Formal specification techniques in the commercial development process*

J S Fitzgerald† P G Larsen‡

Abstract

This paper describes the lessons learned from an application of formal specification techniques in the development of a security-critical system within a UK company. The authors advocate the gradual introduction of formal methods, beginning with an appreciation of existing development processes, and discuss the rôle played by non-software professionals, executable specifications, formal proof, training and tool support in this and future projects.

Keywords: Applications of Formal Methods; Tool Support; Software Process; Training

Introduction

If formal methods are to be more than an academic toy, experts in the field must develop a fruitful dialogue with commercial systems developers. We believe the best way to begin this is to discuss formal methods with engineers on their own terms, and then build on their experience. Professionals are generally open to techniques which will help them do a better job, but such techniques cannot be introduced at the expense of existing good practice. The approach we espouse is to begin by incorporating a limited amount of formalism in an existing development process and go on to increase rigour where this is felt by the engineers to be beneficial.

We begin this position paper by describing our most recent experience introducing formal methods to part of British Aerospace. Using this as an example, we state some of the lessons we feel should be learned about applying formal methods in an industrial development process. Each section ends with a summary of the point we are making.

Experience at British Aerospace

Our experiment in developing a security-critical system using VDM-SL with CASE technology is documented elsewhere ([1], [2], [3], [4], [5]). British Aerospace (Systems &

*Presented at the ICSE-17 workshop on Formal Methods Application in Software Engineering Practice, April 1995, Seattle, USA

†The Centre for Software Reliability (CSR), Bedson Building, University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU, UK

‡IFAD, Forskerparken 10, DK-5230 Odense M, Denmark

Equipment) Ltd. (BASE) have been developing a secure system component, a trusted gateway, to meet high levels of assurance as part of their process improvement work. The main techniques and tools used in the development are: requirements tracing managed by RTM[1]; design representation using data flow diagrams supported by Teamwork[2]; informal pseudo-code and implementation in Visual C++. This is matched with a parallel development of the trusted gateway by a separate team of engineers using the same techniques, plus some formal specification in VDM-SL [6] supported by the IFAD VDM-SL Toolbox. This forms the basis of a comparative study of the costs and benefits of applying simple formal techniques.

The systems and software engineers at BASE were taught VDM-SL and then given freedom in choosing how to use it within their development process, with the proviso that formal description of the system security policy was mandatory in order to meet the required security evaluation level. In the event, the engineers decided to write formal definitions of the data types used in the Teamwork model, putting these in the data dictionary; and to give formal specifications of the operations associated with some of the processes in the model, putting these in the relevant Teamwork process specifications. A number of other aspects of the system besides the security policy were described in VDM-SL.

This experiment is certainly no "clinical trial" of formal specification, and we would agree with Fenton's observation [7] that there is surprisingly little empirical evidence of the benefits of new software engineering techniques, formal methods included. However, we have made some measurement of costs and benefits in the BASE project, if only in a coarse way, with the result that BASE will investigate the closer integration of formal specification into system design activities. One approach under consideration is to make formal definition of data types the norm for certain projects, with formal specification of operations used only where it is felt to be cost-effective, usually in critical or complex processes.

Although we refer mainly to the trusted gateway project, a number of other projects also inform the view expressed here. These include the application of formal modelling to the definition of monitoring and control systems in processing and storage plants (with British Nuclear Fuels, BNFL) and development of techniques for specification of avionic control systems (with British Aerospace Defence [8]). In most of these projects, we have been working with engineers who are are new to formal methods. In fact, we have found little of the industrial prejudice responded to in [9] and [10].

Software is specified by systems engineers

The Software 2000 [11] working party noted that software is increasingly being developed for embedded applications[3]. One consequence of this is that many formal specifications will be written by systems engineers, who do not often have a traditional computing background. This was the case in the BASE project, where a systems engineer led the

[1]RTM is a Registered Trademark of GEC Marconi, Addlestone, Surrey

[2]Teamwork is a Registered Trademark of Cadre Technology Inc.

[3]0.5 MByte in a television and 2kByte in an electric shaver, according to Remi Bourgonjon in [11]!

initial design and specification phase, while a software engineer took charge of more detailed program design.

Systems engineers in both BASE and BNFL have had little trouble understanding the principles of formal specification, even if the technology's roots in computer science are unfamiliar to them. Indeed, we found that the BASE systems engineer had less trouble reading and writing specifications at the appropriate level of abstraction than his software colleague.

Point: Formal techniques will not be used only by software engineers. Skills of abstraction are especially valuable, and may be found more readily outside the traditional group of software specialists.

The development process is rarely negotiable

Many organisations have made substantial investments in their development processes, often to achieve external certification to ISO9000 and related standards. In introducing such organisations to formal techniques, we are not in a position to require radical changes in development procedures in the short term. This has to be done gradually as part of a process improvement plan.

In the BASE project, formal specification supplemented existing techniques. Its use in early design phases added to project costs, since more material (better analysed requirements, formal process specifications etc.) was produced[4]. Over the longer term, it will be possible to identify and eliminate activities which have become redundant. For example, in the software design phase of the BASE development, the engineer using formal techniques was writing both the traditional pseudo-code *and* VDM-SL specifications for operations.

Because of the need to work within a predefined development process, formal specification must complement established techniques and tools. In BASE, rather than suggest an integrated method based on a formal semantics to data flow diagrams (e.g. [12]), a more ad hoc approach was taken: formal specifications were written as comments in the data dictionary and process specifications of the Teamwork diagrams. It was a comparatively simple task to build an interface from Teamwork to the IFAD VDM-SL Toolbox to facilitate parsing, type checking and testing of the specification. This linkage from the CASE tools to the VDM support tools was essential in ensuring that formal methods could fit in with existing mechanisms for tracing the links between designs and the database of requirements clauses.

Point: Listening to our industrial colleagues made us realise that we cannot change development processes to suit the introduction of formal methods: we should apply a limited amount of formality as a supplement to existing practice, then gradually suggest areas where existing practice can be changed.

[4]The number of person-hours used for later software engineering work was lower or about the same in both development streams.

Tools have a strong influence on specification style

The value of executability in specifications has been the subject of debate (e.g. [13], [14]). The BASE project used a tool set which permits specification testing, so it was not surprising that the systems engineer preferred to write executable specifications over non-executable, and perhaps more abstract, versions. Moreover, an explicit specification style was used rather than the pre-/post-condition style also available in VDM-SL. From the point of view of the engineers, there were a number of good reasons for this approach:

- An executable specification working on the abstract types in an early stage of the development process allows specific tests to be included in the system test plan. These are certainly not comprehensive, but they do help the engineer record extreme or exceptional cases for the benefit of later design work.

- An executable abstract specification eases validation against informal customer requirements. Test cases suggested by the customer can be quickly checked against the specification.

The "V" life cycle model used at BASE emphasizes the production of test plans at each stage in system development, with the customer acceptance test as the final arbiter of successful delivery of a product. This has led one industrial colleague to question the value of doing any proof-based refinement from an abstract specification, as long as the customer regards the acceptance test as the measure of success.

This enthusiasm for explicit specification has a price:

- Validation of the abstract specification is limited to the test cases. The style of presentation of an implicit specification may make it easier to see by inspection or rigorous argument that critical properties of the system have been captured.

- Explicit specification early in the development process tends to bias later development activities towards expanding the "code" structure of the more abstract operation specifications. The software designs at BASE reflect this, with abstract specifications seen as higher level designs rather than descriptions of desired behaviour.

One way forward is to develop techniques for generating test cases from implicit (possibly non-executable) specifications (see [15], [16]). Such test cases can then form the basis for validation of explicit versions. The implicit specifications could continue to be the definitive ones handed on to later development phases, while developers can choose to pass on explicit versions or not, as appropriate. A disadvantage is the need to keep two versions of the specification in step during development.

Another important approach is direct symbolic execution of implicit operation specifications [17], although this can suffer from combinatorial explosion. Research is required to make this a more tractable option for model-oriented specifications, perhaps exploiting formula rewriting packages used in mathematical applications.

Point: Commercial systems developers have found the ability to execute specifications a great benefit. We would like to move towards support for less explicit specification

styles, and see the symbolic execution and automated generation of test cases as avenues for useful future research.

Proof is not often a high priority

In the spirit of introducing formality gently, the BASE project did not attempt to apply proof: formal or rigorous reasoning were not felt to be cost-effective given the project goals.

In projects where a safety or security case is to be produced, proofs may add value to the product. Work with BNFL suggests that some element of proof could be incorporated into design reviews or inspections. If the proofs are conducted at a level of rigour appropriate to the importance of the conjecture and the available resources, this may prove cost-effective in revealing subtle errors of specification consistency and fidelity to requirements. Rushby notes ([18], pg 96) that the separation of proof-related projects at verification levels and proof-free projects at requirements modelling levels may be historical accident. We would welcome commercial application experiments measuring the cost-effectiveness of applying proof at early stages.

Our overriding impression is, however, that commercial pressures work against application of proof where it is not mandated, especially given the shortage of relevant skills among systems engineers.

Point: Our industrial collaborators indicate that testing of specifications is much more cost-effective than proof at the present time. We would like to see rigorous proof introduced in the much longer term, but would first welcome more evidence about its cost and value at early stages in the development process.

Timescales

Tradition has it that academic researchers and industrial users work on different timescales. In industry there are always hard deadlines to be met, so there is rarely time to mull over and redevelop a specification (or a program), as could be done in the academic world.

It is important to take product development time into account when considering how formal techniques might be applied. In BASE, where the time to market averages one to two years, and engineers are often working to close deadlines, a technique which saves even a couple of days of redesign and error correction is a major improvement. In other domains, such as aircraft, the time to market can be as much as twenty years. This makes it harder to introduce formal techniques at the beginning of a new product's development process, with the result that one often has to deal with post-facto specification and reverse engineering.

Point: In order to get formal methods applied industrially it is necessary to work with their timescales instead of in an "ideal" academic world.

Training

So far we have given VDM courses to a number of of industrial organisations including BASE (UK), ICL Enterprise Engineering (UK), VTT (SF), and CISI (F). Our experience has been that one week of training gives ordinary engineers enough proficiency to use a formal technique for system specification. Most of these engineers had no knowledge about formal methods prior to the courses, and did not have a particularly strong background in mathematics. The learning curve seems to be heavily influenced by the availability of tool support enabling the students to get computer feedback on their work. After such a one-week course, our experience is that the students, with assistance from experts, are able to develop specifications for reasonably complex systems right away. The ability to test specifications in some way seems to provide a motivating factor in this learning phase.

Point: Tool support during training has a significant effect. As little as one week of training seems to be sufficient to make ordinary engineers capable of producing formal specifications.

Conclusions

This workshop is intended to assess the current state of formal methods in software engineering practice. As our starting point, we would say that it is now possible to apply formal techniques directly in commercial contexts provided this is done gradually and with full regard for current practice.

In this paper, we have listed the main lessons learned from one project. The most important points which we will carry forward to future work are:

- that we should not assume a traditional computing background among engineers who will specify software;
- that we should introduce formalism within existing development processes, and suggest gradual, considered change;
- that support for specification validation and testing is vital, with a longer-term research towards support for greater abstraction, including test case generation and symbolic execution;
- that professional-quality, not amateur academic, tool support makes a significant difference to training and speed of application of formal methods;
- that we require more evidence on the costs and benefits of applying rigorous reasoning techniques at earlier development stages;
- an awareness of researchers' and commercial developers' differing timescales.

However essential we feel formal methods are, the industry continues to develop software which usually works, even in critical applications. Some good development practices have emerged which formal techniques should not supplant, but improve. Future work should gather evidence of this improvement, determine the ways in which formality can be most effectively employed, communicate directly with professionals and not become too diverted into arguments with detractors. For a commercial systems developer, taking up formal methods will be an investment. We should work to ensure that the consequent product and process improvement is worth the price.

Acknowledgements The authors are grateful to systems developers in the companies with which we have worked, especially Tom Brookes of British Aerospace; and to Cliff Jones of Manchester University and Paul Mukherjee of Royal Holloway, London, for discussions on executability and validation. JSF is grateful to the United Kingdom Engineering and Physical Sciences Research Council for support under a Research Fellowship.

References

[1] T. M. Brookes, M. Green, J. S. Fitzgerald, and P. G. Larsen. A Comparison of the Conventional and Formal Design of a Secure System Component. In *Proceedings of the Nordic Seminar on Dependable Computing Systems 1994*. Technical University of Denmark, 1994. Preliminary version in the FACS Europe Newsletter, Vol. 1, No. 2, 1994.

[2] J. S. Fitzgerald, T. M. Brookes, M. Green, and P. G. Larsen. Formal and Informal Specifications of a Secure System Component: first results in a comparative study. In Maurice Naftalin, Tim Denvir, and Miquel Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 35–44. Springer-Verlag, 1994.

[3] J. S. Fitzgerald, P. G. Larsen, T. M. Brookes, and M. A. Green. Developing a Security-critical System using Formal and Conventional Methods. In Michael G. Hinchey and Jonathan P. Bowen, editors, *Applications of Formal Methods*, International Series in Computer Science. Prentice Hall, 1995. To appear.

[4] J. S. Fitzgerald. ESSI Project ConForm: Home Page. World wide web at URL http://www.cs.man.ac.uk/fmethods/projects/essi-ConForm.html, 1994.

[5] Tom Brookes Peter Gorm Larsen, John Fitzgerald and Mike Green. Formal modelling and simulation in the development of a security-critical message processing system. In *Anglo-French workshop on Formal methods, Modelling and Simulation for System Engineering*, February 1995.

[6] ISO. Document Number ISO/IEC JTC1/SC22/WG19/N-20 Information Technology Programming Languages – VDM-SL First Committee Draft Standard CD 13817-1, November 1993.

[7] N. Fenton. How Effective are Software Engineering Methods? *"Controversy Corner" in the Journal of Systems and Software*, 22:141–146, 1993.

[8] L. M. Barroca, J. S. Fitzgerald, and L. Spencer. The Architectural Specification of an Avionic Subsystem. Technical Report 94/19, Dept of Computing, The Open University, 1994. Extended version of the paper to appear in Proceeedings of the First International Workshop on Industrial Formal Techniques, Boca Raton, Florida, IEEE Press 1995.

[9] J. A. Hall. Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19, 1990.

[10] Jonathan P. Bowen and Michael G. Hinchey. Seven More Myths of Formal Methods: Dispelling Industrial Prejudices. In Maurice Naftalin, Tim Denvir, and Miquel Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 105–117. Springer-Verlag, 1994.

[11] Brian Randell, Gill Ringland, and Bill Wulf, editors. *Software 2000: a View of the Future*, D2D, Cavendish Road, Stevenage SG1 2DY, UK, 1994. ICL and The Comission of the European Communities. Report No. P4265.

[12] P. C. Fencott, A. J. Galloway, M. A. Lockyer, S. J. O'Brien, and S. Pearson. Formalising the Semantics of Ward/Mellor SA/RT Essential Models using a process Algebra. In Maurice Naftalin, Tim Denvir, and Miquel Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 681–702. Springer-Verlag, 1994.

[13] I. J. Hayes and C. B. Jones. Specifications are not (necessarily) executable. *Software Engineering Journal*, 4(6):320–338, November 1989.

[14] Norbert E. Fuchs. Specifications are (Preferably) Executable. *Software Engineering Journal*, pages 323–334, September 1992.

[15] Jeremy Dick and Alain Faivre. Automating the Generation and Sequencing of Test Cases from Model-Based Specifications. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods*, pages 268–284. Formal Methods Europe, Springer-Verlag, April 1993. Lecture Notes in Computer Science 670.

[16] Hans-Martin Hörcher and Jan Peleska. The Role of Formal Specifications in Software Test. In *Tutorial Programme FME'94*, October 1994.

[17] Ralf Kneuper. Symbolic Execution: a Semantic Approach. *Science of Computer Programming*, 16:207–249, October 1991.

[18] J. Rushby. Formal Methods and the Certification of Critical Systems. Technical Report CSL-93-7, Computer Science Laboratory, SRI International, December 1993.