INFSOF4159

Information and Software Technology 00 (2000) 000–000

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

Practitioners' views on the use of formal methods: an industrial survey by structured interview

C. Snook[a], R. Harrison[b,*]

[a]Department of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, UK
[b]School of Computer Science, Cybernetics & Electronic Engineering, University of Reading, Whiteknights, P.O. Box 225, Reading, Berkshire RG6 6AY, UK

Received 16 June 2000; revised 2 November 2000; accepted 3 November 2000

Abstract

The recognised deficiency in the level of empirical investigation of software engineering methods is particularly acute in the area of formal methods, where reports about their usefulness vary widely. We interviewed several formal methods users about the use of formal methods and their impact on various aspects of software engineering including the effects on the company, its products and its development processes as well as pragmatic issues such as scalability, understandability and tool support. The interviews are a first stage of empirical assessment. Future work will investigate some of the issues raised using formal experimentation and case studies. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Formal methods; Empirical assessment; Survey; Structured interview

1. Introduction

This document reports on a series of structured interviews which have been conducted with formal methods practitioners. In Section 1, the need for empirical assessment, especially in formal methods, is introduced. The types of empirical assessment are described and the contribution each makes to the establishment and investigation of a hypothesis is discussed. This provides a context for the report on the conduct and findings of the series of structured interviews, which is described in Section 2. In Section 3, our plans for future work arising from the results of the interviews are presented including the statement of two hypotheses and an outline of how they could be investigated further.

1.1. Empirical assessment

Glass [9] comments on the way software engineering research has become insular and 'academic', losing touch with the practitioners and not validating theory with real world evaluation. In response, practitioners have lost faith in research results. In fact, research and development should go hand in hand so that research ideas are transferred into practice via an established process and bad ideas which cannot be put into practice are not kept alive mainly by research advocates. Formal methods have been cited as an example of an idea being kept alive purely by research [7,9]. Similarly, Fenton [7] warns the research community that it should not be exasperated by the poor industrial acceptance of new methods when they lack empirical validation.

1.2. Types of empirical assessment

Empirical assessment can be classified as surveys (systematic post hoc data collection from a known population), formal experiments (controlled and replicated treatments on a number of subjects) and case studies (intensive interpretation of a small sample). Kitchenham [11,12] identifies a method for selecting evaluation techniques for software engineering methods and tools that is based on surveys, formal experiments and case studies. As part of the same project (DESMET), Kitchenham et al. [13] provide a very good critical review of past quantitative assessments and again base this around a classification into surveys, formal experiments and case studies.

Daly [5] points out the value of using all the three forms of empirical assessment to support each other in establishing a hypothesis. The survey contributes to the formulation of the hypothesis and increases the likelihood that it is relevant, formal experiments establish that a relationship exists

* Corresponding author. Tel.: +44-118-931-8615; fax: +44-118-975-1994.

E-mail addresses: cfs98r@ecs.soton.ac.uk (C. Snook), rachel.harrison@reading.ac.uk (R. Harrison).

and case studies demonstrate that the results can be generalised to real life situations.

1.3. Surveys

Surveys rely on the individual's memories of their experiences. Because of this, they can be limited in accuracy. Pannell and Pannell [14] give an informative discussion on the problems of extracting the truth via surveys and on how to maximise the chances of getting valid answers. Some of the problems include incorrect answers (an estimated 5–17% of answers are incorrect), misinformation, changing opinions, wording of questions, misinterpretation and ordering of questions. Nevertheless, surveys provide a powerful method to get an initial indication of the properties of a topic from a wide subject base. Survey data can lead to the formulation of relevant, and widely held, hypotheses.

A survey based on a distributed questionnaire relies on the questions asked and the way they are phrased. This implies a prior knowledge of the interesting issues and a possible outcome. A structured interview [5] consists of an interview based around a predefined set of questions. The questions provide a consistent structure for the interviews but the interviewer can discover knowledge by seeking confirmatory evidence as necessary. The interviewer can also explore the experience and language of the interviewee to put the answers in context. Thus many of the shortcomings of an independent survey are overcome. Structured interviews are limited to a small selected set of experienced subjects, but enable a wider exploration of the subject to be performed and a higher level of confidence in the answers. However, the results will be a reflection of the opinions and prejudices of a small set of subjects. The selection of these subjects may ensure that they are the best placed individuals to give an accurate opinion. On the other hand, other empirical assessment techniques should be used to test the results of the structured interviews. Our structured interview is reported in Section 2 of this report.

1.4. Formal experiments

The purpose of a formal experiment is to test a relationship in a particular system. We must minimise the effect of confounding factors so that we can attribute changes in the dependent variable to changes in the independent variable. We should not expect formal experiments to be set in realistic scenarios. This would be counter to the design aims of eliminating confounding factors. Instead we should concentrate on isolating and demonstrating the relationship under test. Once the relationship has been established as likely to exist we may then consider to what degree it is relevant to real life scenarios.

1.5. Case studies

Case studies lack the level of control that formal experiments have. The behaviour of interest is observed in a real life example. The many other environmental parameters are uncontrolled and may influence the dependent variable being observed. To alleviate this to some extent a typical baseline is used for comparison. However, a case study cannot be considered as rigorous an empirical investigation as a formal experiment. Nevertheless, case studies have an important role because they test whether a relationship is observed in real situations. This can support formal experiment results, either as an investigatory stage (establishing a hypothesis to test) or as a follow up stage (establishing the generality of experimental results).

2. Industrial survey

We performed a survey of the opinions of industrial users of formal methods. This covered a broad range of topics associated with the effects that using formal methods might have on a company and its products. The survey was conducted by structured interviews based on a questionnaire (see Appendix A).

2.1. Purpose

The aim of the survey was to explore the experiences of the practitioners directly. There are many popular theories about formal methods that have questionable validity and it is often unclear whether they are based on actual experience. Hall discusses some of these myths [10] as do Bowen and Hinchy [2]. Therefore, it was seen as important to investigate the effects of using formal methods directly with individuals who had first-hand experience. Of course, the results are still dependent on the subjective opinions of these individuals and the environments in which their experiences were obtained. This must be borne in mind when the results are interpreted and the results should be viewed as provisional until further empirical assessment has been carried out to corroborate them.

We wanted to discover the main issues involved in the use of formal methods, in particular, the issues surrounding understandability and the difficulty of creating and using formal specifications. It was hoped that significant points would be raised that would warrant further empirical assessment. In this way the survey was seen as the first stage of a 'multi-method' programme of research as described by Daly [5]. The purpose of this first stage was to raise interesting and relevant provisional findings for further research rather than firm conclusions which would be suspect based on such a small survey.

2.2. Conduct of survey

The companies were initially contacted by email with a brief outline of our aims and the questions that would be asked. Meetings were set up at the company's premises where the representatives were interviewed. The interviews were structured around a questionnaire but the interviewees

Table 1
Participating companies

Company	Identification in this report
(Wishes to remain anonymous)	Interviewee A
IBM United Kingdom Laboratories, Hursley Park, Winchester, Hants	IBM
Marconi Electronic Systems; Avionics Systems, Airport Works, Rochester, Kent	Marconi
Philips Research Laboratories, Crossoak Lane, Redhill, Surrey	Philips
Praxis Critical Systems, Manvers Street, Bath	Praxis

were encouraged to digress and elaborate on topics as much as they felt necessary. The questions were used to trigger discussion and as a checklist but in an effort to explore the subject widely, the discussions were conducted in an open, free form without constraining the topic to the initial question. The interviewees related answers to their experiences to provide justification, and in the process the context of the interviewees answers and their understanding of key phrases was discovered. This happened mostly as a natural part of the discussions without conscious effort. The final question asked the interviewee if there were any important issues that had not been covered. In most cases, the interviewee recapped some of the more important issues at this point but did not raise any new issues. This indicates that the questionnaire covers the main points of interest with respect to formal methods. Each interview lasted approximately 2 h. The same interviewer conducted all the interviews. The interviews were tape-recorded. It was felt that recording the interviews prevented the interviewer from being distracted by note-taking. It also meant that the interviewees' opinions could be summarised and distilled with greater consideration and care than would have been possible by taking notes 'on the fly'. The tapes were analysed in detail and the comments categorised and matched with like comments from other interviewees. From this process a table of summary notes was built up with rows representing each point made by the interviewees and each column representing the summaries of a particular interviewee's responses. The text of this report was written from the summary table. Despite the careful analysis of the actual conversation on the tapes, it is still possible that the authors might misinterpret responses or inappropriately emphasise a point. To guard against this the report was circulated to the interviewees for review. A few adjustments arose from this review stage but on the whole the interviewees agreed that the report was an accurate representation of their views.

All the interviewees had at least some experience in using formal specifications on full-scale products. Some had also performed refinement, model checking and verification proofs. For various reasons, only one company was using formal methods to the same extent as previously but all retained a capability or interest. The market sector varied greatly, including commercial computing systems, safety critical embedded systems and high street consumer products. Table 1 lists the companies and Table 2 gives an outline of their background and experience. At this stage of investigation the wide spread of market sector backgrounds is an advantage to the broad information gathering process. In subsequent stages, less variability will be needed as we focus more narrowly on selected issues. The companies are, in most cases, the market leaders in their sector and the interviewees are the technical experts within those companies. In several cases, the interviewees have published in the area of formal methods. It is reasonable, therefore, to claim that the interviewees are knowledgeable and experienced in the use of formal methods. It might be argued that the interviewees are all proponents of formal methods and the results might therefore be a biased view. We believe that the commercial pressures upon the interviewees would not allow them to maintain an unrealistic stance. It was apparent however that the market sector has a bearing on the stance

Table 2
Main characteristics of contributors

Company	Market sector	Notations used	Extent of use	Approximate size of systems (Kloc)	Current level of use
Interviewee A	Contractor with personal experience	Z, VDM(some), CSP (some)	Experience with large and small applications	–	Introducing formal methods into a company
IBM	Commercial computer systems	Z, B	Mainly specification	50	At option of project manager
Marconi	Military embedded systems (some safety critical)	B	Full development including refinement proofs, etc.	3	Completed case study — bidding for contracts
Philips	Consumer products	Set theory and first-order logic	Mainly specification	10 +	Isolated usage — investigating applicability
Praxis	Safety critical systems	Z, VDM, CSP (some), CCS (some)	Some full developments, others specification only	10–100 +	Continuing full-scale use

taken, with the safety critical areas having much more compelling reasons for supporting the use of formal methods, and the others having a more guarded response.

Each interviewee was asked to define a formal method. Most answers indicated that a mathematical notation or underlying theory was needed (one interviewee required a precise syntax and semantics). Some required there to be methods for manipulation and refinement, others recognised these as possible extensions but did not require them. It was thought that some companies might have a looser definition of formal methods. To test this the interviewees were asked if they would include modelling languages such as UML. All would not, although several interviewees suggested that some parts of UML (e.g. state charts) are close to being a formal notation. Some added that UML did not contain the facilities to express the semantic details of the behaviour of systems.

The formal methods that had been used by the interviewees are as follows. Z is a specification language developed by the Programming Research Group at Oxford University around 1980. It is based on axiomatic set theory and first-order predicate logic. B is a specification and development method designed by Jean-Raymond Abrial of B Core, UK. It is a system for the formal development of software using the notion of Abstract Machines. Abstract Machines are specified using the Abstract Machine Notation (AMN) which is based on the mathematical theory of Generalised Substitutions. B is related to Z and supports the development of C code from specifications. The Vienna Development Method (VDM) is a notation and set of techniques for modelling computing systems, analysing those models and progressing to detailed design and coding. VDM has its origins in the work of the IBM Vienna Laboratory in the mid-1970s. Communicating Sequential Processes (CSP) is a notation for concurrency, based on synchronous message passing and selective communications, designed by Anthony Hoare in 1978. Calculus of Communicating Systems (CCS) is a mathematical model for describing processes, mostly used in the study of parallelism. A CCS program, written in behaviour expressions syntax, denotes a process behaviour. Programs can be compared using the notion of observational equivalence.

2.3. Results

2.3.1. The customer's viewpoint

The companies interviewed had very different market sectors and this led to large variations in the answers to questions about the customer's views on their use of formal methods.

Marconi, being a UK defence contractor, often bids for contracts with Def-Stan 00-55 as a mandatory standard [1]. Hence Marconi's use of formal methods is imposed by its main customer (or at least by the regulatory authorities that its customer has to satisfy). Marconi also supplies outside of the UK, e.g. USA, and for these customers it is expected that persuasion would be needed to convince them to accept formal proof in place of other verification methods such as testing and reviewing.

Praxis also supply to the UK MoD and to other authorities that are very safety conscious, such as aviation authorities. It also supplies to other markets and finds that some of these customers resist the use of formal methods because of the barrier it creates between supplier and customer. Typically, the customer will need to train some of its employees if it wants to be involved in verification and validation activities during the software development.

The remaining interviewees felt that their customers (which for IBM and Philips were internal) were usually impressed by the use of formal methods and assumed that they would lead to high quality products.

Where the formal specifications were used as interfaces to customers, the customer's technical staff (who sometimes needed special training) usually found formality helpful because they knew the precise behaviour of the specified system. It was recognised that the audience may be restricted by formality but this is the case for any technical specification.

Both IBM and Praxis commented that one of the main barriers to the widespread use of formal methods is the general acceptance that software is error prone. One interviewee said, "If you want highly reliable software then formal methods are the most cost effective way to produce it, but if the customer will accept unreliable software then it is cheaper not to use formal methods." From the suppliers' point of view, any subsequent re-work is either covered in the initial price or is paid for by the customer as a maintenance contract.

IBM went on to say that some customers do not want to be tied down to what they require, but would rather have a vague specification of requirements and hope the supplier produces something over and above it than be forced to address compromises in order to specify precisely their requirements and then take responsibility for the system's validity.

2.3.2. Impact on company

2.3.2.1. Quality assurance. Opinion on how formal methods affect quality assurance issues was uniform. All (except one company that, independent of the method used, had dispensed with its quality assurance function) agreed that the quality assurance function is not changed. The auditors may need to have some appreciation of the records that they will be examining but this is true of any new method. They did not feel that quality assurance personnel would need a full understanding of the formal specification notation. They only need to satisfy themselves that the record has been produced and that the right sort of people have verified and authorised it.

2.3.2.2. Consultancy and skills. Several companies had employed external consultants during the initial projects that introduced formal methods to the company. This was seen as necessary. Training was given to all staff involved in formal methods. Generally a couple of weeks was found sufficient for the staff to assist in formal methods projects. However, it was not thought feasible to train the existing staff to a degree that they could successfully use formal methods without expert guidance on hand until they had built up some experience and practice.

Not many experienced modellers are required as the majority of the project staff need to be able to comprehend specifications and write detailed sections as directed, but do not need to be able to create the overall structure of the specification.

One interviewee felt that external consultants, who are typically extremely intelligent, would make any project successful, no matter what method they used. This could give a biased view in favour of formal methods. Similarly, companies that use formal methods only recruit personnel who demonstrate the ability to use formal methods, thereby increasing the quality of their staff. Evidence of this was provided by another interviewee who reported that his company tended to recruit from research areas to fill vacancies involving formal specification. This filtering effect inherent in the adoption of formal methods could be seen as a beneficial effect on the culture.

However, there can be detrimental effects if, having altered the company's methods and culture, none of the permanent staff are sufficiently skilled to take over when the consultants leave.

2.3.3. Impact on product

2.3.3.1. Reliability. Only IBM and Praxis had any evidence of product improvement. IBM had found (based on informally collected data) a 40% reduction in post-delivery failures compared to their own average product performance. Praxis referred to published data which compares a Praxis software product favourably with the industry average data. As with most case studies, the cause of this improvement cannot be identified with certainty to the use of formal methods, since other factors such as culture may be atypical, but it does provide a positive empirical indication of the possible benefits of formal methods. Of the other interviewees, Marconi's experience was based on a study which did not go into service, and Philips and Interviewee A did not have personal knowledge of the relevant product service histories.

There was, however, an implicit assumption from the interviewees that the product would be more reliable. This was indicated by comments such as, "if you want software that works, then the only cost effective way to do it is with formal methods". This implies that formal methods produce a level of reliability that may only be achieved at significantly greater cost by using conventional methods. This may be a subjective view but it is the view of those who have used both formal and conventional methods in software development.

2.3.3.2. Efficiency. Praxis had noted that the product code was more efficient than the conventionally specified software. The precise and accurate nature of the specification makes the coding task straightforward and the coder is less likely to build in redundant code. Note that this observation is supported in the findings of a comparative study by Brookes et al. [3].

2.3.3.3. Functionality. Praxis also noted that the effort that is needed in formal specification tends to deter the functionality growth that afflicts many software systems.

2.3.3.4. Traceability and maintenance. The interviewees were asked if the structure of the specifications is reflected in the code. Generally, the answer was affirmative and this was thought to be beneficial in aiding traceability between the specification and the code. Some noted that this structuring of the code may not be the most efficient implementation but that the traceability benefit outweighed this. Philips questioned whether the specification should influence the structure of the code or not. One view is that the specification should not if it is at the right level of abstraction to be a requirements document. Another is that it would be beneficial if the specification could impose structuring requirements, for example, to improve re-use.

Two interviewees, Praxis and Philips, felt that the formal specification helped a maintainer to understand what changes were needed and therefore to get them right. Marconi felt that the specifications had little impact on maintenance but that the B toolkit helped a lot in automatically detecting the affected components and re-checking them. IBM said that they do not normally use the documentation for maintenance, although in one case, when they did and it was a formal specification, the project leader estimated a 50% reduction in the cost of maintenance.

The interviewees were asked if they thought that formal specifications help prevent the degradation of code structure through maintenance and also whether the specification itself degrades through maintenance. IBM did not use or maintain the specifications after delivery and therefore could not answer. Interviewee A had not been involved in the product maintenance stages. Marconi felt that the B-tool was largely responsible for preventing code degradation since it maintains the traceability from the specification. Philips thought that the formal specification would help prevent code degradation if traceability could be maintained but that this had been a problem (see comments under Lifecycle). Praxis thought that the formal specification prevented code degradation by supporting good practice (i.e. changing the specification first when implementing changes).

2.3.4. Impact on development

2.3.4.1. Development lifecycle. All agreed that there is no change to the sequence of activities performed during the software development lifecycle but the effort involved in some of the stages is dramatically altered. The specification stages take a lot longer. However, everyone agreed that generally the resolution of specification problems discovered during this stage was well worth the effort because these problems would otherwise have arisen later during the development, with increased re-work consequences. Similarly, Interviewee A believed the primary benefit of formal specifications to be the improved analysis of the problem domain that results from the process of writing them. This leads to a better understanding of the requirements prior to starting a design which may be another reason for the reduction in problems occurring later in the lifecycle.

Verification stages, particularly testing, were much reduced since far fewer errors remained to be discovered. The net effect was that the overall timescales were usually very similar or possibly better for the development that started with a formal specification.

However, Philips found that formal specification did not fit easily with the iterative lifecycle used for some products. Since Philips do not normally have an end-customer performing the requirements specification role, they have to develop the requirements themselves. Also, they typically have very short timescales to develop new products and often refine the requirements as the product is being developed. The time consuming first phase of formally specifying to resolve the requirements issues does not fit into this type of lifecycle easily. In fact, Philips had examples where the product was finished before they could complete the specification. To address this, the company are looking at different levels of specification formality appropriate to different product lifecycles.

Formal specification was also found to aid the verification testing process. Marconi, Philips and Praxis all reported that testing was more efficient and more effective when a formal specification was available. This was the primary driving force for improving specification techniques, as far as Philips were concerned. From the formal specification, it is easy to derive test cases and some companies had gone as far as automating this process. Marconi had used B specifications to generate expected results automatically and Philips had generated test cases from state charts automatically.

2.3.5. Size of system

A guide to the size of the systems developed using formal methods is shown in Table 2. The figures should be taken as a rough guide only due to possible variations in the measurement of a line of code and in the programming languages used. However, they indicate that formal methods were used on systems typically in the region of tens of Kloc. The interviewees were asked if large systems were a problem when using formal methods (compared with any other method). Answers varied somewhat but, generally, the impression was that size is not a major obstacle any more than other methods. Marconi and Praxis indicated that proving becomes problematic with large systems and that the proof checkers and, to a lesser extent, model checkers may not scale up very well. For formal specification, though, IBM said that large systems are dealt with by breaking the system down into 'encapsulated' sub-components that could be dealt with separately. Marconi, using the B toolkit, felt that the system specification was difficult to cope with due to the fact that it could not be subdivided but that, as soon as the design was refined, the system naturally was divided into encapsulated sub-components. It appears that the concept of breaking down the system via encapsulation is crucial in dealing with industrial scale problems.

2.3.6. Understandability

The interviewees did not feel that there were any significant understanding problems with formal notations (although some commented that this may be because they recruit people who will understand them). The notations were not seen as being a problem in this respect. In fact Praxis felt that formal specifications should be easier to understand than code.

Several interviewees said that it is essential to comment Z with English text to explain the structure of the model. This is not so necessary with B as it is more structured. Most companies impose some styling (e.g. lexical) rules on top of the formal notation in order to improve the consistency of style throughout the organisation, although the general impression was that this was not a major factor in understandability. Interviewee A had used a 'friendly' style of Z (a reduced subset avoiding the less intuitive constructs and annotation in a light style to enhance the friendly feel of the document) and felt that it had been beneficial to understanding for unpractised readers.

Only one specific feature that affects understanding was mentioned. Praxis had found that over-reliance on invariants can be confusing. It is sometimes better to explicitly state things that change during an operation rather than rely on implicit changes as a result of satisfying a state invariant, even if this is, strictly speaking, redundant.

The area that the interviewees did think was difficult was in creating the formal specifications. IBM and Praxis had both employed expert consultants to facilitate this stage. Marconi said that the most highly skilled or experienced people were needed to do the initial or higher level structuring, although the others could then cope with adding in the detail. IBM said that the ability to create the right (i.e. useful) model requires the most skill and experience. It is too easy to create a model that might be consistent in itself but which does not contain the abstractions that are useful in describing the problem.

2.3.7. Tools and notations

The interviewees were not questioned specifically about tools but during the course of these discussions, the B toolkit stood out as the only tool that had been used to any extent. IBM had started with Z but switched to B so that the B toolkit could be used. Marconi's entire experience was based around the B toolkit and they were very pleased with it in most respects. They relied on it heavily and found that it helped in tracing, proving and maintenance work. Praxis said that there were few industrial strength tools but agreed that the B tool is an exception. A Praxis interviewee thought that B was not as suitable as Z for the system level specification. However, Marconi have used B for all levels of specification.

Philips thought that tool availability has a big impact on the decision to use certain specification techniques. In particular, tool support to maintain traceability between specifications, implementation and test cases is an area of concern.

Interviewee A was in the process of installing the UML as a company-wide documentation language. They were anticipating using formal specification in conjunction with the UML. Philips was also adopting the UML in some sectors of the company.

3. Conclusions

As this is a first stage, opinion gathering exercise, we are wary of drawing any firm conclusions. The results described above are considered indicators for further investigations. However, we have summarised some of the main opinions recorded. Formal methods are worthwhile in terms of improved quality of software with little or no additional lifecycle costs, but only when compared to a rigorous development lifecycle where the cost of software errors is high. If the market does not demand high quality software, then it is more difficult to justify their use. The introduction of formal methods affects a company's workforce, processes and culture through effects such as skills filtering and consultancy syndrome. It may also impact on the relationships with a customer through kudos, and communication implications. Overall the effects are usually beneficial but there can be some problems to overcome. There is no real problem with understanding the specifications — given suitable training they are no more difficult to understand than programs. The difficult part is creating the specification as appropriate modelling requires practice and skill. Encapsulation is important within the context of large systems. There is a lack of industrial scale tools, the B-toolkit being the only suitable toolkit.

4. Future work

Many interesting points have arisen from the structured interviews. We have selected two hypotheses for further investigation. The first is a comparatively straightforward hypothesis that is suitable for formal experimentation in a laboratory setting. The second is a more complicated issue and will require ingenuity in order to facilitate further empirical investigation.

4.1. Understandability

One of the areas that was expected to be rich with discussion was that of understandability. It is often said that one of the problems with formal notations is that they are difficult to understand and that highly trained mathematicians are needed to read them. However, the interviewees did not support this view. This is significant because it conflicts with popular opinion. All the experienced interviewees agreed that typical software engineers have no real difficulties with understanding formal notations. As one interviewee put it, formal specifications are no more difficult to understand than code. We intend to design and conduct an experiment to test this, by writing a specification using Z and implementing it in a programming language. A close correspondence will be maintained between the specification and the implementation, both in functionality and in structure. Subjects will be given either the formal specification or the code and their understanding will be tested using questionnaires. The effect of bias factors, such as whether the structure is more appropriate for one of the representations, will require consideration when assessing the results.

4.2. Modelling

The interviewees thought that the difficulties with using formal specifications were in finding the useful abstractions from which to create models. This is surprising, because the same engineers are practised at creating models of problems and solutions using less formal notations as a transitory step in programming. The criteria for selecting a model on which to base a formal specification may differ from those of less formal design, nevertheless, one would expect similar skills to be applicable. One is led to suspect that there may be something lacking in the available notations and methods compared to informal program design methods.

Comparing the available formal specification methods with informal program design methods, we find that program design methods concentrate on structure. Their aim is to provide the engineer with mechanisms for visualising the structure of problems from different viewpoints. Engineers are encouraged to manipulate the entities in their models in order to try different abstractions. The tools supporting program design methods are designed to enable them to build up an outline model of the problem in their mind. In contrast, if we look at formal methods, they concentrate on detailed behaviour rather than on problem structure. This is what formal notations are designed to tackle, in accurate precise detail. Formal notations do not provide the modelling constructs, let alone tools, that are available for informal notations. The engineer attempting a formal specification is faced with the need to make difficult and critical choices of model structure but has little support for such work.

Our hypothesis is that formal specification would be easier and of better quality if an informal transitory modelling stage were performed, as is done in program design. Fraser et al. have described such transitory modelling stages [8] and Bruel and France [4] have investigated the use of UML as an aid to producing formal specifications. In order to investigate this hypothesis further, we will look at the effect that such a transitory stage has on the specification process.

The aim will be to provide a method by which the outline model of the formal specification can be built using constructs and tools similar to the informal modelling methods. We shall attempt to demonstrate empirically that a suitable notation and tool to aid the modelling process has a significant beneficial effect. The hypothesis will be tested by comparing measures of effort and quality for the tool assisted, and unassisted, by the creation of a formal specification.

In order to support the findings of the formal experiments, case studies will be carried out with collaborating industrial partners. Several companies who have an interest in formal methods are using, or are in the process of introducing, informal modelling tools and will benefit from specifying parts of their systems in the assisted formal notation. Other companies who have more extensive experience of formal notations may also benefit from using the assisted formal notation for comparison with their previous experiences. This would give contrasting perspectives from the points of view of experienced and novice formal methods users.

5. Summary

We have carried out a survey of the opinions of practitioners who use formal methods for software specification and development. The size of the sample is small (five companies were visited) but covers a range of different market sectors including commercial computing systems, defence and avionics systems, and consumer products. The interviewees are experienced experts in the use of formal methods in real systems. The results cover a wide range of issues including the impact on the company, its products and development processes as well as pragmatics such as scalability, understandability and tools. The survey is the first stage of a sequence of empirical assessments that will assess some of the understandability problems and benefits that have been raised. The work that follows on from this survey will focus more narrowly on these areas and will involve formal experimentation and case studies. In particular, we will investigate empirically whether informal modelling methods have a significant effect on the effort involved in producing formal specifications and on the quality of the resulting specification.

6. Uncited References

[6]. [15]. [16].

Acknowledgements

The authors wish to acknowledge the support of UK EPSRC which has funded EMPAF (GR/L87347) and a PhD studentship for this work. We would also like to thank the companies and individuals who participated in this survey without whom this work would not have been possible.

Appendix A. Questionnaire

(1) How would you define a formal method?
(2) What experience has the company had with formal methods?
(3) Which formal methods have you used most?
(4) How big are the systems that you use formal methods on?
(4a) Does the size of the system affect the practicality of using formal methods?
(5) How do formal methods affect the software life cycle?
(6) How do formal methods affect software quality assurance activities (records, audits, certification, etc.)?
(7) What are the benefits that you have found?
(7a) Are they measurable in terms of the quality of software products?
(7b) Are they measurable in terms of software process improvements?
(7c) Has any quantitative data been collected that demonstrate the benefits?
(8) What problems have been encountered?
(8a) How do the problems affect the quality of software products?
(8b) How were they overcome?
(9) Have any understanding difficulties or benefits been found?
(9a) If so, has this affected the correctness and verification of the resulting code?
(9b) Is there any pattern to the misunderstanding i.e. particular constructs or styles?
(10) Do you use any style rules or codes of practice when writing formal specifications?
(10a) How do they affect understanding (if at all)?
(11) Have you found that the structure of the specification model influences the implementation?
(11a) Is this good or bad?
(12) How have formal specifications affected maintenance issues?
(12a) Do the formal specifications help to determine the correct code change?
(12b) Are the formal specifications difficult to update?

Are they kept up to date?

(12c) Do the formal specifications prevent (or worsen) degradation of the code structure through maintenance?

(12d) Does the structure of the specifications themselves deteriorate through maintenance?

(13) How do customers view the use of formal methods?

(13a) Are formal documents used as an interface to customers?

(13b) If so, how does this affect the understanding of the system by the customer?

(13c) Has it affected the system validation and acceptance stages?

(14) Is there anything we have not covered that you would like to talk about?

References

[1] Ministry of Defence: Def Stan 00-55, Requirements for Safety Related Software in Defence Equipment, issue 2, 1997.

[2] J. Bowen, M. Hinchey, Seven more myths of formal methods, IEEE Software 12 (4) (1995).

[3] T.M. Brookes, J.S. Fitzgerald, P. Larsen, Formal and informal specifications of a secure component: final results in a comparative study, FME '96: Industrial Benefit and Advances in Formal Methods, Springer, Berlin, 1996 (pp. 214–227).

[4] J. Bruel, R. France, Transforming UML models to formal specifications, Proceedings of the OOPSLA '98 Workshop on Formalising UML. Why? How?, 1998.

[5] J. Daly, Replication and a Multi-Method Approach to Empirical Software Engineering Research, PhD Thesis, University of Strathclyde, 1996.

[6] J. Draper, H. Treharne, T. Boyce, B. Ormsby, Evaluating the B-Method on an Avionics Example.

[7] N. Fenton, How effective are software engineering methods, Journal of Systems Software 22 (1993) 141–146.

[8] M.D. Fraser, K. Kumar, V.K. Vaishnavi, Strategies for incorporating formal specifications in software development, Committee of ACM 37 (10) (1994) 74–86.

[9] R. Glass, The software research crisis, IEEE Software (1994) 1994.

[10] A. Hall, Seven myths of formal methods, IEEE Software 9 (1990) 11–19.

[11] B. Kitchenham, Evaluating software engineering methods and tools — part 1, ACM SIGSOFT Software Engineering Notes 21 (1) (1996).

[12] B. Kitchenham, Evaluating software engineering methods and tools — part 2, ACM SIGSOFT Software Engineering Notes 21 (2) (1996).

[13] B. Kitchenham, S. Linkman, D. Law, Critical review of quantitative assessment, Software Engineering Journal (1994).

[14] P.B. Pannell, D.J. Pannell, Introduction to Social Surveying: Pitfalls, Potential Problems and Preferred Practices, SEA Working Paper 99/04, 1999 http://www.general.uwa.edu.au/u/dpannell/seameth3.htm.

[15] S. Pfleeger, L. Hatton, Investigating the influence of formal methods, IEEE Computer (1997).

[16] J. Wynekoop, N. Russo, Studying system development methodologies: an examination of research methods, Information Systems Journal 7 (1997) 47–65.