productivity data. The success of our case study was in finding a suitable project where we were able to integrate a well-chosen formal method into the existing development process.

1 Introduction

For a long time, researchers and practitioners have been seeking different ways to improve productivity in the software development process. Within the software engineering research community, formal methods (mathematically-based languages, techniques or tools for specifying and verifying software systems [2]) have been advocated as one of the viable approaches. If high quality specifications are crucial to the success of system developments, it seems logical to apply formal specification techniques to the requirements for ensuring their completeness and consistency.

However, most successful applications of formal methods have been confined to safety-critical projects [3, 4, 7] where software correctness is the pivotal goal. In contrast, the software industry seeks practical techniques that can be seamlessly integrated into their existing processes and improve productivity; absolute quality is often a desirable but not

*Contact address: Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada. Email: {andre,chechik}@cs.toronto.edu.

crucial objective. As a result, most companies, such as the Canadian-based telecommunications company Nortel[1], opt to rely on manual inspections of natural-language specifications as the only requirement error detection technique, even though the results have been suboptimal. If the advantages of better quality specifications alone do not provide adequate justifications, more benefits can be obtained by leveraging the investment in the formalization process to other stages of the software lifecycle (i.e., generating code or test cases from the formal specifications), amortizing the cost of creating the specifications as well as obtaining measurable and more immediate productivity improvement.

Driven by the need to improve the software development phase, Nortel and the Formal Methods Laboratory at the University of Toronto have jointly proposed a research project[2] to investigate the feasibility of formal methods in a commercial setting. The goal of the project is to find means of using formal methods to improve productivity in various stages of the software lifecycle. Specifically, the emphasis is placed on deriving test cases from the formal model as the Nortel engineers have expressed concerns about the feasibility of code generation for their proprietary platform.

The project was organized as a hybrid quantitative and qualitative case study [8]. We began the study by proposing criteria for the system-selection process and conducting a qualitative formal method evaluation. Our model, which was constructed in parallel with the existing development process, was used to locate errors in the software requirements document and to derive test suites for identifying implementation errors. Throughout the study, a variety of productivity data were kept for comparison with similar information from the actual development process. The qualitative impacts of the formalization process were also noted.

The rest of the paper is organized as follows: Section 2 describes the system and formal method selection process and explains the rationale behind the choices. In Section 3, we discuss the formalization process. Section 4 presents a summary of findings from the study, and Section 5 draws conclusions based on the experience gained during this project.

2 The Selection Process

As not all systems are worth engineering [9], only some systems are worth formalizing. Therefore, it was crucial to select a system that had the best potential to benefit from the formalization process. We felt that the impact of our study would be more apparent if the system to be modeled

- had relatively stable requirements;
- was not too broad in scope but contained complex parts;

[1] Nortel, for the purpose of this paper, refers to the Toronto Multimedia Applications Center of Nortel Networks.

[2] This paper only highlights important aspects of the project. Readers are advised to consult [14] for more details.

- was relatively self-contained and had loose coupling with the underlying system;
- was reactive—almost all inputs resulted in outputs that accurately indicated the state of the system [10].

We worked with a group of Nortel engineers, consisting of developers from the design team and testers from the verification team, to choose a system that satisfied the criteria above. The software chosen was a subsystem in the Operation, Administration, and Maintenance (OAM) software of a multimedia-messaging system connected to a private branch exchange. The subsystem, called Service Creator in our paper, is a voice service creation environment that lets administrators build custom telephony applications in a graphical workspace by connecting predefined voice-service components together. Figure 1 illustrates a simplified graphical view of one such telephony application consisting of four components: the start, end, password-check, and fax-send components. When the application is activated, a call session begins at the *start* component, and a caller is required to enter a numerical password in order to retrieve a fax document from the *fax* component. The caller is directed to the *end* component if an incorrect password is entered. In both scenarios, the call session ends when the *end* component is reached. In our study, we analyzed the run-time behavior of 15 such components, described by an 80-page natural language specification.

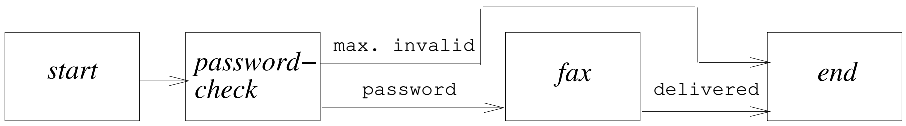

Figure 1: A simplified telephony application example.

The next step was to select a suitable formal method to be used in the feasibility study, which was just as important as the selection of an appropriate system. Not only did a suitable formal method ease the difficulty of the formalization process, but it also had a direct impact on the results of the study. Clearly, easily readable and reviewable artifacts as well as a simple notation were two basic requirements for a formal method to be usable in a commercial setting. Moreover, since one of the overall objectives was to amortize the cost of creating a formal specification, we began the evaluation by conducting a broad survey [13] of usable formal methods that provided support both for modeling and for testing. After we narrowed down our search to three tools (Telelogic SDT [11], Aonix [1] Validator/Req, and Teradyne TestMaster [12]), we structured our evaluation more formally as a qualitative case study [8] and refined our focus to assessing the remaining choices using additional evaluation criteria gathered from the Nortel engineers. The criteria comprised of factors, such as usability and smooth integrations, that were crucial to the use of formal methods in their environment. The tools were used to model the password-check component, and then were evaluated against the criteria based on our impressions of the tools and the models produced. Results from the assessment indicated that SDT met the criteria closer than the other tools.

3 Modeling the System

SDT is an integrated software modeling and testing tool suite that utilizes Specification and Description Language (SDL) [5], which is based on extended finite state machines (EFSMs), for behavioral modeling and Message Sequence Charts (MSCs) [6] for component-interaction specification. We modeled a Service Creator application as a 70-page SDL system, in parallel with the existing development process, and the voice-service components were formalized as separate SDL processes. A total of 23 signals were used in the SDL system; eight of them were external (used for communicating with the environment) and the rest internal. Furthermore, more than 120 MSCs, which could be treated as test cases, were derived semi-automatically from the SDL model.

The model and the experience gained from the formalization process allowed us to locate specification errors that escaped a series of manual inspections by Nortel engineers. The list below presents some problems found in the specification of the password-check component.

1) it was never mentioned whether voice prompts were interruptible, and no conventions for such behavior were defined;
2) lengths of various timeout events were not mentioned;
3) the maximum and minimum lengths of passwords were not defined.

The MSCs derived from the SDL model were used to identify errors in the implementation after Nortel engineers officially completed testing of the voice-service components. It was found that some of the specification errors, such as the third problem listed above, actually propagated to the implementation, i.e. a malicious caller could crash the system by entering an abnormally long password within the password-check component. This was a concrete example of a requirement omission becoming a critical implementation error.

4 Findings

In this section we present quantitative and qualitative results of the study. In addition, we present limitations of our work, hoping that this information would allow the reader to reach accurate and unbiased conclusions about the generalization of our results.

4.1 Quantitative Results

The entire modeling process, which consisted of activities such as understanding and formalizing the specification as well as deriving and executing test cases, took about two person-months to complete. During this period, we kept track of a variety of productivity data in the study (column two of Table 1) for comparison with similar information from the existing development process (column three).

Productivity Data	Study	Existing Process
Time to model (person-days)	11	N/A
Number of specification errors reported	56	N/A
Number of test units	269	96
Test unit creation rate (test units/day)	30	21
Number of implementation errors identified	50	23

Table 1: Productivity Comparison.

Since the sizes of test cases varied greatly, we did not use a "test case" as the unit of comparison for testing-related data. Instead, we counted test units, the smallest externally visible functionality of the system, in each test case to ensure a fair comparison. Highlights from the table are summarized below.

- The *time to model* value included only the time used for modeling the SDL processes. It was impossible to draw conclusions from it, since the modeling task did not have an equivalent in the actual development process. However, we felt that the 11 person-days was a fairly small amount of time when compared to the four person-months it took to develop the corresponding specification.
- More test units could be derived from the SDL model (which translated to better test coverage) at a faster rate, possibly because the model eased the creation task by providing a more in-depth understanding of the system as well as a better sense of completeness. One other reason for the difference in the quantity is that the test units from Nortel were sometimes vaguely specified; the missing details contributed to a decrease in the number of test units reported.
- Of the 50 implementation errors identified, 18 were avoidable errors that could be linked to problems in the specification. Moreover, the number of implementation errors identified in the study was two times larger than that of the existing development process, possibly due to better test coverage.

4.2 Qualitative Observations

While all the quantitative data was in favor of the use of the formal modeling, it was clear that these results alone constituted only a part of our findings. Some observations from the formalization process that were not evident from Table 1 are discussed below.

- The most frequent complaint from test engineers is that the missing information in the specifications often complicates the task of test case creation. The SDL models encourage and assist developers in stating system requirements more precisely and completely, which should allow testers to create better quality (e.g., more detailed and with expected results more clearly defined) test cases and reduce the time needed for test case creation and execution. Developers should also benefit from the more complete specifications in the design and implementation phases. This is one area where SDL can potentially improve the development process.

- A common problem with most formal methods is that they require requirements to be specified and reasoned about in formal logic—skills that most Nortel engineers do not possess. SDL reduces the problem to possibly a minimum by only requiring them to construct an EFSM using a graphical and user-friendly notation. Since it is still difficult for us to assess if SDL addresses the usability issue adequately, we plan to conduct further studies in this particular area.
- As with any other formal specification techniques, a successful integration of SDL with the development process requires firm commitments from the entire development team. For instance, the developers must ensure that the SDL model is always kept consistent with the system requirements and the code, e.g. last-minute changes in the design and implementation are propagated back to the model. Testers also need to ensure that their test cases always reflect the model accurately. While this is possibly one of the biggest obstacles in applying a formal modeling technique, the advantages provided by SDL should justify the extra effort.

4.3 Limitations

Although we believe that our findings, as a whole, demonstrated the usefulness of SDL in formalizing the Service Creator, we cannot infer that all of our results will apply directly to the actual development process as results from case studies are usually not generalizable [15]. In our study, both the formal method and the system to be modeled were carefully chosen to maximize the impact of the formalization process. We might not obtain the same results if a different formal method was applied to some other type of systems. Also, since the study was performed in parallel with the actual development process without the active involvement from the Nortel engineers, many activities of the study could not be used for comparison, which made some of the analyses difficult or impossible, e.g. the number of specification errors identified in the formalization process could not be compared because there was no equivalent in the existing development process.

Furthermore, our prior experience in formal modeling might have also reduced the time and difficulty of the formalization process. While we felt that the process was straightforward, our results might not apply directly because most Nortel engineers had no exposure to formal methods. Yet, we believe that with appropriate training and having an SDL expert on premise, these problems could be overcome.

5 Conclusions and Future Work

In this case study, we formalized the behavior of a multimedia-messaging system in a commercial setting. While we identified quantitative and qualitative improvements of the formalization process over the existing development methodology, the success of the study was also in finding a suitable system and a well-chosen formal method with which requirements could be formalized effectively and economically. Among the outstanding issues, usability was an important one that had not been fully addressed. To reinforce our confidence in the results of our study, we plan to conduct further experiments with the Nortel engineers to investigate if the chosen formal method could be applied by them to obtain comparably good results.

6 Acknowledgment

The authors would like to thank Albert Loe, Steve Okun, Avinash Persaud, and Shawn Turner of Nortel for their technical assistance and continual support throughout the study. The study was supported in part by Nortel and Ontario Graduate Scholarship (OGS).

References

[1] Aonix. "Aonix Home Page". `http://www.aonix.com`, September 1998.

[2] E.M. Clarke and J. Wing. "Formal Methods: State of the Art and Future Directions". *ACM Computing Surveys*, 28(4):626–643, December 1996.

[3] J. Crow and B.L. Di Vito. "Formalizing Space Shuttle Software Requirements". In *Workshop Formal Methods in Software Practice*, San Diego, California, January 1996.

[4] Mats P.E. Heimdahl. "Lessons from the Analysis of TCAS II". In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'96)*, San Diego, CA, January 1996.

[5] ITU-T. "ITU-T Recommendation Z.100: Specification and Description Language (SDL)". *ITU-T*, 1993.

[6] ITU-T. "ITU-T Recommendation Z.120: Message Sequence Chart (MSC)". *ITU-T*, 1993.

[7] Paul K. Joannou. "Experiences from Application of Digital Systems in Nuclear Power Plants". In *Proceedings of the Digital Systems Reliability and Nuclear Safety Workshop*, Rockville, Maryland, 1993.

[8] Barbara Ann Kitchenham. "Evaluating Software Engineering Methods and Tools. Part 1". *ACM SIGSOFT Software Engineering Notes*, 21(1):11–15, January 1996.

[9] Roger S. Pressman. "Can Internet-Based Applications be Engineered?". *IEEE Software*, 15(5), September 1998.

[10] Tony Savor and Rudolph E. Seviora. "Toward Automatic Detection of Software Failures". *IEEE Computer*, 31(8):68–74, August 1998.

[11] Telelogic. "Telelogic SDT Home Page". `http://www.telelogic.com/solution/tools/sdt.asp`, September 1998.

[12] Teradyne. "TestMaster Home Page". `http://www.teradyne.com/prods/sst/ssthome.html`, September 1998.

[13] Andre Wong. "The Diary of the Formal-Method Survey". `http://www.cs.toronto.edu/~andre/progress.html`, September 1998.

[14] Andre Wong and Marsha Chechik. "Applying Formal Methods to a Telecommunications System in a Commercial Setting'. CSRG Technical Report 377, Department of Computer Science, University of Toronto, 1998.

[15] Marvin V. Zelkowitz and Dolores R. Wallace. "Experimental Models for Validating Technology". *IEEE Computer*, 31(5), May 1998.