Formal Methods When Money is Tight

Marsha Chechik Andre Wong *

March 14, 1999

Abstract

Formal methods have been shown to improve the quality of software, but they are seldom if ever used outside the safety-critical system domain. Typically, the initial cost of creating formal requirements specifications is perceived to be prohibitively large while not necessarily guaranteeing future benefits. In this paper we discuss ways of tailoring formal methods to suit current economic realities.

1 Introduction

In the profit-driven commercial software industry, the reality is that companies can deliver products with less-than-perfect quality but must do so in a timely manner to stay competitive; this is true even for traditionally-conservative telecommunications companies. Although the level of tolerance to errors varies between software for office productivity and that for public branch exchanges, it is generally true that absolute software quality is a desirable but not a crucial objective. Shorter time-to-market, lower development costs, and higher productivity typically have a greater impact on profitability. As a result, there is an ever-growing emphasis on obtaining immediate and visible rather than long-term benefits.

In contrast, most formal methods research traditionally focuses on achieving the higher quality of software. After an initial investment into creating formal specifications, applying formal methods can result in better understanding of systems, lower code complexity [16], and higher quality software artifacts (e.g. requirements and implementations) [5]. These benefits logically lead to a reduction in the time and cost of the development, although they are perceived to be long-term. Coupled with limitations such as generally complex notations, a lack of methodology, reduced benefits when requirements are not stable (and most of the software development is now iterative, having requirements that inevitably change), and the need to employ experts with a solid mathematical background, it is not surprising that formal methods have not yet achieved a wide adoption in the commercial software industry. Practical issues such as usability and maintainability are also seldom addressed.

Without a substantial evidence of the promised benefits resulting from relevant case studies, it is inconceivable that industry would adopt a relatively radical technique requiring large up-front investments and having incomprehensible notations [7]. In the 1989 survey of practitioners

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

working in the telecommunications industry where formal description techniques have had relative success, 84% and 50% of the subjects believed that such techniques were not viable and would not become viable in the next three to five years, respectively [15]. 10 years past since that study, and we feel that it is time to investigate ways to apply formal methods in the profit-driven commercial development. For the purposes of this paper, commercial systems are defined as large software systems that do not require a level of assurance necessary in safety-critical systems and are developed for profit. Observations made below are based on our experience conducting a formal methods feasibility study in collaboration with Nortel Networks (Nortel) [21].

2 Making Formal Methods Economical

The major challenge in making formal methods economical is to find ways to obtain immediate savings to compensate for the up-front cost. One possible way is to leverage the investment in other stages of the software lifecycle, in particular in testing. An ability to derive and automatically execute black-box testcases significantly shortens the testing phase, thus delivering immediate benefits. Not only does this amortize the cost of creating the specifications, but the productivity improvements also become more quantifiable and visible. Of course, the total decrease in the development cycle is only achievable if the formalization can be done fairly inexpensively.

2.1 Creating Formal Models

From the overwhelming popularity of UML [2], it is evident that there is a large demand for techniques that provide a better understanding of software systems. UML's small up-front cost, flexibility, fairly simple and standardized notation, and industrial tool support have almost made it the de facto modeling language in industry. The key to a wider adoption of formal methods is to tailor them to mirror some of UML's successes. Given that we do not place a large emphasis on verification, or reasoning about software systems, we will use the term *formal description techniques* (FDTs) instead of formal methods.

If FDTs are to be used in industry, they should have the characteristics summarized below.

- Application of FDTs should be allowed to be evolutionary, since this presents less risk to companies, allowing developers to experiment with them and revert to the old techniques if the expected benefits are not attained.
- Application of FDTs should fit into the existing environment, i.e., be used with only minor changes to the existing development methodology. For example, if changes are made in the specification notation for requirements, then design and testing should remain intact. The technique should allow to retain standard practices like review meetings, version control, etc.
- FDTs should be flexible, allowing to specify artifacts at different levels of abstraction, and allowing to model different views of the system: system states, object relationships, etc.
- FDTs should be supported by industrial-quality tools.

- FDTs should have an easy to use and review notation. Research has shown that difficult and inconvenient techniques are simply abandoned [18]. The notation should allow the engineers to maintain the models easily as requirements change. In fact, *usability of FDTs* is probably one of the most important factors in achieving their wider acceptance. As we investigated opportunities to effectively integrate FDTs into the Nortel's development domain, Nortel engineers specifically told us that it is essential that the use of FDTs require no deep understanding of logic and is accessible to people with less technical skills, e.g., training and management.

- Partial application of FDTs should lead to achieving partial benefits. Nortel engineers specifically mentioned that the need to go into too much detail and not see any results before all the details are filled in, was the biggest reason why previous attempts to bring formal modeling techniques, specifically STATEMATE [8], into their company have failed.

Many of the above criteria have been proposed by advocates of *light-weight formal methods* [6, 12, 14]. These criteria ensure that the application of FDTs is feasible in the profit-driven setting.

2.2 Testing

The increase in the time/cost of the requirements phase associated with creating a formal model is easily leveraged during testing. Precise semantics of FDTs allows to automate the generation of black-box testcases. Alternatively, such testcases can be captured during the symbolic execution of the model. These testcases accurately reflect the requirements, allow to achieve better coverage than those created manually, and allow to maintain a good traceability between testcases and requirements. Maintaining this traceability is often non-trivial as requirements change [20], and coupling requirements and testing makes it easy to determine which testcases should be part of the regression test suit and which need to be redeveloped.

2.3 Verification

Since the majority of industrial software is not in the safety-critical domain, the emphasis should be on reducing the cost of software rather than increasing its quality. Still, even without rigorous verification, FDTs can improve the quality of software without increasing its cost. First saving, as many researchers pointed out, comes from the rigor applied on the requirements level. Second comes from high-quality testing. Finally, it may often be desirable to verify simple invariants or some safety or security policies. With the growing use of model-checking techniques, such properties can often be verified quickly and effectively. If this verification is virtually free, it will be adopted.

3 Our Results

We applied some of the principles mentioned above to the study that we have recently conducted at Nortel [21, 20]. We started by looking for a tool [19] that allowed combining formal modeling with generation (not necessarily automatic) of testcases and could be applied in the telecommunications domain, and chose SDT [17] – an integrated software modeling and testing tool suite that utilizes the Specification and Description Language (SDL) [10]. Using SDT, we formalized a part of the Operation, Administration and Maintenance (OAM) software and used the model to derive testcases in the form of message sequence charts [11]. This project was done in parallel with Nortel's regular development. Table 1 summarizes the productivity data collected as part of our study. 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. More information about this study appears in [21].

Productivity Data	**Study**	**Existing Process**
Time to model (person-days)	11	N/A
Time to inspect (person-days)	N/A	50
Number of specification errors reported	56	N/A
Number of test units	269	96
Time to create test units (person-days)	7	7
Time to execute test units (person-days)	5	14
Number of implementation errors identified	50	23

Table 1: Productivity comparisons.

In this exercise, SDL allowed us to achieve a thorough understanding of the system with a relatively small cost of formalization, and enabled us to uncover many errors in the specifications and the code. Our project did not involve any verification, and we note that it is not essential to use a modeling language with a fully-defined formal semantics to achieve immediate and measurable benefits. However, the language should give adequate semantics for testcase generation.

Of course, not all projects can result in such a clear productivity improvement, and yet we believe that this exercise shows the feasibility of using FDTs in an economical way if they are coupled with testcase generation capabilities.

4 Future Work

Our survey of formal tools and techniques [19] showed that most of them cannot be applied in the manner advocated in this paper: they support either modeling or testing but not both. We were lucky to find a tool that gives some support for both, but its may not be applicable outside the telecommunication domain. So, in order to move ahead on applying formal methods to commercial projects, the community needs to create/improve tools that support inexpensive modeling and combine it with testing, and conduct many feasibility studies to show that the techniques work.

Although there has been a number of applications of formal methods outside the safety-critical domain, e.g. [3, 9, 4], mostly these were one-time efforts done by formal methods experts. Researchers also do not fully understand the impact of using formal techniques for systems with rapid changes in requirements. It is important to conduct more studies that capture the real environment conditions more closely by attempting parallel development and doing a thorough cost/benefit analysis. Such studies also lead to improving tools to make them more usable within the industrial business process. For example, Ontario Hydro developed a tool to store tabular requirements for the shutdown system of the Darlington Nuclear Generation Station [1] and generate a text document on demand. Changing formats of the tables and the instability of the tool made it more convenient to keep the requirements in an MS-Word document and built a parser to generate a model from it. This way a snap-shot of the formalization work was always available for printing and review [13].

Unfortunately, tools and case-studies are connected: adequate tools must exist before large field studies are effective, but studies can be fully effective only if the tools are of industrial-quality. The advancement of tools and the increase in the number of relevant studies that make use of such tools will happen slowly and in parallel. We feel strongly that this "parallel advancement" should be pursued.

References

[1] G. Archinoff, R. Hohendorf, A. Wassyng, B. Quigley, and M. Borsch. "Verification of the Shutdown System Software at the Darlington Nuclear Generation Station". *International Conference on Control and Instrumentation in Nuclear Installations*, May 1990.

[2] Grady Booch, Invar Jacobson, and James Rumbaugh, editors. *"UML Semantics (Version 1.1)"*. UML Partners Consortium, 1997.

[3] D. Brownbridge. "Using Z to Develop a CASE Toolset". In *Proc. Z Users Workshop*, pages 142–149, London, U.K., 1989. Springer-Verlag.

[4] M. Buchi. "The B Bank: A Complete Case Study". In *Proceedings of the Second International Conference on Formal Engineering Methods*, December 1998.

[5] J. Crow and B.L. Di Vito. "Formalizing Space Shuttle Software Requirements: Four Case Studies". *ACM Transactions on Software Engineering and Methodology*, 7(9):296–332, July 1998.

[6] R.E. Davis and R.L. Danielson. "Practically Formal Methods". In *Proceedings of International Conference on Software Engineering: Education and Practices*, pages 168–175. IEEE Computer Society Press, January 1996.

[7] Anthony Hall. "Seven Myths of Formal Methods". *IEEE Software*, pages 11–19, September 1990.

[8] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems". *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[9] Jonathan P. Hoare. "Application of the B-Method to CICS". In M. G. Hinchey and J. P. Bowen, editors, *Applications of Formal Methods*, pages 97–124. Prentice Hall International Series in Computer Science, 1995.

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

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

[12] Daniel Jackson and Jeannette Wing. "Lighweight Formal Methods". *IEEE Computer*, April 1996.

[13] Paul Joannou. Private communication, 1998.

[14] Cliff B. Jones. "An Invitation to Formal Methods: A Rigorous Approach to Formal Methods". *IEEE Computer*, 20(4):19, April 1996.

[15] M. T. Norris and S. G. Stockman. "Industrialising Formal Methods for Telecommunications". In C. Ghezzi and J. A. McDermid, editors, *Proceedings of the 2nd European Software Engineering Conference*, Warwick, U.K., September 1989. Springer-Verlag.

[16] Shari Lawrence Pfleeger and Les Hatton. "Investigating the Influence of Formal Methods". *IEEE Computer*, 30(2):33–43, February 1997.

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

[18] Debora Weber-Wulff. "Selling Formal Methods to Industry". In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings of FME'93: Industrial Benefit of Formal Methods, First International Symposium of Formal Methods Europe*, pages 671–678, Odense, Denmark, April 1993. Springer-Verlag.

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

[20] Andre Wong. *"Formalizing Requirements in a Commercial Setting: A Case Study"*. M.Sc. thesis, University of Toronto, Department of Computer Science, Toronto, ON, Canada, 1999. (in preparation).

[21] Andre Wong and Marsha Chechik. "Formal Modeling in a Commercial Setting: A Case Study". (submitted for publication), February 1999.