Lessons from the Application of Formal Methods to the Design of a Storm Surge Barrier Control System

Michel Chaudron[1], Jan Tretmans[2] & Klaas Wijbrans[1]

[1]CMG Public Sector B.V., Division Advanced Technology,
P.O. Box 187, 2501 CD The Hague, The Netherlands
{michel.chaudron, klaas.wijbrans}@cmg.nl

[2] University of Twente, Department of Computer Science,
Formal Methods & Tools group,
P.O. Box 217, 7500 AE Enschede, The Netherlands
tretmans@cs.utwente.nl

Abstract We describe the experience of the industrial application of formal methods in the development of a mission critical system. We give a description of the system that was to be developed and the methods that were employed to realize the high level of reliability that was required. In this paper we will describe which formal techniques were used, how these techniques were used, the influence of formal methods on the development process and recommendations for managing the use of formal methods.

1. Introduction

The control of more and more processes that are critical to businesses and society are trusted to computer systems. This calls for methods for the engineering of systems with very high quality requirements such as reliability, safety and security. The developments in this area are aimed at improving the quality of the engineering process (such as ISO 9001, Capability Maturity Model (CMM) [Pau94]) as well as at improving the quality of the product (such as formal methods).

The application of formal methods in industrial software development projects is gaining maturity, but still raises a number of technical and managerial questions to which no definitive answers have been given. In this paper we will touch upon a number of technical and managerial questions related to the use of formal methods. These questions were encountered in the course of the engineering of a safety-critical system in a fixed-time, fixed-price project where the project members had had prior experience with software engineering, but hardly any experience with formal methods. The experiences described in this paper are based on interviews with the people involved in the development of the system. The issues raised in these interviews were mainly non-technical and are concerned with how engineers and managers experienced the use of formal methods. No attempts to quantification or measurements are made. For technical issues with respect to the formal techniques used we refer to [Kar97, Kar98].

J. Wing, J. Woodcock, J. Davies (Eds.): FM'99, Vol. II, LNCS 1709, pp. 1511-1526, 1999.

This paper is organized as follows: in section 2 we describe the context of the system that was to be built, the systems high quality requirements and the approach used in the engineering of the system. In section 3 we describe our evaluation of the use of formal methods in this process. The lessons learned from this project are described in section 4 and conclusions in section 5.

2. Case Description: The BOS System

BOS (Dutch: *Beslis* & *Ondersteunend Systeem*, i.e., Decision & Support System) is the system that controls the storm surge barrier in the Nieuwe Waterweg near Rotterdam. BOS was developed by CMG, division Advanced Technology. In this section we describe the storm surge barrier that BOS has to control. This context of the project explains the very high requirements that were put on the reliability and safety of the BOS system. Because of the special nature of this system, a dedicated system engineering process was devised for the project. This dedicated engineering process was ISO 9001 certified.

2.1 The Battle with the Sea

The Netherlands are located in a low delta by the sea, into which important rivers such as the Rhine and IJssel flow. The history of The Netherlands has been shaped by the struggle against the sea. The great flood disaster of 1953 in Zeeland was a rude shock to the Netherlands, demonstrating yet again that the country was not safe. It was shortly after this flood disaster that the Delta Plan was drafted, with measures to prevent such calamities from occurring in the future. This Delta Plan was a defense plan which involved the building of a network of dams in Zeeland and upgrading the existing dikes to a failure rate of 10^{-4}, i.e., one flooding every 10,000 years.

The realization of the Delta Plan started soon after 1953 and in 1986 the impressive dam network in Zeeland was finished. The weak point in the defence was now the Nieuwe Waterweg. The Nieuwe Waterweg connects the main port of Rotterdam with the North Sea, hence it is an important shipping route. Because the Nieuwe Waterweg is completely open and large parts of Rotterdam are situated below sea level, it forms a major risk for flooding of Rotterdam. Moreover, the Nieuwe Waterweg is a major outlet for water coming from the Rhine.

To protect Rotterdam from flooding, a storm surge barrier, called the *Maeslant Kering*, was constructed in the Nieuwe Waterweg. An impression of the barrier is given in Figure 1.

Fig. 1. Top view of the Maeslant Kering near Hoek van Holland. At the top of the figure the Nieuwe Waterweg flows to the North Sea; in the bottom direction is Rotterdam. The Nieuwe Waterweg is about 300m wide.

The requirements that Rotterdam should be protected from flooding, that its port should be reachable at all times (except at unacceptable weather conditions), and that the water coming from the Rhine should not cause Rotterdam to be flooded from the inside, has led to a design of a movable barrier. The barrier consists of two hollow floating walls, called sector doors, connected with steel arms to pivot points on both banks. Each sector door, which should resist the huge forces of the incoming water, is as large as the Eiffel Tower. During normal weather conditions the two sector doors rest in their docks. Only when storms are expected with danger of flooding the two sector doors are closed. The closing procedure consists of several steps. First the docks are filled with water, so the doors start to float, then the doors are moved to the centre of the Nieuwe Waterweg and then they are filled with water until they touch the bottom. A big advantage of the design of the movable barrier is that the construction and maintenance can be done without interfering with the ship traffic. For animation and more information, see the internet-site of the Dutch Ministry of Transport, Public Works and Water Management [RWS].

The main requirement on the barrier is that it is as reliable as a dike. Careful failure analysis showed that a manual control of this barrier would undermine the reliability. For complex tasks – like deciding when to close the barrier and then closing it – normal human beings have a failure probability of one in thousand. Therefore it was considered to be safer to let a computer control the barrier.

2.2 The BOS System

The BOS system decides autonomously about opening or closing the barrier. BOS has the responsibility for closing the barrier when predictions indicate that the expected water level in Rotterdam will be too high. But since Rotterdam is a major port with a lot of ship traffic, the barrier should be closed only when really necessary and as for as short a period as possible. An unnecessarily closed barrier will cost millions of guilders because of restricted ship traffic, while there is also the danger of flooding from the landside through the Rhine if its water cannot flow freely to the sea.

The design of the BOS system is an effort in linking several distinct disciplines. These include the organizational and global overview of the system functionality and requirements by *Rijkswaterstaat* (the Dutch Ministry of Transport, Public Works and Water Management), the hydrological knowledge and model-based water level predictions by the *Waterloopkundig Laboratorium* (independent research institute for water management and control), and the controlling and automation discipline and systems' integration knowledge by CMG.

2.3 Building a Safety Critical System

Because of the dangers and costs involved, very strict safety and reliability requirements are imposed on the BOS software. The failure probability for not closing the barrier when this is deemed necessary should be less than 10^{-4}, and the failure probability for not opening the barrier when requested should be less than 10^{-5}. The latter is seen as more critical because of the danger of destruction of the whole barrier if, due to water flowing from the Rhine, the pressure at the inside, i.e., landside, of the barrier is higher than the pressure from the seaside.

The high safety and reliability requirements make BOS a *mission critical system* (or safety critical system) for which special care, effort and precautions should be taken in order to guarantee its safe, reliable and correct operation. To this extent, the design and development of the BOS software was guided by the standard IEC1508 [IEC1508]. This standard is aimed at software development for safety critical systems. It is a best practices standard that categorizes systems according to their safety and reliability requirements into different *Safety Integrity Levels* (SIL). According to this categorization BOS belongs to the highest SIL level (SIL 4). IEC1508 denotes methodologies, techniques and activities as “not recommended”, “recommended”, “highly recommended”, etc. depending on SIL level. For SIL 4 inspection and reviewing, use of an independent test team and the use of formal methods are “highly recommended”.

None of the “highly recommended” techniques can completely assure the required safety, reliability and correctness [Bro95]. Only a carefully chosen combination of appropriate techniques can help to increase the confidence that the system has the required quality. This has led to the formulation of a dedicated system engineering process depicted in Figure **2**. This Figure indicates which techniques have been used in the different phases of development.

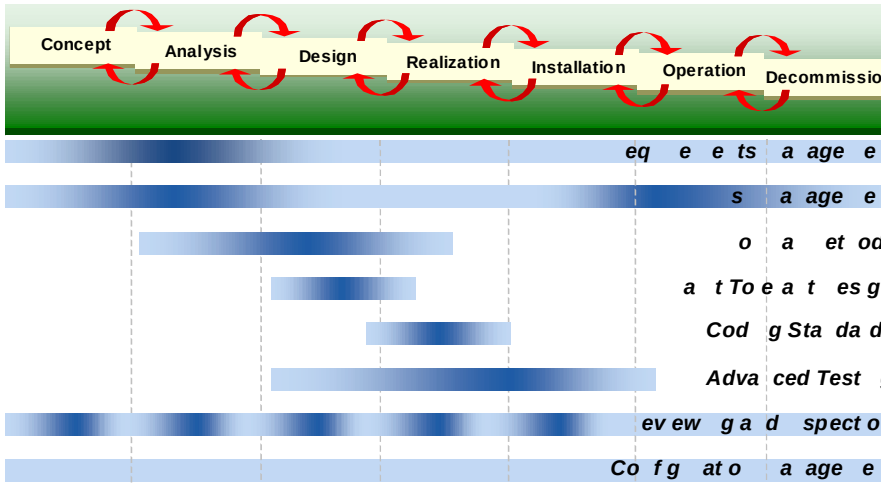


Fig. 2. The dedicated system engineering process combines different techniques.

An integral, risk oriented approach in the system development path identifies at an early stage the aspects of the system that are critical for carrying out the mission of the system. These risks are managed by carrying out both process and product measures. In the development path, a number of methods and techniques are integrated. These are mutually supporting and bring about an effectiveness of the whole that is greater than the sum of the parts. One of the techniques that is proposed in the system development process and that was used in the development of the BOS system is formal methods. The remainder of this paper concentrates on issues concerning the practical use of formal methods in the BOS project.

The BOS system was delivered on time and within budget and is fully operational since October 1998. Its development took three years and about 25 man-years of effort. It resulted in 20,000 lines of formal specification and in 450,000 lines of (a safe subset of) C++ code.

3. Putting Formal Methods into Practice

In this section we describe how formal methods were used in the engineering of the BOS system. It should be noted that a number of observations made here hold more generally for the use of new techniques, not only for formal methods. Moreover, not all benefits achieved within the project can be solely attributed to the use of formal methods because there appeared to be a synergetic effect between the different, quality improving techniques that were applied.

3.1 The Degree of Formality

On the grounds of the promises of formal methods and their recommendation by the IEC 1508 standard, it was decided to investigate their use for the engineering of the BOS system. It was thought that most benefits of the use of formal methods were to be gained if they would be used as an integral part of the system engineering process. This is a clear difference with many other projects where formal methods are used in a parallel or in a "shadow" project. Important consequences of integrating formal methods in the critical path of the engineering process are:

- All project members had to acquire working knowledge of formal methods. Hence a phase of learning and acceptance becomes an integral part of the engineering process.
- The degree of formality that is achieved is probably somewhat lower than is typically achieved in a parallel development process because such processes usually employ highly specialized team-members.
- At completion of the project, it is not possible to compare between processes with and without formal methods. This complicates the evaluation of the use of formal methods.

Central to the engineering process of the complete system was the reduction of risks. For the software system, the following risks were identified:

- The BOS system consists of multiple concurrent processes. This incurs the risks of deadlock or the use of 'bad data' due to synchronization issues.
- The incomplete specification of the behaviour of the system could lead to unexpected behaviour in situations that had not been foreseen. Experience had shown that situations that are typically overlooked are error handling and error recovery.
- Interface faults may occur if interfaces are not specified completely or when they lack robustness.

The use of formal methods was aimed at reducing the probability of the manifestation of these risks. To this end, the degree of formality used was adjusted to the degree of criticality of the different parts of the system. The following levels of formality were identified:

1. Formal annotation of "informal" specifications in order to increase precision and reduce ambiguity.
2. Formal definition of specifications: the purpose is to come to a complete, precise and unambiguous specification of a system.
3. Formal specifications as basis for informal reasoning about the system.
4. Formal specification and reasoning about system properties: mathematical reasoning, possibly supported by tools, is employed to deduce properties of the specification.

Most modules were specified at level 2, sometimes with informal reasoning (level 3). This level was deemed useful for prevention and early detection of faults introduced in the development process through miscommunication and misinterpretation between designers, implementers and testers. In particular, it was hoped that this would aid in reducing the faults that would occur in the integration test (when modules that have been coded and tested individually are composed). Less critical parts of the system were dealt with at level 1, for example, the graphical user interface. Some of the modules that could potentially contribute to the manifestation of one of the aforementioned risks were dealt with at a high level of formality (some at level 3 and some (almost) at level 4). This concerned the internal process scheduling and the communication protocols with the "outside world".

It is important to note that only very small parts of the BOS system were dealt with at level 4, and that for these parts only the design was considered at this level of formality. Not a single line of program code was completely proved correct. Hence, in the BOS approach formal methods do not guarantee complete correctness of code. Different levels of formality and, consequently, different levels of (expected) correctness, were considered. For some parts "a little bit of formal methods" was applied while for other parts "a bit more formality" was used.

3.2 The Selection of Formal Methods

Once it was decided that formal methods were to be used, a selection had to be made in favour of a (combination of) particular technique(s). The techniques had to be suitable for modelling the aspects of the system that were considered critical.

The aspects that are addressed by a formal technique are related to the view a technique takes on a software system. Formal techniques can be classified according to these different views of software systems:

- the *data view*: which data plays a role in a system
- the *functional* (or *input-output*) *view*: which functions play a role in the system and in what way do these transform the data
- the *dynamic* or *behaviour* view: in what order are functions executed

The BOS project focussed on the formalization of the behavioural and functional views. The main candidates considered for the behavioural aspects were CSP [Hoa85], Promela [Hol91] and LOTOS [ISO8807]. The main candidates for the functional view were Z [Spi92] and VDM [Jon90]. The choice in favour of these techniques was made on pragmatic grounds. The arguments that played a role in the selection procedure were: expected learning time, familiarity of the team with a technique and the availability (and price) of tools.

Because of its resemblance to C, it was expected that Promela was easy to learn. Furthermore, a free validation tool called Spin, is available for Promela and can be easily obtained [Spin]. (Free tools were important, in the first place, because there was only a restricted budget for formal methods tools. Moreover, free tools allow

experimenting, playing and learning without bureaucracy and without having to convince managers of their necessity. Especially, in the more or less experimental starting phase, it helps if tools can be obtained easily. These advantages were considered to be more important than consistent and guaranteed level of support provided by commercial tools.)

For the functional view, Z seemed easier to learn than VDM. Furthermore there was some familiarity within the project team and some free tools for Z are available. Hence, the behavioural view was modelled using Promela and the functional view using Z.

Promela was used for modelling the interaction between processes and the interaction between the BOS system and the "outside" world. Verification using Promela was limited to the verification of standard properties such as the absence of deadlock and live-lock. Furthermore, the Promela specifications were simulated. This increased insight into system properties. The use of Promela and Spin has led to the identification of significant errors and omissions in early designs. The use of Promela and Spin is considered successful because (1) it helped in reducing defects and (2) it helped in detecting defects early in the development process which reduces the effort and cost required in later stages of development.

Z was used for specifying the functions performed by processes. A common critique on Z is the great diversity of mathematical symbols used. In practice, this was not considered to be a problem for learning Z. A more significant issue was the great degree of expressive freedom of Z. The bases for Z are set theory and predicate logic. These make Z a very powerful formalism with a great expressiveness. However, as a result, Z allows a great deal of freedom and offers little structure for the style in which it is to be used. In the initial phases of using Z, different people used different styles for writing schemas and schemas were not very comprehensible. A need arose for a common 'style' for using Z, which would be acceptable to all project members. The most important issues this style had to provide were guidelines and conventions for writing specifications and guidelines for choosing suitable (levels of) abstractions. But the style should also take into account the needs of implementers, testers and reviewers. Implementers prefer a style that is concrete and can be easily mapped onto programming language constructs. Testers need clearly distinguishable, testable constraints, and easy controllability (bringing the system in a desired state) and observability (observing that the system is in a required state). Reviews are most easily performed if there is a close relationship, preferably one-to-one, between the concepts of the document to be checked and the document with respect to which it is checked.

For the BOS project, a specification-standard – comparable to a coding-standard – was developed. This standard constrained the use of Z and contained heuristic and pragmatic rules for its use. Also very practical issues like layout of schemas and naming conventions were fixed by this standard. Examples of style are a clear separation between pre- and post-conditions for all operation schemas and a constructive style of writing Z constraints where the new value of a variable (primed variable) always appears at the left-hand side of an equation. The recognition of this style-problem and the development of the standards in such a way that they satisfied

designers, reviewers, implementers and testers, have taken much time. We found that the literature on learning and using Z did not provide sufficient support for these issues. Moreover, the nature of these problems is such that learning by own experience is a necessity anyhow. After the introduction of the standards, the situation improved rapidly: specifications were written according to a similar structure and were more easily comprehensible by implementers, reviewers and testers. From this stage onward, it was found an important benefit of Z that programmers and testers could use the formal specifications as a clear, precise and indisputable basis for their work.

The tool used for Z was ZTC (Z Type Checker) [ZTC]. (The associated animator ZANS was not used.) ZTC can only verify static properties such as syntax, variable declarations and typing. Such a tool is essential for obtaining a reasonable level of completeness and consistency, in particular, since large portions of errors were simple type-errors. Other errors that were encountered were incomplete cases, i.e., not all combinations of predicates of an operation were covered, however, these errors could only be discovered by manual, laborious checking and not through the use of ZTC. It was felt that tools were lacking for rewriting specifications (for instance, for rewriting preconditions into a standard format such as disjunctive normal form) and for (simple) proofs such as checking pre- and post-conditions and invariant properties – which could have been used to find the incomplete cases of operations.

An important problem in the use of formal methods is the making of models, i.e., abstractions of reality. Choosing the level of abstraction seems to be inherently difficult. Although this problem occurs with all modelling methods, it is more manifest with Z than with Promela. This is probably because Promela is, as a language, less abstract than Z: the concepts of Promela (processes, messages, channels, etc.) are more concrete and closer to the concepts which software engineers usually use for thinking and reasoning. The best way of learning abstraction and modelling seems to be through practice.

3.3 Combining Promela and Z

In the BOS project, the behavioural view was modelled using Promela and the functional view using Z, hence there was a need for combining the specifications of the different views of the system. This was done in a fairly informal manner by using naming conventions in Promela and Z.

The use of multiple formalisms brings along advantages as well as disadvantages. An advantage is that the system is considered from different points of view and that special attention is paid to connecting these views. The confrontation of the different views increases the likelihood of finding problems or omissions in an early stage. On the other hand, disadvantages are that the different specifications may overlap, thus introducing possible inconsistency, or that the different specifications may leave certain systems parts unspecified, thus introducing incompleteness. Another disadvantage is that there is no tool support for the integrated use of Promela and Z. Although the use of one, integrated language would have been beneficial for the BOS project, the use of different formalisms was not considered to be a big hindrance.

3.4 Formal Methods in the Development Process

In the BOS project, formal methods were used in the technical design phase for the writing of formal specifications. The resulting formal specifications were used as basis for coding and testing. In this section we describe issues related to formal methods that arose in the different phases of the software development process.

Functional specification & technical design
A functional specification in natural language, combined with Hatley & Pirbhai kind of diagrams [HP87], was input to the project. On the basis of this specification a formal technical design was to be written. The formalization of the functional specification led to the detection and resolution of many ambiguities, omissions and errors in the functional specification. In hindsight, more errors were found through the process of formalization (making the formal description) than in a later stage through the validation of the formal specification. We conclude that the use of formal methods requires precision, structure and consistency, which help in the prevention and early detection of errors.

It takes more time to write a formal technical design than an "informal" design. This is because more thought has to be put into details of the design. Also it is more difficult to leave open (or hide) design decisions. The extra investment in the formalization of a technical design is easily compensated during the implementation, testing and maintenance phases.

Validation
Promela was used for a number of validations and simulations, in particular, of protocols for communication between BOS and its environment. Also, a formal model has been made of the interaction of the modules of the BOS system. These analyses have shown the absence of deadlock and live-lock. No formal validations were performed using Z. Only static checks and informal, manual reasoning about Z schemas was used.

The possibilities for validation were limited by the functionality (in particular for Z) and performance (mainly for Promela) of the available tools. For the Promela tool Spin the state-space explosion problem was the main bottleneck. For Z the possibility of rewriting a specification was missed; see also the discussion in section 3.2 about ZTC.

Design reviews
Reviews were performed based on formal specifications annotated with natural language descriptions. These reviews were found to be much more effective than reviews based solely on specifications in natural language. The increase in effectiveness was attributed to the fact that concepts, attributes and properties could be addressed, discussed and pointed to with more preciseness and less ambiguity. There were less disputes of the form "what do you exactly mean with that?"

Implementation
The implementations of the system modules in (a safe subset of) C++ were developed on the basis of the formal specifications of the technical design. However, no formal derivation of programming code was used because the benefits were estimated to be marginal in relation to the large efforts which would be needed for this. In hindsight, this estimation turned out to be valid. The number of defects introduced by the manual implementation process was relatively small.

An important lesson from the implementation phase is that programmers have to learn to be very precise in reading the formal specifications. They have to convert the specifications into program code without making their own interpretations and design decisions. This is different from what most programmers use to do and it implies the need for a change of mentality.

Testing
We previously reported on the benefits of formal methods for testing in [GWT98]. Although no formal derivation of tests was applied, the use of formal methods facilitated the systematic identification of test cases. The precise and formally specified requirements led to a clear and structured set of tests with a high degree of code coverage.

Testers appeared to be more rigorous than they would have been without formal specifications, in the sense that more detail-errors were found (but less major design errors were found, as expected). Furthermore, the formal specifications settled easily interpretation differences between testers and implementers. Future improvements of the testing phase are possible through increased automated support including automatic derivation of tests from formal specifications [Tre99].

General
Current formal methods focus on one of the views of software systems. It would be desirable to have a formal method which deals with the different views of software systems in an integrated fashion – preferably in combination with existing software development methods and techniques, such as data-flow diagrams, Ward & Mellor [WM85], Hatley & Pirbhai [HP87] or UML [BRJ98].

4. Lessons Learned

In this section we describe some of the important lessons we learned from our experience with formal methods.

4.1 Quality

The general conclusion of most project members is that the quality of the system is higher than could have been achieved without the use of formal methods. Once a working knowledge of formal methods had been acquired, the system modules produced were close to “first time right.” Modules that were specified formally

required less maintenance and rework. Most problems were encountered in modules where formal methods were not used or where the quality of the formal specification was low due to time-pressure.

4.2 Costs

Costs of formal development were estimated to be comparable with costs without the use of formal methods. It should be noted, however, that a large amount of these costs were related to learning and obtaining experience, hence it could be expected that a next formal project will save money.

4.3 The Learning Phase

On the first use of formal methods a training phase is unavoidable. Besides the learning of syntax and semantics of the formalisms, people had to learn how to use the formal methods effectively. In particular, people had to learn that a formal specification had to be read in a much more precise manner than a specification in natural language. Also, in the learning phase it was found that a specification "style" was needed to constrain the degrees of freedom of the specification methods.

In the process of getting acquainted with a formal method, it was found to be important to have an experimentation phase. In this phase people should be allowed to get a feeling for the possibilities, structure, constructions and the like of a formal method by making specifications and programs that are not part of the final product. This leaves opportunity for exploring and investigation and learning from making mistakes without the pressure of having to produce fault-free products. This learning phase is also a good time to explore the possible ways in which one formal method can be combined with other (formal) methods. The availability of tools helps in this learning phase. In particular, a formal-methods simulation tool is a good means for providing feedback to the student of a method. The use of the Promela simulator in Spin was profitable in this respect.

4.4 Support from Academia

During the development of BOS CMG was supported by the Formal Methods & Tools group of the University of Twente. It turned out that the expertise of the University of Twente was not completely sufficient for supporting the application of, mainly, Z in large projects such as BOS. The expertise was mainly oriented towards formal methods as formal (mathematical) languages and towards formal syntax, semantics and proof techniques. The problems encountered in BOS, however, were not formal (mathematical) problems, but mainly problems related to the use of formal methods in a large project: how to use formal methods effectively in a software development trajectory; how to combine formal methods with other software engineering techniques; and the identification and definition of the "specification standards" for Z. With these aspects there was little experience at the University of Twente. Also in the literature there is little known about these practical aspects of the

use of formal methods. The combination of knowledge and experience of both practical software engineering and formal methods is still rare.

4.5 Tools

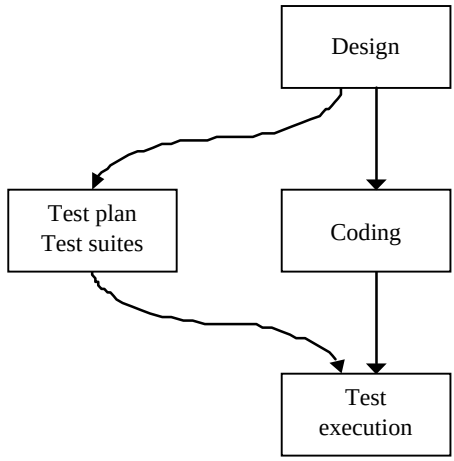
Although it is known for some time that the availability of tools is essential to the industrial acceptance of formal methods, the tools we used provide too little support. In particular, attention should be paid to scaling up tools to large applications and to the integration or compatibility of formal methods tools with existing software development methods and tools.

4.6 Planning and Monitoring

The use of formal methods in the development process had several consequences for the planning of the project. Firstly, formally specifying the functional design takes up more time than specifying in natural language. However, using a formal method for the design leads to the identification of omissions, ambiguities and inconsistencies that can be removed at a relatively early stage. We are confident that the effort invested in the design phase has been (more than) compensated by reductions of effort in subsequent phases.

A second consequence is that large parts of the testing phase can be performed concurrently with coding of the implementation. Whereas in many system development projects test suites are only developed and implemented after the system has been implemented, BOS shows that the level of detail provided by a formal design makes is possible to perform these phases concurrently. Starting from the formal design, implementers start coding, while at the same time testers start writing test plans, generating test suites and developing a test environment. Usually, testers are faster so that when the implementation is ready, test execution can start immediately. Clearly, this reduces the total project time by reducing the critical path for testing.

In the beginning, metrics from the design phase were not used for planning the coding and testing phase. After some time, it turned out that the number of lines of Z could be used as a rudimentary metrics for planning. Analysis showed a correlation between the number of lines of Z and the number of lines of C++ code or the testing effort, respectively. In particular, after some experience had been obtained and some data had been collected, module test execution could be planned relatively precisely based on this metrics. This effect was strengthened by the fact that test plans and test suites were developed concurrently with coding, see above. Further analysis is needed to explore the precise nature of the correlation between the size of the specification and the implementation and testing effort. Other metrics that could be explored are the number of Z schemas or the number of data items in a description.



4.7 People Management

People are stimulated by the possibility of learning new skills and experiencing that they improve the quality of their work. This can be illustrated by the fact that at later stages of the project, some project members decided to write and analyse a module (GUI) at a higher level of formality than followed from its degree of criticality.

The different levels of formality require different skills in using formal techniques. Hence, the training of different project members may need to be aimed at different skill levels. Learning to read and review formal specifications requires less training than learning to write specifications, which, in turn, requires less training than necessary for doing model-checking. Of course, each person's tasks should be adapted to his or her capabilities and education, e.g., while almost all software engineers, after some training, can read formal specifications, model-checking is better performed by people having some mathematical background. Some team members did not have any affinity with formal methods. These persons left the project, to mutual benefit of both these persons and the project.

4.8 Communication with the Client

A common critique of formal methods is that they are not suitable for communication with clients. In the BOS project we have dealt with this issue in the following way. Firstly, all specifications consisted of a combination of formal text and accompanying text in natural language. In practice, the clients focussed mainly on the description in natural language. Secondly, it was found that the animations of Promela specifications using the Spin simulator were very helpful in communicating with the client, in order to have the client understand the design and to make the client aware of potential problems.

5. Conclusions and Recommendations

We have described some of our experiences with the use of formal methods in the commercial engineering of an industrial, safety-critical system. The system was delivered on time and within budget. It was found that formal methods had a positive contribution to the quality of the system. However, it should be noted that formal methods were used as one of a number of integrated techniques and that there is a synergetic effect between these techniques. Formal methods have the greatest added value when they are applied in combination with other quality improvement techniques. Hence, methods to improve the quality of the software product should be used in combination with methods for the improvement of the quality of the engineering process (such as are suggested by CMM). More about the other quality improving techniques used in BOS can be found in [WBG98].

On pragmatic grounds we used Promela for modelling the behavioural view of the system and Z for the functional view. The introduction of Promela went relatively easy because of its ease in use, among others, through its visualization using message sequence charts. A few early successes in finding major design errors stimulated its further use. Overall, it was found useful in exposing errors in interface design.

The use of Z in a professional software-engineering project requires more than a few courses in first-order predicate logic and Z-notation. A useful idiom and conventions have to be developed ranging from high-level structuring conventions to naming conventions. The developed set of conventions for Z together with extensive coding standards (implementation guidelines) proved to enhance efficiency in the coding phase. In [GWT98] we have reported the additional benefits of this approach to the independent test phase.

More errors were found through the process of formalization (making a formal description) than in later stages through the validation of the formal specifications. Formal derivation or proof of code was not used at all. Yet, we conclude that the use of formal methods was profitable. Their use provides precision, structure and consistency that help in the prevention and early detection of errors.

Acknowledgements

The authors would like to thank Eric Burgers, Wouter Geurts, Franc Buve, Rijn Buve, Sjaak de Graaf, Hedde van de Lugt, Peter Bosman, Peter van de Heuvel and Robin Rijkers, all of CMG Public Sector B.V. in The Hague, for their active participation during the interviews that form the basis of this paper. Annemieke van Wijk, CMG, for is thanked for her secretarial support. Ed Brinksma, Pim Kars, Wil Janssen, Job Zwiers and Theo Ruys from the University of Twente gave support and feedback during different phases of the BOS development. The second author acknowledges the financial support of CMG The Netherlands while performing part of the work underlying this paper. The anonymous referees are thanked for their constructive criticism, which helped in improving this paper.

References

[BRJ98] G. Booch, J. Rumbaugh and I. Jacobsen. The Unified Modeling Language – User Guide. The Addison-Wesley Object Technology Series, Addison Wesley, 1998.

[Bro95] F. P. Brookes. *The Mythical Man-Month: Essays on Software Engineering.* Anniversary edition. Addison Wesley, 1995.

[GWT98] W. Geurts, K. Wijbrans and J. Tretmans. Testing and Formal Methods – BOS Project Case Study. In: *EuroSTAR'98*: 6th European Intl. Conference on Software Testing, Analysis & Review, pages 215 – 229, Munich, Germany, November 30 – December 1, 1998.

[Hoa85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[HP87] D. J. Hatley and I.A. Pirbhai. *Strategies for Real Time System Specification*. Dorset House, 1987.

[Hol91] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[IEC1508] IEC (International Electrotechnical Commission), Functional Safety of Electrical/Electronic/Programmable Systems: Generic Aspects, IEC 1508, 1995. (Now: IEC 61508).

[ISO8807] ISO, Information Processing Systems Open Systems Interconnection, LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, International Standard IS8807, ISO, Geneva, 1989.

[Jon90] C. B. Jones, *Systematic Software Development using VDM* (2nd edition), Prentice Hall, 1990.

[Kar97] P. Kars, The Application of Promela and SPIN in the BOS Project, in J.-C. Grégoire, G. J. Holzmann and D. Peled (eds), *The Second Workshop on the SPIN Verification System; Proceedings of a DIMACS workshop*, August 5, 1996, volume 32 of DIMACS series in Discrete Mathematics and Theoretical Computer Science, pages 51-63. American Mathematical Society, 1997.

[Kar98] P. Kars, Formal Methods in the Design of a Storm Surge Barrier Control System. In: G. Rozenberg and F. W. Vaandrager (eds.) *Lectures on Embedded Systems*, pages 353 – 367, Lecture Notes in Computer Science 1494, Springer-Verlag, 1998.

[Pau94] M. C. Paulk et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1994.

[RWS] Dutch Ministry of Transport, Public Works and Water Management. URL: http://www.minvenw.nl/rws/dzh/svk/engels/index.html

[Spin] Spin. On-the-fly, LTL Model Checking with Spin. URL: http://netlib.bell-labs.com/netlib/spin/whatispin.html

[Spi92] J. M. Spivey. *The Z notation*: *a Reference Manual* (2nd edition). Prentice-Hall, 1992.

[Tre99] J. Tretmans. Testing Concurrent Systems: A Formal Approach. In: J. Baeten and S. Mauw, *Concur'99*. Lecture Notes in Computer Science, Springer-Verlag, 1999.

[WBG98] K. C. J. Wijbrans, F. Buve and W. Geurts. Practical Experiences in the BOS Project. In: *Proceedings of the Embedded Systems Symposium*, May 19, 1998, Eindhoven University of Technology, Eindhoven, The Netherlands.

[WM85] P.T. Ward and S.J. Mellor. *Structured Development for Real Time Systems*. Volume 1: Introduction & Tools. Yourdon Press Computing Series. Prentice Hall, 1985.

[ZTC] ZTC. Z Type Checker. URL: http://saturn.cs.depaul.edu/~fm/ztc.html