Verification and Formal Methods: Practice and Experience

J. C. P. Woodcock
University of York, UK
and
P. G. Larsen
Engineering College of Aarhus, Denmark
and
J. C. Bicarregui
STFC Rutherford Appleton Laboratory, UK
and
J. S. Fitzgerald
Newcastle University, UK

We describe the state of the art in the industrial use of formal verification technology. We report on a new survey of the use of formal methods in industry, and compare it with the most significant surveys carried out over the last 20 years. We review the literature on formal methods, and present a series of industrial projects undetaken over the last two decades. We draw some observations from these surveys and records of experience. Based on this, we discuss the issues surrounding the industrial adoption of formal methods. Finally, we look to the future and describe the development of a Verified Software Repository, part of the worldwide Verified Software Initiative. We introduce the initial projects being used to populate the repository, and describe the challenges they address.

Categories and Subject Descriptors: D.2.4 [**Software/Program Verification**]: Assertion checkers, Class invariants, Correctness proofs, Formal methods, Model checking, Programming by contract; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Assertions, Invariants, Logics of programs, Mechanical verification, Pre- and post-conditions, Specification techniques; F.4.1 [**Mathematical Logic**]: Mechanical theorem proving; I.2.2 [**Automatic Programming**]: Program verification.

Additional Key Words and Phrases: Experimental software engineering, formal methods surveys, Grand Challenges, Verified Software Initiative, Verified Software Repository.

1. INTRODUCTION

Formal verification, for both hardware and software, has been a topic of great significance for at least forty years. A substantial research community has developed,

Corresponding Author's address: Jim Woodcock, Department of Computer Science, University of York, Heslington, York YO10 5DD, UK; email: jim@cs.york.ac.uk.

focused on the problems of designing and validating computer systems, but the challenge has often been levelled at the community to show the industrial applicability of this technology. Several significant surveys in the 1990s fuelled a debate about the suitability of formal methods for industrial deployment.

The purpose of this survey is to give an appraisal of current practice and experience in the application of verification technology with a particular emphasis on industrial deployment. We focus on the lessons learned from industrial applications of formal methods in software development since those reviews of the 1990s, identifying the areas in which progress has been substantial, and identifying remaining challenges.

The paper is structured as follows. In Sect. 2, we revisit a number of influential surveys of the use of formal methods and verification technology in industry. We present the results of a new survey of industrial practice in Sect. 3. We discuss the results, comparing them to previous surveys. In Sect. 4, we describe a series of industrial projects conducted over the last 20 years. The projects represent a cross section of applications including national infrastructure, computer microcode, electronic finance, and security applications. Sect. 5 contains our observations about the current state of the art, drawn from what we found in our survey and in the literature. In Sect. 6, we consider the major issues in industrial adoption of formal methods and verification technology. In Sect. 7, we describe the Verified Software Repository that is being built as part of the Verified Software Initiative. Finally, in Sect. 8, we drawn some conclusions about the current practice and experience.

2. CHALLENGES IN THE INDUSTRIAL TAKE-UP OF FORMAL METHODS

The transfer of verification and formal methods technology into industrial use has been an objective of researchers and practitioners for several decades. The potential benefits for reduced defect densities in specifications, designs and code have to be achieved at reasonable cost and within the constraints of real industrial settings. By the early 1990s, questions were being asked about whether formal methods could ever be viable parts of industrial development processes. Several significant surveys from that time identified challenges to verification practice and experience that subsequent research has sought to address.

Austin and Parkin's 1993 survey [Austin and Parkin 1993], used a questionnaire to assess the uses made of formal methods in both research and application, and to gather opinions on the barriers to wider industry adoption. The majority of responses analysed (126) reported use of model-oriented formalisms and the vast majority concentrated on specification and proof rather than verification.

Craigen, Gerhart and Ralston's survey [Craigen et al. 1993a; 1993b] covered 12 case studies, each based on an application of formal techniques in an industrial setting. A combination of questionnaires, literature reviews and interviews was used to derive information on each application. The range of formalisms included model-oriented approaches, a process calculus (CSP) and verification environments. None of the studies addressed systems bigger than around 300k lines of code, and the majority were much smaller, in several cases because they were high integrity applications. The survey came to positive conclusions about the improving maturity of formal methods and the fact that they had been applied to significant systems.

Regulatory support for enhanced software engineering practices was important in providing increased motivation for adoption.

At the same time as these surveys, a growth of interest in formal methods was reflected in the literature positively advocating the technology [Hall 1990; Bowen and Hinchey 1995; 2006]), from a business and cost-effectiveness perspective [Thomas 1992] or providing comprehensive overviews [Rushby 1993; NASA 1997; 1998]. The perception of a long-term crisis in software development motivated a wider interest in the potential of verification [Gibbs 1994; Cuadrado 1994]. By the mid 1990s, it was possible to publish a collection [Hinchey and Bowen 1995] of 15 applications of formal methods each of what was claimed to be "an industrially useful scale".

In spite of the optimistic conclusions of some surveys, there seems to have been a general sense that formal techniques were failing to achieve the level of industry adoption that their advocates had hoped for. A round-table article in IEEE Software in 1996 [Saiedian 1996] showed the divergence of opinion. Hinchey and Bowen [Hinchey and Bowen 1996] felt that standards, tools and education would "make or break" industrial adoption, while Glass [Glass 1996] saw a chasm between academics who "see formal methods as inevitable" and practitioners who "see formal methods as irrelevant". Other articles in the same collection cite weaknesses in notations, tools and education as challenges to wider acceptance of this technology.

Bloomfield and Craigen's wide-ranging review [Bloomfield and Craigen 1999] includes evaluations of research programmes, major conferences and industrial application areas. A leading point is the suggestion that models of technology diffusion should consciously be applied to the formal methods adoption problem. Although they saw significant take-up in critical application domains, the authors identified several reasons for the general failure to adopt formal techniques.

The surveys take very different viewpoints. Some, such as Craigen et al., base conclusions on analyses of a selected group of applications. Others, such as Austin and Parkin, use a wider ranging survey of both industry and academia. Still others, such as Bloomfield and Craigen, use reviews of whole research programmes. Nevertheless, several significant common themes emerge as challenges to successful industrial adoption.

2.1 Deployment Strategies: Lightweight or Heavyweight Formal Methods?

The formalisms available, while believed by their users to be within the capabilities of typical engineers [Snook and Harrison 2001], seem to present a significant barrier to potential adopters. However, a formalism need not be applied in full depth over an entire product and through all stages of the development process. The term "lightweight formal methods" has been applied to various forms of application that do not entail such a deep application of the technology. For example, Jones' approach [Jones 1996] favours rigour over full formality, and the production of human-readable proofs that could be formalised if the need arises. Jackson and Wing [Jackson and Wing 1996] suggest a focused application of nonetheless fully formal techniques to partial problems.

2.2 The Importance of Tool Support

The case studies reported in surveys were predominantly applications of formal methods performed by technical specialists using tools that, with few exceptions,

were not felt to be rugged enough for wide-scale application. Almost all surveys in the area point to the importance of producing tools that are well enough supported to merit commercial application. Particular challenges identified include support for automated deduction, especially where human interaction with a proof tool is required, common formats for the interchange of models and analyses and the lack of responsive support for tools users in the case of the many tools offered on a "best efforts" basis. The development of such robust tooling involves addressing prosaic issues such as providing multi-language support, porting to a variety of platforms, version control and assistance for cooperative working by multiple engineers on single developments.

Craigen et al. [Craigen et al. 1993a; 1993b] observed that "Tool support, while necessary to the full industrialization process, has been found neither necessary nor sufficient for successful application of formal methods to date." Nevertheless, the immaturity of theories and tool bases mean that some successful applications require a "heroic" level of effort [Bloomfield and Craigen 1999]. Bloomfield and Craigen also comment that tools are developed but not transferred across platforms or research groups and that in tools are not always accompanied by advances in methodology or theory.

2.3 Disappearing Formal Methods

Craigen et al. indicate that tools need ultimately to be seen as integral parts of development environments while Clarke and Wing [Clarke and Wing 1996] anticipate formal verification technology being applied by developers "with as much ease as they use compilers". Indeed, Rushby raises the possibility that formal methods tools might disappear from view [Rushby 2000], allowing certain analyses to be invoked within existing development environments. There are some strong arguments in favour of this approach as a means of moving formal verification technology from innovators to first adopters [Bloomfield and Craigen 1999]. Such automated analyses can be integrated into existing development processes at the points where they are seen to be immediately beneficial. This can be done with less upheaval than the adoption of a complete new development framework.

2.4 Making the Business Case

Few authors believe that the business case for adoption of formal methods has been made convincingly. Among the barriers to industrial adoption, Austin and Parkin note the absence of demonstrations of cost-effectiveness for formal techniques. They also suggest that many of the problems of industry adoption were those naturally associated with process change. Craigen, Gerhart and Ralston point to the lack of a generally accepted cost model on which to base the discourse.

There have been some studies comparing developments with and without the use of formal techniques, such as Larsen et al. in the aerospace sector [Larsen et al. 1996]. Weaknesses were identified in the quantitative claims made for earlier projects [Finney and Fenton 1996], suggesting that strong empirical evidence is needed to encourage adoption. Bloomfield and Craigen comment that the results of scientific studies do not scale to engineering problems and that there has been over-selling and excessive expectation.

3. A SURVEY OF CURRENT USE AND TRENDS

3.1 Objectives

In order to ascertain the state of formal methods usage in industry today, we collected data on a number of industrial projects which we knew to have employed formal techniques. The underlying motivation was to help understand which aspects of development are adequately supported by formal tools and techniques, and which aspects are in need of further research in order to provide support which can be used in an industrial context.

The data collection was targeted towards gathering information regarding the current state and trends in informal methods uptake. We were interested in a better understanding the answers to the following questions:

—In what domains and to what classes of system have FMs been applied industrially?

—What kinds of methods, techniques and tools are being used in these different domains?

—Where in the development process are they being used?

—What if any is the trend over time of the application of FM?

The questions we asked concerned:

—type of software developed and the application domain in which it is applied,

—the timescale and size of the project undertaken,

—the tools and techniques used for different development activities,

—the experience and training provided to the personal involved, and

—the satisfaction with the adequacy of the techniques used.

3.2 Data Collection

Data was collected on a number of industrial projects known to have employed formal techniques. Using a structured questionnaire, we collected data on 48 projects directly from individuals who had been involved those projects. We supplemented this sample with data on a further set of 6 projects which was collected from the literature. Respondents were asked to complete a questionnaire relating to a formal methods project they were involved with. Where an individual had experience of more than one project, then separate questionnaires were completed for each project. Whilst recognizing that this was a highly biased sample, both in its selection from projects which were known to have employed formal methods, and in that its respondents would be subject to acquiescence and social biases, it was nevertheless possible to draw some conclusions concerning the relative penetration of formal methods into certain application domains and development activities.

The application. Data was gathered on the application domain and type of application the development related. Respondents were asked to choose one or more options from a predefined list of application domains such as: Transport, Nuclear, Defence, etc. and from a list of application types such as: Real-time, Parallel, Hardware, etc. Respondents were also asked to list any certification standards that applied to this application.

The project. Data was gathered on the timescale and size of the project. Respondents were asked to give the start and end dates of the project and any significant project milestones. Respondents were also asked to give an indication of the scale of the development in terms of code size, team size, cost etc.

Methodology. Respondents were asked to list the methods, notations, tools and techniques used and to describe for which life-cycle activities each of these was used.

Personnel. Data was collected on the composition and experience of the project team. Respondents were asked to state the roles which had been part of the project team and the level of experience of the individuals in those roles. They were also asked to describe any training that was provided to team members as part of the project.

Results. Data was collected on whether the use of formal methods had had an effect on the quality, cost and duration of the projects. Respondents were asked whether they had been able to assess the effect of the use formal techniques on the outcome of the project in each of these dimensions and if so whether this effect had been positive, neutral or negative.

Conclusions. Respondents were asked to rate their experience of the effectiveness of the techniques used against a 5 point scale and to provide metrics if possible to support their statements. For each of the following three statements they were asked to state whether they: Strongly disagree, Disagree, Mixed Option, Agree, Strongly Agree. The statements were:

—"The use of formal methods in this project was successful."

—"The techniques used were appropriate for the tasks required."

—"The tools used were able to cope with the tasks undertaken."

3.3 Survey Results

This section summaries the data collected from the 54 responses.

Whilst it is clear that due to the non-random nature of the sample it is impossible to generalise the conclusions, we still consider it valuable to describe the nature of the responses as this review is a survey of *industrial* use of formal methods.

The Application. The largest single application domain was Transport (26%) followed by Financial (22%). Other major sectors were Defense (14%), Telecommunications (10%) and Office and Administration (10%). Other areas were two or less responses ($\leq 5\%$) were: Nuclear, Healthcare, Consumer Electronics, Space, Semantic Web, Resource planning, Automated car parking, embedded software, Engineering, Manufacturing.

Some 20% of responses regarding the application domain additionally indicated that the projects related to software development tools themselves such as Operating Systems, Compilers, CASE tools, and a further 10% related to computing applications within the domain such as high performance computing, Runtime code optimisation, File system replication, Access Control, Communications protocols, Microcomputer design.

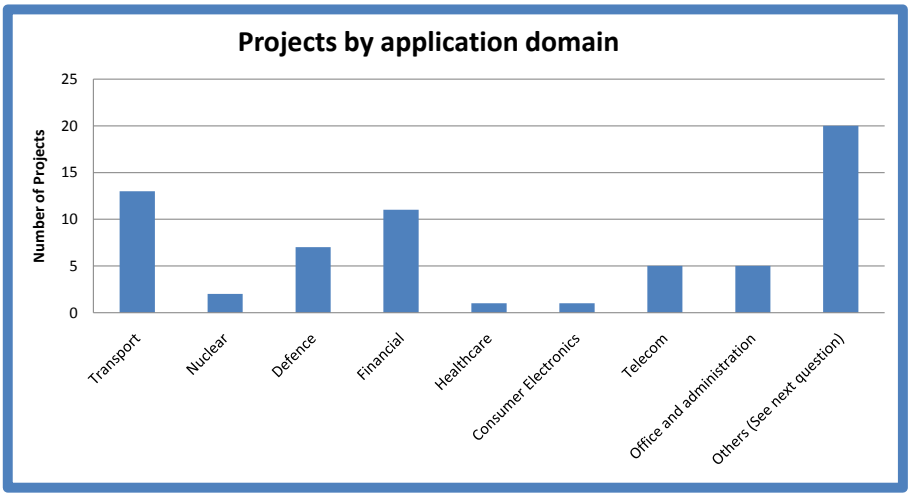

Fig. 1. Application Domains

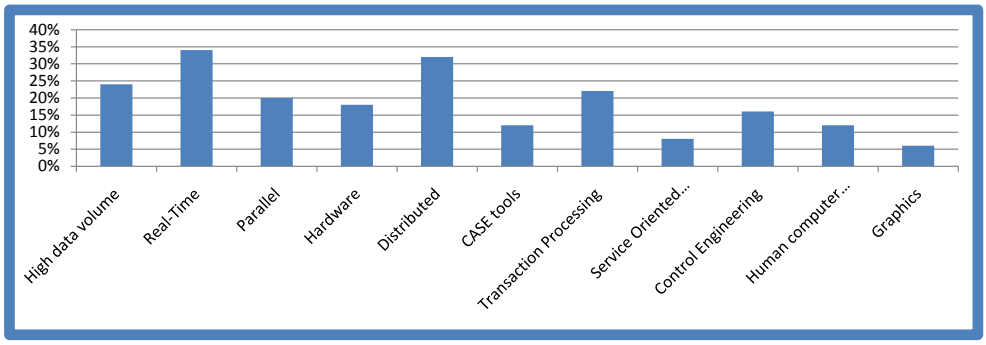

Fig. 2. Application Types

The largest groups of application types were Real-Time applciations (34%), distributed applications (32%), Transaction processing (22%) and Parallel (20%). Hardware was recorded by 18% of the responses and Control Engineering by 16%. Others included HCI (12%), Service Oriented Computing and Graphics.

Certification standards were indicated as a applying to the applciation in 34% of responses, notably IEC 61508 (10%) [IEC 1997] and Common Criteria (6%). Others included ITSEC Level E6, CENELEC EN50128, DO-178B Level A, Def Stan 00-55/6, and IEEE Standard 754 Floating Point Numbers.

The Project. Figure 3 presents the start dates of the projects. Not surprisingly the frequency of projects is higher in recent years although this could simply be a reflection of the way that the data was collected with more recent projects being more easily targetted.

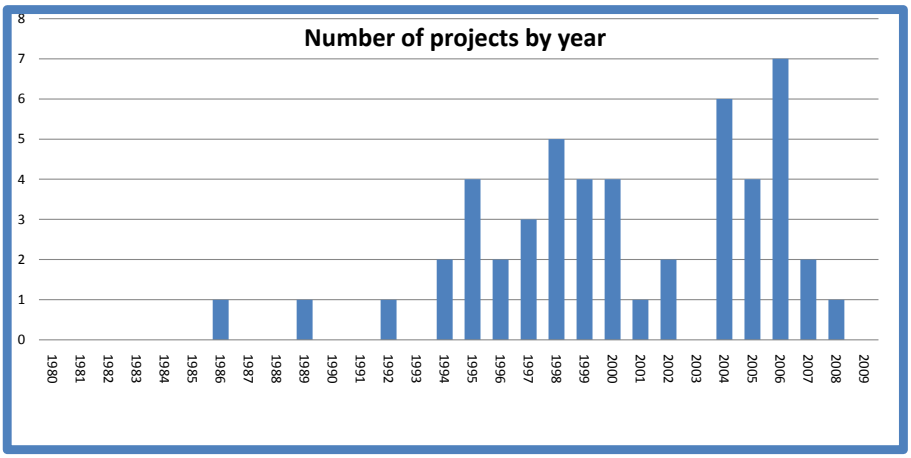

Fig. 3. Project start date

Software size. 56% of respondents gave indication of the size of the software in terms of lines of code. Of these, the split was roughly equal on a logarithmic scale between 1-10 KLOC (29%), 10-100 KLOC (39%) and 100-1000 KLOC (32%).

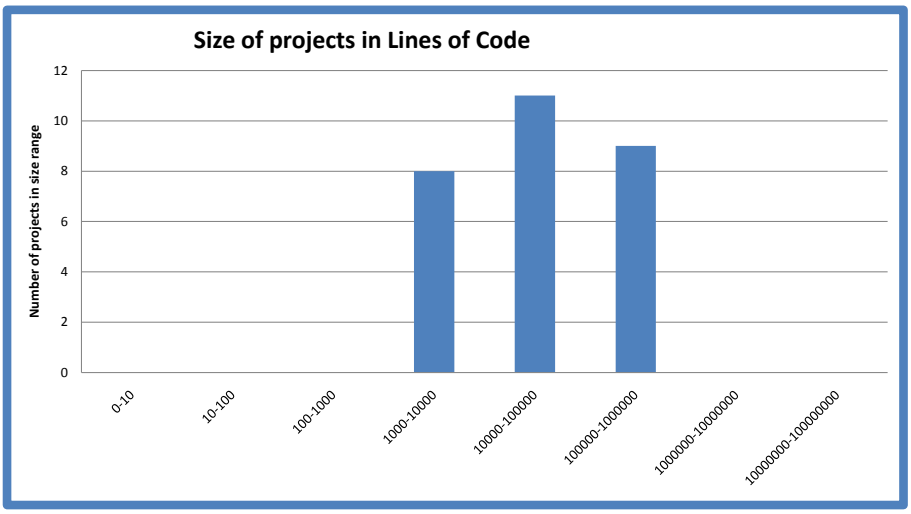

Fig. 4. Project size in Lines of Code

Methodology. Figure 5 presents the rates of use of various techniques such as Specification/Modelling, Execution (specification interpretation/testing), Inspection (of specification/model).

The use of these techniques was correlated against the date of the project and it was found that the use of Model Checking has increased from 14% in 1990s to 59% in this decade. This is a highly significant change. The use of Proof has decreased slightly in this period from 33% to 22% and the use of refinement

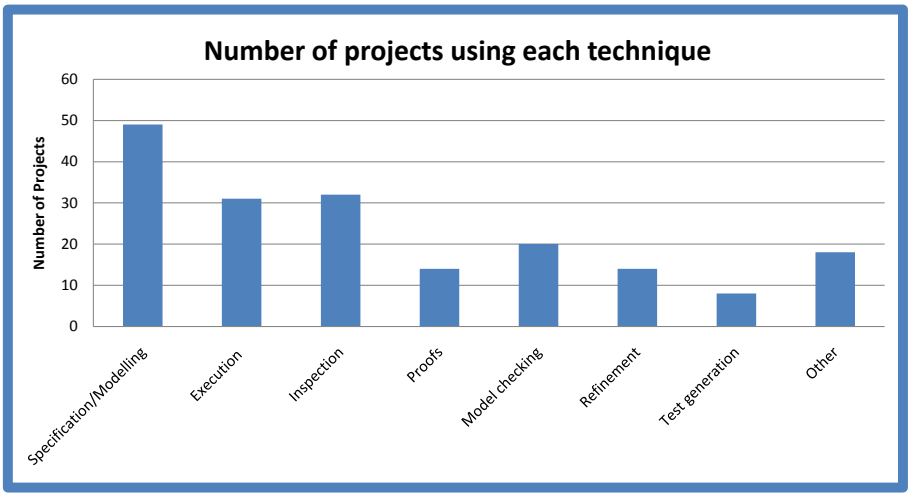

Fig. 5. Techniques used

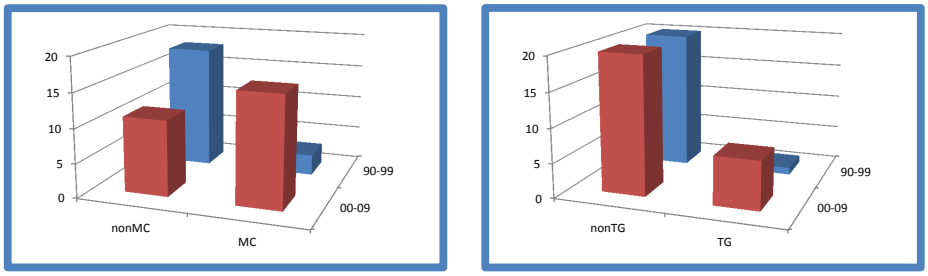

Fig. 6. Change in use of Model Checking and Test Generation by decade

has also decreased slightly from 33% to 22%, although neither of these changes are statistically significant. (The similarity between these last two set of figures is purely coincidental.) Similarly there had been no significant change is use of execution from 67% to 59%. On the other hand, the use of Test Generation had increased significantly from 5% to 26%.

Personnel. Interestingly, when asked to indicate which roles were part of the project team from a prediefined list of: Product management, Program management, Development, User experience, Test, Release Management, Architect, Other; the largest response was "Architect" (46%) rather than "Development" (14%). Other responses were all under 10%.

Regarding previous expertese in the techniques used, 50% reported "Considerable previous experience", 48% reported "Some previous experience" and 16% reported "No previous expertese". The total adding up to more than 100% because some respondents reported a mixed teams of more than one category.

Of those reporting "No previous expertise", one half (8%) were in a mixed team with more experienced collegues, leaving the remaining 8% which introducing tecniques to a team not previously experienced in thse techniques.

Regarding training 36% reported "No training given", 56% reported "Some training given" and 4% reported "considerable training given".

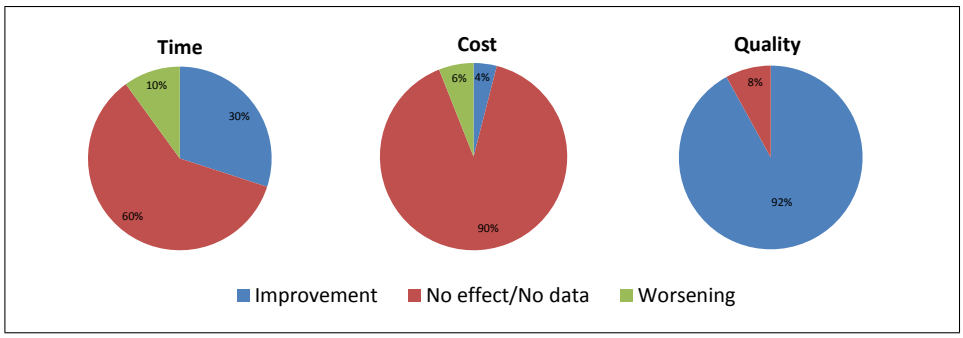

Fig. 7. Did the use of formal techniques have an effect on time, cost and quality?

Outcomes - the effect on Time, Cost and Quality.

Time. The effect of the use of formal techniques on time taken to do the work was on average positive. Of those cases where a view was expressed (40%), three times as many reported a reduction in time (30%) as compared to those reporting an increase in time (10%). Many responses (60%) indicated that it was difficult to judge the effect on time taken although several noted increased time in specification phase which may or may not have been compensated for by decreasing time later. For example:

> "Difficult to judge but extensive specification phase probably added elapsed time, although it may have saved time later"
>
> "Long specification phase probably added to elapsed time"
>
> "Modest increase in time spend during early design ... was recovered many times over during system integration and testing."

Cost. There was little useful data on the effect of the use of formal techniques on cost with only 10% of cases expressing a view. Of these, slightly fewer reported a cost decrease (4%) compared to those reporting and increase in cost (6%).

Some notable comments with respect to the effect on cost include:

> "We observed a substantial improvement in productivity once the code generation subsystem of the tool had been bootstrapped, due to the use of code generation from specifications. ..."
>
> "The cost increase was largely due to the lack of precise, complete information about the required externally visible behavior of the software product ... Once the code was implemented the required behavior was clear, and applying formal specification and formal verification was relatively straightforward. The one expensive part of the code verification process was the annotation of the code with pre- and postconditions.

> Once the annotated code was available, showing the correspondence between the annotated code and the abstract specification of the required behavior was straightforward. This latter process included adding more annotations and correcting annotations that were incorrect. During this process, the abstract specification and the required security properties changed very little."

Quality. In contrast to the above results, the use of formal techniques is believed strongly to have improved quality with 92% of all cases reporting an increase in quality compared to other techniques and no cases reporting a decrease in quality.

Of the descriptions of improvements in quality reported many were related to the detection of faults (36%). Other common reasons given for improvement in quality were: improvements in design (12%), increased confidence in correctness (10%), improved understanding (10%), and early identification of faults or other issues (4%).

Conclusions. As shown in Figure 8 the respondents to the survey have generally speaking been positively about the success with the use of FM in the projects that has been reported. However, one would also expect that stakeholders that did not see the use of FM in their project as a success would spend the time to fill out the questionnaire. Figure 9 illustrates that the respondents in general is also satisfied with the FM techniques used in their projects whereas figure 10 shows that in 10% of the projects that tools applied have not fully been able to live up to expectations. Finally figure 11 also shows that a majority of the respondents wish to use a similar technology again on new projects.

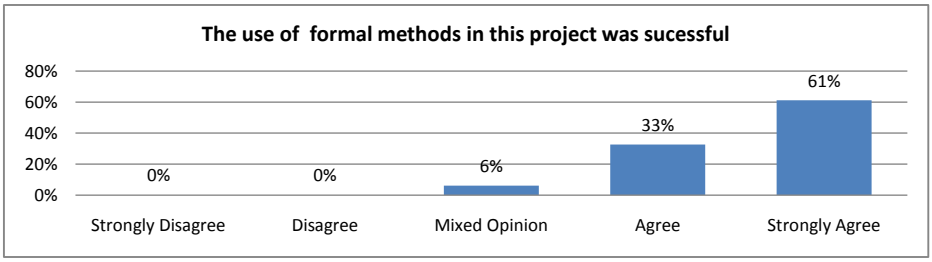

Fig. 8. Overall satisfaction with the used of formal techniques

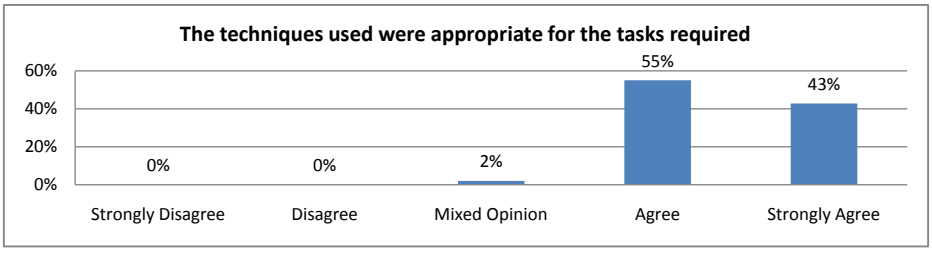

Fig. 9. Overall satisfaction with the techniques used

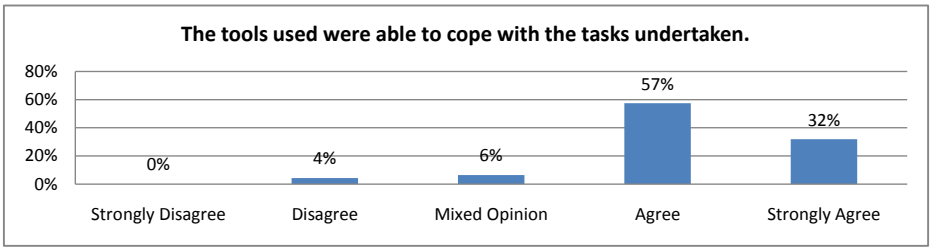

Fig. 10. Overall satisfaction with the tools used

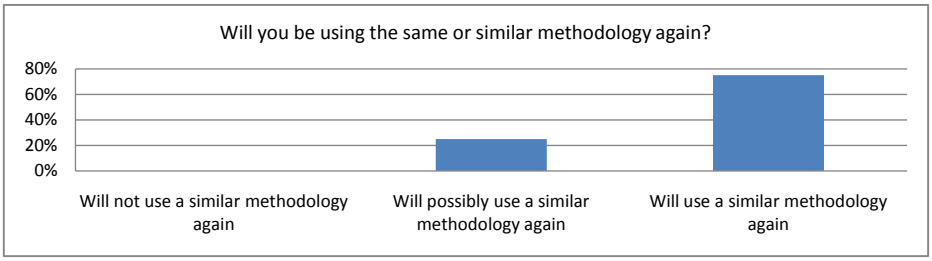

Fig. 11. Intention to use formal techniques again

3.4 Discussion on the results of the survey

Market penetration of use of formal techniques. The survey results show that the take up of formal techniques is distributed across a wide range of application domains with some activity in just about every application domain. Similarly, there are examples of use of formal techniques in a large number of different types of application including a considerable number related to the development of the software development tools themselves. This reflects perhaps the "wide spectrum" approach of many techniques which have been developed without any particular focus on application domain or type of software and, arguably, little specialisation of techniques for particular types of development. It could be argued that this "scattergun" approach dilutes the evidence and thus limits markets penetration in any particular domain.

Tools and techniques. The projects reported in the survey could be classified into two types by their ambition with respect to the extent of formal analysis undertaken. Some projects undertook "deep" analysis charecterised by extensive and comprehensive analysis, significant manual intervention ion the formal analysis, and therefore a major change to the informal techniques. On the other hand, some projects were more intereted in "broad" analysis, hoping for some benefit from little or no manual effort, a "something for nothing" one might say.

It is clear that Moore's law has had a very significant impact on the availablity of computational power over the two decades spanned by the projects reported in the survey, with typical desktop computing and storage resources increasing by perhaps some 10,000 times in that period. Together with significant theoretical advances, this has had a major impact on the scale of problem which can be addressed by automated techniques such as model checking and would seem to indicate a continuing

increase in the level of automation and move towards fomral methods disappearing "under the hood" in due course. Such a trend is epitomised by the following quotation from project whcih employed significant levels of model checking:

> "To be useful during product development, FM needs to provide answers in seconds or minutes rather than days. Model-checking can do this very effectively when applied to the right kinds of system designs. To take advantage of this, model-checking has to be tightly integrated into the commercial design and verification tools that developers are already using."

One might speculate as to what further automation might be possible in future with Moore's law inexerably replacing specialist expertese with automated analysis. To what degree, and in what timescale, will mathematical proof succumb to Moore's law? One respondent reflected on the tools available as follow:

> "Tools for formal methods still very weak compared with what is theoretically possible"

The effect on time, cost and quality. It is surprising perhaps how little data is available in the responses on the costs, relative or absolute, of employing formal techniques. On the other hand, the overall value of their use amongst the respondents was clear.

> "No precise measurements have been done. By rule-of-thumb it was estimated that the design effort exceeded a conventional effort by about 10%. Presumably maintenance costs were reduced drastically. But this judgement is based only on hypothetically assumed fault rates of software developed conventionally."

This prompts one to question how such a judgement can be made without evidence of cost. The answer comes perhaps from the fact that the value judgement is based on a perceived increase in absolute quality rather than on value for money. Arguably, this approach may be being taken because the estimation and assessment of cost of software development are so poorly understood in general that basis data is not available for a relative judgement on the cost of formal techniques to be possible. This also supports the "something for nothing" approach where an extra technique of very low cost is perceived as useful, whereas a more substantial investment cannot be justified.

> "The techniques were much more scalable when analyzing finite-state models [. . .] than with "infinite-state" models, which required the use of abstractions and a combination of model checking and theorem proving. We would argue that finite-state model checking is a mature technology that should be used on critical finite state models as a matter of standard engineering practice in a model-based development environment. The tools are straightforward to learn for practicing engineers and automated translation tools can be used to transform MBD notations into the native languages of the analysis tools. The analysis of "industrial-sized" numerically-intensive models is much more difficult using either

theorem proving or model checking; for model checking the scalability of the tools and the restriction to linear models for current decision procedures makes analysis extremely challenging. For theorem proving, the scale of the models makes the proof effort very large.

4. HIGHLIGHTED PROJECTS

We present a series of selected industrial formal methods projects in this section. We have chosen them to show a cross-section of projects over the last two decades. The emphasis is on developing verified systems cost effectively, and the applications include microcode and firmware, railways, national infrastructure, smart cards, and a biometric-based security application.

We start in Sect. 4.1 with the inmos transputer project, which was carried out in 1988. It was an influential project in its day, showing how the industrial use of formal methods could give substantial cost reductions over traditional methods. We follow this with an overview in Sect. 4.2 of the long-term use of the B-method in railway signalling and automatic train control. B has been used over the last 20 years to produce thousands of lines of safety-critical code responsible for billions of passenger journeys. The next two projects, Sects 4.3 and 4.4, were part of a much larger enterprise to develop smart cards for electronic finance. The Mondex project was influential in the late nineties, showing that formal methods could be cost-effectively applied to assurance at the highest level (then, ITSEC Level E6). The extensive proofs that were carried out were done manually, a decision taken at the time to keep costs under control. Recent work has shown that this was overly cautious, and that Moore's Law has swung the balance further in favour of cost-effective mechanical verification.

In Sect. 4.5, we describe the verification of the AAMP5 microprocessor at Rockwell Collins during 1996–98. Rockwell demonstrate through a number of experiments how the cost of using formal methods can be reduced by repeated application within a problem domain (by as much as an order of magnitude).

The development of the software for the Maeslant Kering Storm Surge Barrier in 1997 is described in Sect. 4.6. This is a high-profile project, with the barrier helping to guarantee the safety of South Holland, a province situated on the North Sea in the western part of the Netherlands.

The final two projects show the state of current work. Sect. 4.7 reports on the development in 2005 by Praxis of the Tokeneer ID Station for the US National Security Agency. This shows that it is possible to produce a high-quality, low-defect system in a cost effective manner following a process that conforms to the Common Criteria EAL5 requirements (and beyond). The Mobile FeliCa project in 2006 (described in in Sect. 4.8) produced firmware for a new generation of IC chips with strict timing requirements for consumer electronics.

4.1 The Transputer project

The transputer [inmos ltd 1988b] is a microprocessor chip designed specifically for parallel processing; Gibbons describes the development of the T800 transputer floating-point unit [Gibbons 1993]. The T800 transputer combined a 32-bit reduced instruction set central processing unit, some memory, and four bidirectional

communications links on a single chip. It also had a floating-point arithmetic unit. Its successor, the T9000, was rather more sophisticated, with richer connectivity, memory model, and pipelined processor. A comprehensively redesigned 32-bit transputer intended for embedded applications, the ST20 series, was subsequently produced, based on T9000 technology. Today the ST20 core is still very widely used in chip-sets for set-top box and GPS applications.

The programming language for the transputer is occam [inmos ltd 1988a; Jones and Goldsmith 1988], a simple, low-level, executable subset of CSP [Hoare 1985]. It allows programmers to exploit parallelism in a simple and natural way.

Inmos started to develop the T800 transputer in 1986, using a conventional approach that required months of testing, since floating-point units are notoriously complex devices and prone to design bugs. As the extent of the required testing became clear, work started on the formal development of a correct-by-construction floating-point unit [Shepherd 1988; Barrett 1987; 1989; Shepherd and Wilson 1989; Barrett 1990; May et al. 1992]. The formal development took less time and money than the traditional approach, and it identified an ambiguity in a competing floating-point chip. Oxford University and inmos jointly received the Queen's Award for Technological Achievement in 1990 "to recognise and encourage outstanding achievements advancing process or product technology in the UK".

The formal development started by formalising in Z the IEEE-754 standard for floating-point arithmetic [Institute of Electrical and Electronic Engineers 1985], which was written in natural language. The Z notation [Spivey 1989; Woodcock and Davies 1996] was chosen because of its suitability for the task and the presence of extensive local expertise. The specification is described in [Barrett 1987; 1989]. This formalisation revealed some problems in the standard. For example, there was a problem with the treatment of `NaN`s ("not a number"), the result of an invalid operation, such as taking the square root of a negative number. The standard requires that diagnostic information about the cause of such a problem be propagated through further operations. But this is not always possible. Another example involved underflow: the standard is ambiguous as to whether an underflow exception should be raised when an operation produces a positive result that is too small to be represented, and which is subsequently rounded up to the smallest representable positive number.

The other departure point for the verification was a floating-point package written in occam and used in previous transputers, and the next task was to show using Hoare logic [Hoare 1969] that this was a refinement of IEEE-754. The attempted verification revealed errors in rounding and remainder operations. Barrett later remarked to Gibbons that "it was only a very small class of test vectors that would have shown up the errors" [Gibbons 1993]. With corrections in place, the occam package was verified correct.

The occam package now served as an intermediate representation of the functionality of the required microcode. It was too abstract to be directly useful for hardware design, so the occam transformation system [Goldsmith et al. 1987] was used to apply the laws of occam programming [Roscoe and Hoare 1988] to produce an equivalent microcode program.

The development of the floating-point unit, from natural language to silicon, was

at least three months [Barrett 1990] faster than the alternative informal development, which ran concurrently. Each month's delay in production was estimated to cost US$1M [Barrett 1990].

Gibbons reports that exactly two bugs have been found in the floating-point microcode [Gibbons 1993]. The first was introduced by a bug in the translation program that converted the micro-level occam into assembly code for the chip. The second was introduced by a hand optimisation of the machine-generated assembly code.

4.2 Railway signalling and train control

In 1988, GEC Alsthom, MATRA Transport, and RATP started working on a computerised signalling system for controlling RER commuter trains in Paris [Bowen and Stavridou 1993]. The objective of the project was to increase traffic movements on the network by 25%, whilst preserving the high level of safety provided by the existing system. The resulting SACEM system with embedded hardware and software was delivered in 1989 and has since been controlling the speed of all trains on the RER Line A in Paris. The dependability of SACEM has had obvious safety implications for the seven billion passengers who have used it since its introduction.

The SACEM software consists of 21,000 lines of Modula-2 code, of which 63% is regarded as safety-critical and has been subjected to formal specification and verification [Guiho and Hennebert 1990; Hennebert and Guiho 1993]. The specification was constructed in the B language and the proofs were done interactively using automatically generated verification conditions for the code. The validation effort for the entire system (including non-safety-critical procedures) was about 100 man-years. Hennebert and Guiho [Guiho and Hennebert 1990] claim that the system is safer as a result of the formal specification and verification exercise. The project team reported a difficulty in communication between the verifiers and the signalling engineers, who were not familiar with the B-method. This was overcome by providing the engineers with a French description derived manually from the formal specification. The SACEM system is further described in [Guiho and Hennebert 1990], which presents various dependability requirements and their implementation. Techniques to ensure safety include on-line error detection, software validation, and fault tolerance of the onboard-ground compound system.

Abrial reports on two further railway projects carried out using B [Abrial 2007]. The first is Line 14 of the Paris Métro, a system that has been working since October 1998 [Behm et al. 1999], and the second is the driverless shuttle at Paris Roissy Airport, became operational in 2007 [Badeau and Amelot 2005]. Only the safety critical parts were developed using B, representing one third of each software system. Table I (taken from [Abrial 2007]) shows the main characteristics of the two systems. Since Line 14 is completely automatic, the safety critical part concerns the running and stopping of trains, and the opening and closing of train and platform doors. The program is distributed into three different kinds of sub-system: the wayside equipment (several of these along the tracks), the on-board equipment (one instance in each train), and the line equipment (one instance), which are all heavily interconnected. In each sub-system, the safety parts that were developed using B are sequential and cyclic (350 ms), and constitute a single non-interruptible task.

	Paris Métro Line 14	***Roissy Shuttle***
Line length	8.5km	3.3km
Number of stops	8	5
Inter-train time	115s	105s
Speed	40km/hr	26km/hr
Number of trains	17	14
Passengers/day	350,000	40,000
Number of lines of Ada	86,000	158,000
Number of lines of B	115,000	183,000
Number of proofs	27,800	43,610
Interactive proof percentage	8.1	3.3
Interactive proof effort (person-months)	7.1	4.6

Table I. Statistics for the Paris Métro Line 14 and Roissy Shuttle projects.

The Roissy Airport shuttle system is derived from the light shuttle system of Chicago's O'Hare Airport. The difference between the two is that the former has several significant computerised parts located along the tracks, the Wayside Control Units, linked by an Ethernet network. These units drive the trains by sending them predefined speed programs that they have to follow. The requirements documents had to be modified and extended in order to deal with the new functionality of the Roissy Airport shuttle. This caused a number of problems that were discovered only during the development of the model. Similar train control systems are presently being developed with these techniques for the New York City subway, the Barcelona subway, the Prague subway, and Line 1 of the Paris Métro.

No unit tests were performed for the Line 14 or Roissy Shuttle projects. They were replaced by some global tests which were all successful.

4.3 Mondex

The following description of the formalisation of Mondex is taken from [Woodcock et al. 2008]. In the early 1990s, *platform seven* (at that time part of the National Westminster Bank, a UK high-street bank) started developing a smartcard-based electronic cash system, *Mondex*. This is an application designed to work like electronic cash, suitable for low-value cash-like transactions, with no third-party involvement, and no cost per transaction. Once the issuing bank has loaded electronic cash into the system, it has very little further control over it: just like real cash. A discussion of the security requirements can be found in [Stepney et al. 2000; Woodcock et al. 2008]; a description of some wider requirements can be found in [Aydal et al. 2007].

Because of the lack of third-party control, it is crucial that the security of the card cannot be broken; otherwise criminals could electronically "print" money with ease. So *platform seven* decided to develop the full Mondex product to one of the very highest standards available at the time: ITSEC Level E6 [ITSEC 1991]. This mandates stringent requirements on software design, development, testing, and documentation procedures, and also mandates the use of formal methods to specify the high-level abstract security policy model, to specify the lower-level concrete architectural design, and to provide a formal proof of correspondence between the two levels, in order to show that the concrete design enjoys the abstract security

properties.

The target platform smartcard (state-of-the-art in the early 1990s) had an 8-bit microprocessor, a low clock speed, limited memory (256 bytes of dynamic RAM, and a few kilobytes of slower EEPROM), and no built-in operating system support for tasks such as memory management. Also, power could be withdrawn at any point during the processing of a transaction, if say the card was withdrawn from the reader. Engineers at *platform seven* designed and implemented the secure cash-transfer protocol under these severe space and speed constraints. Early in 1994, they contacted Logica to deliver the specification and proof requirements of the E6 development. Logica's formal methods team chose the Z notation [Spivey 1989; Woodcock and Davies 1996] in which to write the two specifications and to perform the correspondence proof.

There was little difficulty in formalising the concrete architectural design from the existing semi-formal design documents, but the task of producing an abstract security policy model that both captured the desired security properties (in particular, that "no value is created" and that "all value is accounted for") and provably corresponded to the lower-level specification, was much harder. A very small change in the design (very small from the perspective of the formal methods team, that is) would have made the abstraction much easier, but was deemed by *platform seven* to be too expensive to implement, as the parallel implementation work was already well beyond that point. Eventually, after much experimentation with different approaches, an abstraction of the original design was successfully constructed.

Proof obligations were discharged by hand, with more than 200 pages of mathematics. The proof discovered a small flaw in one of the minor protocols; this was presented to *platform seven* in the form of a security-compromising scenario. Since this constituted a real security problem, the design was changed to rectify it. At the time, the impression was that the expense of automatic proof would dominate the cost of the project.

In 1999, Mondex achieved its ITSEC Level E6 certificate: the very first product ever to do so. As a part of the ITSEC E6 process, the entire Mondex development was additionally subject to rigorous testing, which was itself evaluated. Mondex International acquired the Mondex product and its sister product Multos after the completion of the evaluation and the award of ITSEC E6. No errors were found in Mondex in any part of the system subjected to the use of formal methods.

The project was revived in 2006, as the first pilot project for the Grand Challenge in Verified Software [Jones et al. 2006; Woodcock 2006; Woodcock and Banach 2007], an ambitious, international, long-term programme of research for achieving a substantial and useful body of code that has been formally verified to the highest standards of rigour and accuracy.

Preliminary technical work was started with the Mondex case study, which demonstrated how different research groups around the world can collaborate and compete in scientific experiments, and to generate artifacts to populate the Repository [Bicarregui et al. 2006]. The main objective was to test how the state of the art in mechanical verification had moved on in ten years. Eight groups took up the challenge (Alloy [Ramananandro 2008], ASM [Haneberg et al. 2008], Event-B [Butler and Yadav 2008], OCL [Kuhlmann and Gogolla 2008], PerfectDeveloper,

π-calculus, Raise [George and Haxthausen 2008], Z [Freitas and Woodcock 2008]). The main results from this project show that mechanising the proof of correctness of Mondex is clearly within the state of the art. In particular, the cost of mechanising the Z proofs of the original project is just 10% of the development cost, and so does not dominate costs as initially believed. Interestingly, almost all techniques achieved the same level of automation, producing the same number of verification conditions and requiring the same effort.

4.4 Multos

Multos (Multi-Application Smart Card Operating System) [France-Massey 2005] is a smartcard product produced by Mondex International (MXI) on a non-proprietary open systems basis. When Multos was developed in the late 1990s, smartcards had been used for a number of years, with applications such as Mondex electronic cash (see Sect. 4.3) proving highly successful. But after this initial success, suppliers began to question the business model. The infrastructure required for smartcards was based on single-application smartcards, which are costly to adapt and limit the number of functions that can be made available to customers. MULTOS multi-application smartcards allow many different functions to reside securely on a single card, thus increasing the number of functions on the card while reducing their cost of administration and distribution.

A crucial part of the Multos system is its *Certification Authority* (CA), which was developed for MXI by Praxis [Hall and Chapman 2002]. The CA is used to issue new cards containing essential system information and signed electronic certificates, without which new applications cannot be loaded onto Multos cards. The development cost, functional correctness, performance, and usability of the Multos CA is absolutely vital to the security and commercial success of the Multos product. To satisfy this mixture of requirements, a distributed architecture is used, blending COTS hardware and infrastructure software. The CA was to be certified to UK ITSEC Level E6, which approximates to Common Criteria Level 7 [CCRA 2006], requiring a rigorous development process, including the use of formal methods in early system development. The use of E6 ensures that both the customer and the supplier explicitly and unambiguously understand system requirements, avoiding the need for late changes in development.

Praxis used a correctness-by-construction process, starting from an abstract description of requirements, and proceeding by a series of refinement steps, each introducing a design concept, and each correct with respect to its predecessor. By taking small steps, the verification remains tractable. The process started with understanding the business requirements for Multos, which were captured using Praxis own-brand Reveal requirements engineering method. User requirements were validated by interaction with the client and checked for consistency. ITSEC Level E6 requires a Formal Security Policy Model (FSPM), which was formalised from an informal security policy that identified assets, threats, and countermeasures containing 28 technical items. Of these, 23 were formalised; the remaining five dealt exclusively with implementation details.

Praxis used Z [Spivey 1989; Woodcock and Davies 1996] to express the FSPM, based on the UK Communications-Electronics Security Group's Manual F [CESG 1995], which uses the Z notation. No proofs of correctness were carried out, although

extensive reviews were conducted. Information separation was harder to express in Z than the other properties, although Z proved a good choice of language overall.

Next, the system was specified and an architecture defined. The abstract system specification was divided into two parts: the user interface specification and the formal top-level specification (FTLS). The former was validated by operational staff experimenting with a user-interface prototype. The FTLS defines the functionality behind the user interface and was validated against the requirements and by client review.

The process structure was modelled in CSP [Hoare 1985]. Z operations and interprocess communications in the FTLS were mapped to CSP events. The resulting model was automatically checked for deadlock freedom using the FDR model checker produced by Formal Systems (Europe) [Goldsmith 2005]. A further check was performed to make sure that no security-critical functions were executed concurrently. These checks found significant flaws in the first design and gave much greater confidence in the final one.

A systematic translation was made from CSP processes to Ada tasks. The code produced worked first time, a rare event in distributed and concurrent programming.

The number of system faults is low compared with systems developed using less formal approaches. The delivered system satisfies its users, performs well, and is highly reliable [Hall and Chapman 2002]. In the first year of productive use, four faults were found, *and fixed as part of Praxis' standard warranty.* Altogether, there are 100 kLoC, so the defect rate is just 0.04 defects per kLoC, far better than the industry average for new developments. Jones [Jones 2000] reports that CMM (Capability Maturity Model) organisations at Levels 1–5 typically produce 7.5, 6.2, 4.8, 2.3, 1.0 defects per thousand lines of code, respectively.

Overall productivity on the development, taking into account all project activities, was 28 lines of code per day. This compares with typical industry figures of 10 lines per day [Jones 2000]. Combining a low defect rate with high productivity, Praxis are claim that "Multos is 2.5 times as reliable as the space shuttle software, yet at a fifth of the industry standard cost".

Two significant conclusions can be drawn from this work concerning the practicality of formal methods, and the choice of programming language. Firstly, using formal methods, as required by the higher levels of ITSEC and the Common Criteria, is practical, and they do reduce the number of late-discovered errors and thus the overall system cost. SPARK's support for strong static analysis and proof of program properties (for example, partial correctness or exception freedom) allows developers to meet the Common Criteria requirements for formal development processes cost effectively. Secondly, the language subset's simplicity and the data and information-flow analysis offered by the Examiner make a large class of common errors simply impossible to express in SPARK [Amey 2002].

4.5 Rockwell Collins

Miller reports experiences from several formal methods projects carried out in Rockwell Collins [Miller 1998]. One of their earliest experiments was to specify and refine a micro Real Time Executive μRTE in the RAISE notation, RSL [Group 1992; RAISE Method Group 1995]. This was not a successful a project: the language

was thought to be too complex and required substantial training, and the only tools were syntax and type checkers, so there was little confidence in the correctness of manual proofs.

A subsequent experiment with SRI International, sponsored by NASA Langley [Miller and Srivas 1995; Srivas and Miller 1995], formally verified the microcode for the AAMP5 microprocessor using PVS [Owre et al. 1992]. The AAMP5 is a proprietary microprocessor widely used in Rockwell Collins products. It has a stack-based architecture, a large instruction set, makes extensive use of microcode, has a pipelined architecture, and contains complex processing units such as an LFU (Look Ahead Fetch Unit). It contains approximately 500,000 transistors and provides performance somewhere between an Intel 386 and 486. They specified the AAMP5 at both the register transfer and instruction set level, with an abstraction function mapping between the two to prove the correctness of microcode instructions. The main lesson learned from the AAMP5 project was that it was technically possible to prove the correctness of microcode, and that their engineers can read and write formal specifications.

Two errors were found in the AAMP5 microcode while creating the specification, and this convinced them that there is value in just writing a formal specification. But they also convinced themselves that mechanical proofs of correctness provide a very high level of assurance. They did this by seeding two very subtle errors in the microcode that they delivered to SRI, and then waiting to see if they would find them. SRI did indeed discover the errors using a systematic process: the only way not to have found the errors would have been to fail to carry out the proofs.

The biggest problem with the AAMP5 project was that the cost was too high: more than 300 hours per instruction. This figure appears to have been inflated for a variety of reasons, including the steep learning curve using PVS for the first time and the need to develop many supporting theories. They knew their costs would drop dramatically the next time around, but they could not predict by how much, so they undertook a second experiment[Miller et al. 1996], the verification of the microcode in the AAMP-FV, again sponsored by NASA and with SRI. The goal was to demonstrate dramatic reduction in cost through reuse of the AAMP5 infrastructure and expertise.

Significantly, the AAMP-FV project confirmed that the expertise gained on the AAMP5 project could be exploited to dramatically reduce the cost of formal verification. Of the 80 AAMP-FV instructions, 54 were proven correct, and the cost of their verification dropped by almost an order of magnitude from that of the AAMP5 project. But as more complex instructions were attempted, proof techniques first developed on the AAMP5 project broke down and new approaches had to be devised. This phase progressed more as an exploratory project, with a steep learning curve and unexpected delays. One of the main contributions of the AAMP-FV project was the development of methods to handle instructions with complex microcode.

4.6 The Maeslant Kering Storm Surge Barrier

The *Maeslant Kering* is a movable barrier protecting the port of Rotterdam from flooding as a result of adverse weather and sea conditions. The decision to deploy and to reopen the barrier is made on the basis of meteorological data by a comput-

ing system because the probability of human error in making such a decision was considered so high (at around 10^{-3}) that it would undermine the required reliability for the system (order 10^{-4}). In terms of the international standard IEC61508 [IEC 1997] the application was placed at Safety Integrity Level (SIL) 4, for which the use of formal methods is "highly recommended".

The developers (CMG) were deliberately cautious [Kars 1997; Tretmans et al. 2001; Wijbrans et al. 2008] in defining goals for a formal methods deployment. Refinement technology was not felt to be feasible for a system of this scale. It was also felt to be too high-risk an option to introduce several new techniques in one project. The approach was therefore to integrate modelling and verification technology within the normal design trajectory. The approach used formal modelling and verification technology in the analysis, design and realisation phases of system development. The focus of the formal work was the decision-making subsystem and its interfaces to the environment.

Data and operations were modelled in Z, and this was embedded into a Promela model describing control. Designs were validated using the SPIN model checker. Promela and SPIN were selected because of the developers' prior experience with the tool and a perceived ease of use, meaning that the CMG engineers could perform most of the modelling and analysis work without having to bring in outside assistance. The Promela/Z linkage was ad hoc and did not itself have a formal semantics.

The validation tasks utilised models consisting of user process interacting with an environment model through an interface. An iterative approach was used in which functional properties of the user processes and interface were verified against a "friendly" or fault-free environment. Defects were then corrected and the validation again performed but with an environment model containing more complex fault models. The approach was regarded as highly beneficial in identifying flaws at an early enough stage and in a convincing enough way that the commissioning authority could be persuaded to re-specify significant parts of the product.

The final detailed design specified 29 programs with 20,000 lines of Z. Implementation was done via systematic coding in a safe subset of C++. There was no formal code verification. The final implementation was 200kLoC for the operational system and 250kLoC of supporting code (simulators, test systems *etc*). Informal but systematic test derivation from the Z model resulted in 80-90% code coverage for black box testing, with the remainder covered by white box tests. The problems raised during the process were logged; about 85% of them arose during development phases and around 15% during reliability and acceptance test. The residual faults have been minor. About half the faults detected during development were in code or design, the remainder being weaknesses in test specifications, configuration parameters or documentation.

The experience was seen as largely positive. The experience report [Tretmans et al. 2001] deliberately echoes Hall's Seven Myths [Hall 1990]. The developed software was believed by the developers to be of significantly better quality that would otherwise have been achieved, and that this quality benefit would hold for non-critical systems also. A significant shift was noted in effort and cost towards specification and design phases.

The authors noted that abstraction skills were an essential part of the modelling process and that the ease of constructing formal models should not seduce engineers away from devoting effort to selecting appropriate abstractions.

No major defects have been reported in the system developed using formal techniques. A mid-life upgrade was reported in 2008 [Wijbrans et al. 2008] and the development of the successor application will continue to use formal techniques. The developers argue that future work should focus on support for the specification and design phase, practical methods and tooling, and greater standardisation on a smaller set of methods taught in education.

4.7 Tokeneer

The Tokeneer ID Station (TIS) project [Barnes et al. 2006], carried out by Praxis in conjunction with SPRE Inc., under the direction of NSA (National Security Agency), has shown that it is possible to produce high quality, low defect systems conforming to the Common Criteria requirements of Evaluation Assurance Level 5 (EAL5) [CCRA 2006].

The Tokeneer system was originally developed by the NSA to investigate various aspects of biometrics in access control. The system consists of a secure enclave, physical access to which must be controlled. Within the secure enclave are a number of workstations whose users have security tokens (*e.g.*, smartcards), in order to gain access to the workstations. Users present their security tokens to a reader outside the enclave, which uses information on the token to carry out biometric tests (*e.g.*, fingerprint reading) of the user. If the user passes these tests, then the door to the enclave is opened and the user is allowed entry. The user also uses the security token to gain access to the workstations—at entry time, the Tokeneer system adds authorisation information to the security token describing exactly the sort of access allowed for this visit to the enclave, such as times of working, security clearance, and roles that can be taken on.

The Tokeneer ID Station (TIS) project redeveloped one component of the Tokeneer system. To facilitate the development, TIS device simulators implemented by the independent reliability consultants (SPRE Inc.) were used in place of actual TIS devices.

The core functionality of the system was written in SPARK, and the support software to interface it to simulated peripherals was written in full Ada. Tables II and III, taken from [Barnes et al. 2006], record the project's size, productivity, and effort. The project required 260 person-days of effort, comprising

	Size/source lines		***Productivity (LoC/day)***	
	Ada	SPARK annotations and comments	During coding	overall
TIS core	9,939	16,564	203	38
Support software	3,697	2,240	182	88

Table II. Tokeneer: size and productivity.

three part-time staff working over one year. Significantly, the number of defects

Project phase	*Effort %*	*Effort Person-days*
Project management	11	28.6
Requirements	10	26.0
System specification	12	31.2
Design Core functions	15	39.0
TIS Core code and proof	29	75.4
System test	4	10.4
Support software and integration	16	41.6
Acceptance	3	7.8
Total	***100***	***260.0***

Table III. Tokeneer: breakdown by project phase.

in the system, found during independent system reliability testing and since delivery in 2006, is *zero*. Barnes reports [Barnes et al. 2006] that the testing team from SPRE Inc. actually discovered two in-scope failures as part of their testing regime: both concerned missing items from the user manual, rather than errors in the TIS Core. The entry in Table III for System test does not include the testing contribution from SPRE Inc. Barnes estimates [Barnes et al. 2006] that a more representative figure might be 25%. The functional specification, written in Z and explanatory English, consists of about 100 pages.

The task set by NSA was to develop a system in conformance with the requirements in the Common Criteria for EAL5. In fact, Praxis exceeded the EAL5 requirements in a number of areas because it is actually more cost-effective to use some of the more rigorous techniques.

Praxis met the EAL5 criteria in the main body of the core development work, covering configuration control, fault management, and testing. They exceed EAL5, coming up to EAL6 or EAL7 levels, in the development areas covering the specification, design, implementation, and demonstration of correspondence between representations. Other aspects were out of scope, such as delivery and operational support.

The TIS project demonstrates that the Praxis Correctness by Construction development process is capable of producing a high-quality, low-defect system in a cost effective manner following a process that conforms to the Common Criteria EAL5 requirements.

4.8 The "Mobile FeliCa" IC Chip Firmware

"FeliCa" is a contactless IC card technology widely used in Japan, developed and promoted by Sony Corporation. Mobile telephones with FeliCa chips can serve as electronic purses, travel tickets, door keys etc. FeliCa Networks Inc. decided to use VDM++ and VDMTools [Fitzgerald et al. 2008] for the development of the firmware for a new generation IC chip containing new features but nevertheless operating to the strict timing requirements provided by earlier generations.

The project lasted three years and three months, and involved 50 to 60 people with an average age of a little over 30 years. No members had knowledge of or experience with the formal method at the time of project launch. VDM++ training (in Japanese) was provided for the development team by CSK. In addition an

external VDM consultant from CSK Systems was used throughout the project. The new version of the firmware was subsequently released in millions of IC chips [Kurita et al. 2008; Larsen and Fitzgerald 2007].

Following an iterative approach, specifications were developed initially in natural language augmented with informal diagrams based on UML notations. These then formed the basis of a formal model in executable VDM++ which was validated through test. The validated specification formed the basis of manual firmware and test case design and development before firmware test on development boards. More functionality was added on each iteration. In total some 30 iterations were carried out resulting in formal models and implementations of 86 firmware commands and the file system security specifications.

A large volume of VDM++ test cases was developed and then executed using the VDMTools interpreter. Using the VDMTools test coverage analysis facility, it was possible to display test coverage information on the VDM++ model after executing the entire test suite. Here 82% of the VDM++ model was covered, and the remaining parts of the model were manually inspected. In order to support this, the speed of the interpreter was improved by CSK Systems by more than a factor of 100.

The main outputs included a 383-page protocol manual written in natural language (Japanese), a 677-page external specification document written in VDM++ (approximately 100 kDSI[1] including comments, of which approximately 60kDSI are test cases formulated in VDM++). The implementation was approximately 110 kDSI of C/C++, including comments.

The team was divided into three groups: specification, firmware implementation and test. Questionnaires and interviews were used to assess the developers' impressions of the use of VDM++ after the project. None of the groups had negative reactions to using VDM++ but the implementers did not see the benefit as much as the others. VDM++ was felt to be an effective communication tool between team members and between teams. They all acknowledged that new knowledge was required but also that there is a substantial difference between the skills required in writing and reading a formal model. It was felt that advanced mathematics was not necessary for writing or validating the VDM++ model (although note that this development did not exploit proof).

Felica networks took the view that the application had been highly effective [Kurita et al. 2008]. From a quality perspective, more errors were found in the early phases of the development than in other similar projects at FeliCa Networks. In total 440 defects were detected in the requirements and the specifications. Of these, 278 were directly a result of the use of VDM++. Of these, 162 were found by review of the model whereas 116 were discovered using the VDMTools interpreter with test cases against the executable VDM++ model.

5. OBSERVATIONS

Taking into account new survey and the experience in the highlighted projects, we make the following observations.

[1]DSI = Delivered Source Instructions

Take-up has strengthened

There is significant take-up of formal techniques in some industries. Leading hardware developers continue to apply model checking and proof technology. In software, the exploitation of formal techniques has provided some evidence of the potential for applications focused on particular domains, including code verification. Application in the more traditional high integrity critical applications remains strong, and is largely performed by technical specialists.

Tools are more rugged

Our highlighted projects cover a long time period, and include projects as substantial as Mondex, in which proofs were constructed manually. The recent "revisiting" of Mondex by proponents of several formalisms have shown how far tool support has come in recent years. In the more recent highlighted projects, and in the new survey, there are examples of robust tools with support offered to a commercial standard. However, these have still only had limited take-up. In some of the free comments, there is a sense that tools are not usable by "mere mortals".

Verification technology is increasingly capable

Verification theory has reached a level at which substantial, previously intractable, problems can be tackled. In addition, both model checking and proof technology have benefited from Moore's Law and are now viable for commercial applications. The user interaction issues for proof mean that it is still treated as a specialist technology.

Formal methods are disappearing

Research is moving towards tools that may be used to provide value-added analyses "under the hood" of existing development environments. The SLAM and SDV experience suggests that a targeted approach can yield significant benefits on an exiting code base, when the tools are carefully integrated with existing development environments. The trend is also supported by the development of tools on open platforms, as plug-ins.

Integration with existing processes is significant

Model checking and test generation are significant areas of industry application. We could speculate that this is because they can be integrated into existing development processes with less upheaval than the adoption of a complete new design or programming framework.

Cost-benefit evidence is still incomplete

The surveys and highlighted projects suggest that there is anecdotal evidence for the claim that formal techniques can be used to derive relatively low defect density systems. However, the survey suggests that there is limited evidence available on the effect on development cost profile: around half of the contributions to our recent survey do not report the cost consequences of using formal techniques. The evidence from the highlighted project sis perhaps stronger. For example, the reduction in development times suggested by the transputer experience is notable. The Rockwell

Collins work makes an interesting point that the cost of a one-off formal methods project is significantly greater than the cost of repeated projects within a domain.

6. TOWARDS INDUSTRIAL ADOPTION

In spite of some significant successes, verification technology and formal methods have not seen widespread adoption as a routine part of systems development practice except, arguably, in the development of critical systems. The surveys reviewed in Section 2 suggest that there is a strong degree of consensus regarding the impediments to adoption, and some of these are borne out by our recent survey. In this section, we identify three research frontiers that are likely to contribute to enhanced adoption in future.

6.1 Deep and Broad Adoption

Our review suggests that it is not advisable to generalise about industry adoption. Verification technology can be applied in many domains; each domain, and indeed each potential adopter, has different requirements for analysis and verification. Different forms of adoption exist, and each has its own unique demands on the underlying theories, tools and methodology. Thus, for each formal methods adoption study, it is vital to have clear objectives for deployment of the methods and tools beyond the study itself. This subsequent deployment can take several forms: we will refer to "deep" and "broad" broad forms of adoption, bearing in mind that these are two extreme points on a spectrum.

Deep adoption refers to the use of theories and tools to gain high assurance in designs and implementations, as is typically the case for critical systems, where a key requirement is certification to standards requiring documented development processes, dependability cases and verified designs or code. Deep adopters are likely to use teams containing specialist practitioners who are able to construct formal models, formulate conjectures for verification, guide and interpret the results of semi-automated formal analyses.

Broad adoption refers to the widespread use of verification technology by large numbers of developers, not necessarily in critical applications. Drivers here are reductions in time to market, defect rates or costs of testing and maintenance. Practitioners may not be familiar with formalisms, and would not expect to interact directly with analysis tools. Models in a familiar design notation or code would be analysed using an underlying semantic framework, with automatic discharging of standard proof obligations, essentially as an enhanced form of static analysis in development environments that have been extended with verification technology "beneath the hood". Advances in automated analysis, in both model checking and proof support, have made technologies for broad adoption viable, and a lively area of research in recent years.

6.2 Tools: ruggedised tools, disappearing formal methods and plug-in architectures

Several studies listed in Section 2 and around a quarter of the projects surveyed in Section 3 identify the lack of commercially supported or "ruggedised" tools as an impediment to take-up of formal methods. In spite of the observation that tools are neither necessary nor sufficient for an effective formal methods application [Craigen et al. 1993a], it appears almost inconceivable that an industrial application would

now proceed without tools. The verification research community has focused a considerable effort on the development of new tools and to a limited but increasing extent, tools integration. We see advances in both areas as key contributors to future adoption.

An area of weakness pointed out over the last 20 years, and still remaining, is the quality of integration between specialist formal methods tools and the apparatus of design management. Support for cooperative working is rarely found in verification tools yet is essential on large-scale projects. Tools developers rarely seem to consider the issues of version control, yet this assumes considerable importance in the management of verification artefacts such as proofs and base theories.

Deep adoption requires the use of multiple modelling and analysis frameworks and their semantics in coherent, trusted, tool chains. Broad adoption typically requires a form of integration with existing development tools that may lack a formal basis. The requirement for automation means that analyses may not be as conclusive in identifying defects in models or code, but they can be applied at low cost.

Successful take-up involves "packing specific analyses into easier to use but more restricted components" [Bloomfield and Craigen 1999]. Verification analyses might become commonplace through integration with existing tools and techniques, and formal methods tools might even disappear from view [Rushby 2000], being invoked invisibly in conventional design environments, adding more powerful analyses to the range already available to conventional developers. Broad adoption entails having analyses that proceed behind the scenes automatically and return comprehensible results to the user [Arvind et al. 2008]. It seems likely that, for critical applications in which specific conjectures must be proven about developed code, skilled practitioners will have to interact directly with verification systems. However, interactions with formal analysis engines must be done at the level of the design language, not the formalism.

Successful integration requires that tools become decoupled components that can be integrated into existing tools for design, programming or static analysis. The integration of multiple verification approaches has been pioneered in development environments such as PROSPER [Dennis et al. 2003] and "plug-in" architectures such as Eclipse have been successfully applied for tools supporting Event-B [RODIN-Project-Members 2007] and VDM [Overture-Core-Team 2007].

Integrations between graphical design notations and the mathematical representations required for formal analysis are increasingly common. For example, the UML to B link [Snook and Butler 2006] allows use of a familiar modelling notation to be coupled to a formal analytic framework. Support for the verification of implementations of systems specified using control law diagrams has been addressed using Z, CSP and Circus [Cavalcanti et al. 2005].

6.3 Evidence to support adoption

An important first step towards adoption is to provide evidence on which to base a decision to deploy formal techniques after pilot studies. Yet in our recent review, around half of the projects examined did not report results allowing a cost-effectiveness calculation. Some of this may not be publicly accessible, but nonetheless, it suggests that pilot deployments are not being conducted with a view to

subsequent stages in technology adoption.

There have been many calls for improved evidence and cost models to support adoption arguments. However, it is worth bearing in mind that the decision to adopt development technology is a risk-based process and convincing evidence of the value of formal techniques in identifying defects (with as few false positives as possible) can be at least as powerful as a quantitative cost argument. We would argue for a strong body of evidence showing the utility of formal techniques and ease of use at least as strongly as we would call for the gathering of more evidence regarding development costs.

It is essential that pilot applications of formal technology are observable and that the observations are relevant to the decision-making process in the deploying organisation. Again, this requires a clear understanding of the forms of deployment to be sought at a later stage. For example, the FeliCa networks study focused on measuring queries against informal requirements documents that were attributable to formal modelling and analysis because this was seen as a significant feature in a development process where the object was to improve specifications. The Deploy project [Romanovsky 2008][2] explicitly addresses the industrial adoption of formal methods, addressing the movement from innovators to early adopters by stressing the importance of tools integrated into existing development processes.

7. THE VERIFIED SOFTWARE REPOSITORY

In 2003, Tony Hoare proposed the Verifying Compiler as a Grand Challenge for Computer Science [Hoare 2003]. As the proposal started to gain the support of the community, it became the Grand Challenge in Verified Software [Hoare and Misra 2008] and then the Verified Software Initiative, which was officially launched at the *2008 Conference on Verified Software: Theories, Tools, and Experiments* [Shankar and Woodcock 2008]. The UK effort is in the Grand Challenge in Dependable Systems Evolution, and current work includes building a Verified Software Repository [Bicarregui et al. 2006].

The Repository will eventually contain hundreds of programs and program modules, and amounting to several million lines of code. The code will be accompanied by full or partial specifications, designs, test cases, assertions, evolution histories, and other formal and informal documentation. Each program will have been mechanically checked by one or more of the VSI tools, and this is expected to be the major activity in the VSI. The eventual suite of verified programs will be selected by the research community as a realistic representative of the wide range of computer applications, including smart cards, embedded software, device routines, modules from a standard class library, an embedded operating system, a compiler for a useful language (possibly smartcard Java), parts of the verifier itself, a program generator, a communications protocol (possibly TCP/IP), a desk-top application, parts of a web service (perhaps Apache).

The notion of verification will include the entire spectrum, from avoidance of specific exceptions like buffer overflow, general structural integrity (or crash-proofing), continuity of service, security against intrusion, safety, partial functional correctness, and (at the highest level) total functional correctness [Hoare and Misra 2008].

[2]http://www.deploy-project.eu

Similarly, the notion of verification will include the entire spectrum, from unit testing to partial verification through bounded model checking to fully formal proof. To understand exactly what has been achieved, each claim for a specific level of correctness will be accompanied by a clear informal statement of the assumptions and limitations of the proof, and the contribution that it makes to system dependability. The progress of the project will be marked by raising the level of verification for each module in the repository. Since the ultimate goal of the project is scientific, the ultimate level achieved will always be higher than what the normal engineer and customer would accept.

In the following sections we describe five early pilot projects that are starting to populate the repository. The verified file-store in Sect. 7.1 is inspired by a space-flight application. FreeRTOS in Sect. 7.2 is a real-time scheduler that is very widely used in embedded systems. The bidding process for the Radio Spectrum Auctions described in Sect. 7.3 has been used in auctions with bids ranging from several thousands of dollars to several billions. The cardiac pacemaker in Sect. 7.4 is a real system, and is representative of an important class of medical devices. Finally, the hypervisor in Sect. 7.5 is provided by Microsoft and is based on one of their future products. The topics of two other pilot projects have been described above: Mondex, in Sect. 4.3, is a smart card for electronic finance; and Tokeneer, in Sect. 4.7, is a security application involving biometrics. These seven pilot projects encompass a wide variety of application areas and each poses some important challenges for verification.

7.1 Verified file store

Pnueli first suggested the verification of the Linux kernel as a pilot project. Joshi and Holzmann suggested a more modest aim: the verification of the implementation of a subset of the Posix file store interface suitable for flash-memory hardware with strict fault-tolerant requirements to be used by forthcoming NASA missions [Joshi and Holzmann 2007]. The space-flight application requires two important robustness requirements for fault-tolerance: (i) no corruption in the presence of unexpected power-loss; and (ii) recovery from faults specific to flash hardware (*e.g.*, bad blocks, read errors, bit corruption, wear-levelling, *etc*). In recovery from power loss in particular, the file system is required to be reset-reliable: if an operation is in progress when power is lost, then on reboot, the file system state will be as if the operation either has successfully completed or has never started.

The Posix file-system interface [Josey 2004] was chosen for four reasons: (i) it is a clean, well-defined, and standard interface that has been stable for many years; (ii) the data structures and algorithms required are well understood; (iii) although a small part of an operating system, it is complex enough in terms of reliability guarantees, such as unexpected power-loss, concurrent access, or data corruption; and (iv) modern information technology is massively dependent on reliable and secure information availability.

An initial subset of the Posix standard has been chosen for the pilot project. There is no support for: (i) file permissions; (ii) hard or symbolic-links; or (iii) entities other than files and directories (*e.g.*, pipes and sockets). Adding support for (i) is not difficult and may be done later, whereas support for (ii) and (iii) is more difficult and might be beyond the scope of the challenge. Existing flash-

memory file-systems, such as YAFFS2, do not support these features, since they are not usually needed for embedded systems.

Joshi and Holzmann are producing their own verified flash file system. The work has gone beyond being merely a demonstration of formal methods: it was adopted in 2006 by NASA's Multi-Mission Architectural Platform and will be used by the Mars Science Laboratory, with a launch date in 2009.

Woodcock, Freitas, and Butterfield have been constructing a tower of verified models from the POSIX interface all the way down to the flash memory hardware level. Their abstract specification of the POSIX interface is mechanised in Z/Eves [Freitas et al. 2007; Butterfield and Woodcock 2007; Freitas et al. 2007]. It is based on the following: (i) Morgan & Sufrin's 1984 UNIX filing store specification [Morgan and Sufrin 1984]; (ii) Patrick Place's specification of POSIX 1003.21 Real-time distributed systems communication [Place 1995]; and (iii) the IEEE POSIX Working Group interface requirements [IEEE POSIX Working Group 1995]. The abstract specification is refined to a concrete implementation of the directory structures specified in Z using a hashmap abstract data type lifted from JML annotations.

The abstract file maps are implemented as B^{+}-trees using a specification and refinement written by Liz Fielding and Cliff Jones in VDM in 1980 [Fielding 1980]. The specification, refinement, and proof of correctness of these trees uses an embedding of VDM in Z, which then allows the use of the Z/Eves theorem prover [Saaltink 1999; Meisels and Saaltink 1997; Saaltink 2003]. The embedding is a formal account of the differences between VDM and Z, including the difference in their logics (especially the use of three-valued logic in VDM and semi-classical logic in Z), the use of precondition-postcondition pairs in VDM and relations in Z, and the difference in the notion of types and their invariants.

The York-Dublin team have been specifying the interface to flash memory and verifying its consistency [Butterfield and Woodcock 2007]. They are working towards a mechanised model of flash memory containing the following:

—Pages, blocks, logical units, targets, devices.

—Memory addressing, defect marking.

—Mandatory command set: reset, read, write, protect, page program, change read/write column, block erase.

—Workload-related aging and wear-levelling algorithms.

—Memory reclamation (garbage collection).

For technological reasons, the management of flash memory requires some intricate algorithms and data structures. The memory cells in NAND flash memory can withstand only a limited number of program/erase cycles before they wear out, typically between 100,000 and 1,000,000 cycles. A naive approach would mean that the flash device would become unreliable as soon as the most-used memory cell became unreliable. To avoid this, wear-levelling algorithms are usually used to manage memory use. The key idea is to make the mapping between logical and physical blocks a dynamic object, where the physical location for a logical block changes on each update. These algorithms live in the layer immediately above the hardware adaptation layer, which contains the device driver, and the wear-levelling

algorithm may be in a special flash translation layer, or directly in the operating system itself. Work has started on using the Daikon invariant generator to infer potential assertions that can be used in proofs of correctness of device drivers and wear-levelling and memory reclamation programs.

Butler (Southampton) has been using Event-B and the Rodin tool-set to model hierarchical file systems in a rather different manner from that of [Morgan and Sufrin 1984]. The model is focused on basic functionalities affecting the tree structure, including create, copy, delete, and move. They aim at constructing a clear and accurate model with all proof obligations discharged. They start from an abstract model of a file system, and then add more details through refinement steps. Careful formulation of invariants and development of a reusable application-oriented theory make models simpler and easier to prove.

Jackson (MIT) has produced an Alloy model and analysis of a simple flash file system [Kang and Jackson 2008]. The current functionality of their file system is rather limited, but plans are in place to provide a larger set of Posix file operations and include support for directories. They also plan to model recovery mechanisms for other types of failures besides power loss, such as bad blocks and bit corruption. The model of flash memory abstracts from certain details in order to achieve simplicity. For example, the block map associating each virtual block to a physical sector is stored in the file system. In reality, the map is usually decomposed into two parts divided between RAM and flash memory. Future models will include this detail in to reflect a more realistic file system.

The Alloy model raises some interesting questions about the expressiveness of Alloy. Due to the declarative nature of the language, modelling imperative statements in Alloy is not always straightforward. For example, in order to model changes to the device after each call to fProgram in writeConc, they explicitly introduce a sequence of state atoms and impose a constraint between each pair of adjacent states. The resulting model can be cumbersome to write and difficult to understand. Thus, an imperative language may be more suitable for this particular aspect of the file system model. They are currently investigating an extension to Alloy that will provide the user with built-in imperative constructs, while maintaining the declarative power of the language.

7.2 FreeRTOS

Richard Barry (Wittenstein High Integrity Systems) proposed the correctness of their open source real-time mini-kernel as a pilot project. FreeRTOS is designed for real-time performance with limited resources, and is accessible, efficient, and popular. It runs on 17 different architectures and is very widely used in many applications. There are over 5,000 downloads per month from SourceForge, making it the repository's 250th most downloaded code (out of 170,000 codes). It is less than 2,500 lines of pointer-rich code. This makes it small, but very interesting.

FreeRTOS supports pre-emptive, co-operative, and hybrid configurations and has an extensive set of communication and synchronisation mechanisms: queues, binary semaphores, counting semaphores, recursive semaphores, mutexes, and interrupts.

7.3 Radio Spectrum Auctions

Robert Leese (Smith Institute) proposed the management of Radio Spectrum Auctions as a pilot project. The radio spectrum is an economically valuable resource, and OfCom, the independent regulator and competition authority for the UK communications industries, holds auctions to sell the rights to transmit over particular wavelengths. These auctions are often combinatorial, offering bundles of different wavelengths, which may then also be traded in secondary markets. The underlying auction theory is still being developed, but there are interesting computational problems to be overcome besides the economic ones.

The auction is conducted in two phases. The primary stage is a clock auction for price discovery; the supplementary stage is a sealed-bid auction. Both are implemented through a web interface. As well as the computational problems, there are challenges in getting a trustworthy infrastructure.

7.4 Cardiac pacemaker

Boston Scientific has released into the public domain the system specification for a previous generation pacemaker, and are offering it as a challenge problem. They have released a specification that defines functions and operating characteristics, identifies system environmental performance parameters, and characterises anticipated uses. This challenge has multiple dimensions and levels. Participants may choose to submit a complete version of the pacemaker software, designed to run on specified PIC hardware, they may choose to submit just a formal requirements documents, or anything in between.

McMaster University's Software Quality Research Laboratory is putting in place a certification framework to simulate the concept of licensing. This will enable the Challenge community to explore the concept of licensing evidence and the role of standards in the production of such software. Furthermore, it will provide a more objective basis for comparison between putative solutions to the Challenge.

An important characteristic of the challenge project is that it includes system-level requirements affecting hardware as well as software. Larsen and Fitzgerald have started work on the pacemaker by using a pragmatic incremental approach [Macedo et al. 2008]. They demonstrate how such cross-disciplinary requirements can be introduced gradually into a model in a phased fashion, along with validation of the functional and timing properties expressed in the model. They start from abstract system-level specifications of functional and timing properties, and progressively add detail via intermediate models that express system architecture, concurrency and timing behaviour. The models are expressed using VDM and are validated primarily by scenario-based tests, including the analysis of timed traces. The insight gained using this staged modelling approach is valuable in the subsequent development of implementations, and in detecting potential bottlenecks within suggested implementation architectures.

Of the 19 modes of the pacemaker, eight have been modelled so far, covering 18 of the 26 controlling variables. The study revealed that the regression test suite built from the validation activities on the intermediate models was valuable in validating the later, more complex models. The approach is very pragmatic, driven by the aim of providing a fully formal modelling approach with a low barrier to industrial

adoption. The relationship between the incremental addition of detail and formal refinement is still to be worked out, and must address the treatment of atomicity in the abstract and sequential models (for example in handling the maintenance of invariants). The intention is to automate most of the validation process.

7.5 Hypervisor

Thomas Santen proposed work on the Microsoft Hypervisor as a challenge project. The European Microsoft Innovation Center (EMIC) is collaborating with German academic partners and the Microsoft Research group for Programming Languages and Methods on the formal verification of the new Microsoft Hypervisor to be released as part of Windows Server 2008. The Hypervisor will allow multiple guest operating systems concurrently on a single hardware platform. By proving the mathematical correctness of the Hypervisor they will guarantee stability and security of the implementation.

8. CONCLUSIONS

We have to be able to relate to negative papers such as [Knight et al. 1997; Parnas 1998; Finney and Fenton 1996]

[Tiwari et al. 2003] introduces a methodology called *invisible formal methods*. It is described in the context of MathWorks' Simulink, which is used to model and design embedded controllers, but the lesson is a general one. The semantics of Stateflow is quite complex, and designs can benefit from formal analysis. Simulink is a *de facto* industry standard and so it is important that the formal analysis should be unobtrusive and acceptable to engineering practice. *Invisible formal methods* provides a spectrum of formal analysis that ranges from extended type-checking, through approximation and abstraction, to model checking and theorem proving. There is an increasing trend towards making formal methods invisible in many industrial settings.

There were heroic efforts to use formal methods 20 years ago when few tools were available. For example, in the 1980s IBM and Oxford University collaborated on the application of the Z notation to the CICS transaction processing system [Houston and King 1991]. This very successful project led to a Queen's Award for Technological Achievement, but it is significant that it used only very simple syntax and type-checkers. In the 1990s, The Mondex project, described in Sect. 4.3, was largely a paper-and-pencil exercise, but it still achieved the highest level of certification. Times have changed: today it would be inconceivable not to use some kind of verification tool.

In certain areas, there are collections of mature tools with broadly similar capabilities. For example, the Mondex experiment described in Sect. 4.3 shows very similar results from the application of state-based, refinement-oriented techniques and their tools. This suggests that there is scope for convergence and inter-operation between tools.

There is a resurgence of interest in the industrial application of formal methods, as shown by the recent emergence of the Verified Software Initiative. In most current applications of formal methods, we see tools and techniques that have been around for some time. Significant advances in verification technology have yet to filter through to more widespread use.

Many projects using formal methods demonstrate early phase benefits. But we were surprised that many respondents to our questionnaire didn't know the cost implications of their use of formal methods and verification. Project managers need to be aware of the need to measure costs.

Through the Verified Software Initiative, there are many experiments being undertaken in the industrial application of formal methods. A well-designed experiment should state clearly the hypothesis being tested and address the validity of the experiment as a means of testing the hypothesis. Not all experiments are set up this way, and we believe that there should be better appreciation of experimental method. One of the goals of the Verified Software Repository should be to set the standard for the well-designed experiment.

REFERENCES

Abrial, J.-R. 2007. Formal methods: theory becoming practice. *JUCS 13,* 5, 619–628.

Amey, P. 2002. Correctness by construction: Better can also be cheaper. *CrossTalk Magazine, The Journal of Defense Software Engineering 9,* 3 (March), 24–28.

Arvind, Dave, N., and Katelman, M. 2008. Getting Formal Verification into Design Flow. In *FM 2008: Formal Methods*, J. Cuellar, T. Maibaum, and K. Sere, Eds. Lecture Notes in Computer Science, vol. 5014. Springer-Verlag, 12–32.

Austin, S. and Parkin, G. 1993. Formal methods: A survey. Tech. rep., National Physical Laboratory, Teddington, Middelsex, UK. March.

Aydal, E. G., Paige, R. F., and Woodcock, J. 2007. Evaluation of OCL for large-scale modelling: A different view of the Mondex purse. In *MoDELS Workshops*, H. Giese, Ed. Lecture Notes in Computer Science, vol. 5002. Springer, 194–205.

Badeau, F. and Amelot, A. 2005. Using B as a high level programming language in an industrial project: Roissy val. In *ZB*, H. Treharne, S. King, M. C. Henson, and S. A. Schneider, Eds. Lecture Notes in Computer Science, vol. 3455. Springer, 334–354.

Barnes, J., Chapman, R., Johnson, R., Widmaier, J., Cooper, D., and Everett, B. 2006. Engineering the Tokeneer enclave protection system. In *Proceedings of the 1st International Symposium on Secure Software Engineering.* IEEE.

Barrett, G. 1987. Formal methods applied to a floating-point number system. Technical monograph.

Barrett, G. 1989. Formal methods applied to a floating-point number system. *IEEE Trans. Software Eng. 15,* 5, 611–621.

Barrett, G. 1990. Verifying the Transputer. In *Transputer Research and Applications, Proceedings of the 1st Conference of the North American Transputer Users Group*, G. S. Stiles, Ed. IOS, 17–24.

Behm, P., Benoit, P., Faivre, A., and Meynadier, J.-M. 1999. Météor: A successful application of B in a large project. In *World Congress on Formal Methods*, J. M. Wing, J. Woodcock, and J. Davies, Eds. Lecture Notes in Computer Science, vol. 1708. Springer, 369–387.

Bicarregui, J., Hoare, C. A. R., and Woodcock, J. C. P. 2006. The verified software repository: a step towards the verifying compiler. *Formal Asp. Comput. 18,* 2, 143–151.

Bloomfield, R. and Craigen, D. 1999. Formal methods diffusion: Past lessons and future prospects. Tech. Rep. D/167/6101, Adelard, Coborn House, 3 Coborn Road, London E3 2DA, UK. December.

Bowen, J. and Stavridou, V. 1993. Safety-critical systems, formal methods and standards. *Software Engineering Journal 8,* 4, 189–209.

Bowen, J. P. and Hinchey, M. G. 1995. Ten Commandments of Formal Methods. *IEEE Computer 28,* 4 (April), 56–62.

Bowen, J. P. and Hinchey, M. G. 2006. Ten Commandments of Formal Methods ... Ten Years Later. *IEEE Computer 39,* 1 (January), 40–48.

Butler, M. and Yadav, D. 2008. An incremental development of the Mondex system in Event-B. *Formal Asp. Comput. 20,* 1, 61–77.

Butterfield, A. and Woodcock, J. 2007. Formalising flash memory: First steps. In *12th International Conference on Engineering of Complex Computer Systems (ICECCS 2007), 10-14 July 2007, Auckland, New Zealand*. IEEE Computer Society, 251–260.

Cavalcanti, A., Clayton, P., and O'Halloran, C. 2005. Control law diagrams in circus. In *FM 2005: Formal Methods*, J. Fitzgerald, I. J. Hayes, and A. Tarlecki, Eds. Lecture Notes in Computer Science, vol. 3582. Springer-Verlag, 253–268. ISBN 978-3-540-27882-5.

CCRA. 2006. Common criteria for information technology security evaluation. part 1: Introduction and general model. Tech. Rep. CCMB-2006-09-001, Version 3.1, Revision 1, Common Criteria Recognition Agreement. September.

CESG. 1995. CESG computer security manual "F": A formal development method for high assurance systems. Tech. Rep. Issue 1.1, Communications-Electronics Security Group. July.

Clarke, E. M. and Wing, J. M. 1996. Formal methods: state of the art and future directions. *ACM Computing Surveys 28,* 4, 626–643.

Craigen, D., Gerhart, S., and Ralston, T. 1993a. *An International Survey of Industrial Applications of Formal Methods*. Vol. Volume 1 Purpose, Approach, Analysis and Conclusions. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA.

Craigen, D., Gerhart, S., and Ralston, T. 1993b. *An International Survey of Industrial Applications of Formal Methods*. Vol. Volume 2 Case Studies. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA.

Cuadrado, J. 1994. Teach formal methods. *Byte 19,* 12 (December), 292.

Dennis, L. A., Collins, G., Norrish, M., Boulton, R. J., Slind, K., and Melham, T. F. 2003. The PROSPER Toolkit. *International Journal on Software Tools for technology Transfer 4,* 2 (February), 189–210.

Fielding, E. 1980. *The Specification of Abstract Mappings and their Implementation as B+ trees*. Number PRG-18 in Technical Monograph. Oxford University Programming Research Group.

Finney, K. and Fenton, N. 1996. Evaluating the Effectiveness of Z: The Claims Made About CICS and Where We Go From Here. *Journal of Systems and Software 35*, 209–216.

Fitzgerald, J., Larsen, P. G., and Sahara, S. 2008. VDMTools: advances in support for formal modeling in VDM. *Sigplan Notices 43,* 2 (February), 3–11.

France-Massey, T. 2005. MULTOS—the high security smart card OS. Tech. rep., MAOSCO, Multos Consortium HQ, London. September.

Freitas, L., Fu, Z., and Woodcock, J. 2007. Posix file store in z/eves: an experiment in the verified software repository. In *12th International Conference on Engineering of Complex Computer Systems (ICECCS 2007), 10-14 July 2007, Auckland, New Zealand*. IEEE Computer Society, 3–14.

Freitas, L., Mokos, K., and Woodcock, J. 2007. Verifying the cics file control api with z/eves: An experiment in the verified software repository. In *12th International Conference on Engineering of Complex Computer Systems (ICECCS 2007), 10-14 July 2007, Auckland, New Zealand*. IEEE Computer Society, 290–298.

Freitas, L. and Woodcock, J. 2008. Mechanising Mondex with Z/Eves. *Formal Asp. Comput. 20,* 1, 117–139.

George, C. and Haxthausen, A. E. 2008. Specification, proof, and model checking of the Mondex electronic purse using RAISE. *Formal Asp. Comput. 20,* 1, 101–116.

Gibbons, J. 1993. Formal methods: Why should I care? the development of the T800 Transputer floating-point unit. In *Proceedings of the 13th New Zealand Computer Society Conference*, J. Hosking, Ed. 207–217.

Gibbs, W. W. 1994. Software's chronic crisis. *Scientific American*, 72–81.

Glass, R. L. 1996. Formal methods are a surrogate for a more serious software concern. *IEEE Computer 29,* 4 (April), 19.

Goldsmith, M. 2005. FDR2 user's manual. Tech. Rep. Version 2.82, Formal Systems (Europe) Ltd. June.

Goldsmith, M., Cox, A., and Barrett, G. 1987. An algebraic transformation system for occam programs. In *STACS*, F.-J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, Eds. Lecture Notes in Computer Science, vol. 247. Springer, 481.

Group, T. R. L. 1992. *The RAISE Specification Language*. The BCS Practitioners Series. Prentice-Hall.

Guiho, G. D. and Hennebert, C. 1990. SACEM software validation (experience report). In *ICSE: Proceedings 12th International Conference on Software Engineering*. 186–191.

Hall, A. 1990. Seven Myths of Formal Methods. *IEEE Software 7,* 5 (September), 11–19.

Hall, A. and Chapman, R. 2002. Correctness by construction: Developing a commercial secure system. *IEEE Software 19,* 1, 18–25.

Haneberg, D., Schellhorn, G., Grandy, H., and Reif, W. 2008. Verification of Mondex electronic purses with KIV: from transactions to a security protocol. *Formal Asp. Comput. 20,* 1, 41–59.

Hennebert, C. and Guiho, G. D. 1993. SACEM: A fault tolerant system for train speed control. In *FTCS*. 624–628.

Hinchey, M. G. and Bowen, J. P., Eds. 1995. *Applications of Formal Methods*. Prentice Hall. ISBN 0-13-366949-1.

Hinchey, M. G. and Bowen, J. P. 1996. To formalize or not to formalize? *IEEE Computer 29,* 4 (April), 18–19.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Communications of the ACM 12,* 10 (October), 576–581.

Hoare, C. A. R. 1985. *Communicating Sequential Processes*. Prentice Hall.

Hoare, C. A. R. 2003. The verifying compiler: A grand challenge for computing research. *J. ACM 50,* 1, 63–69.

Hoare, T. and Misra, J. 2008. Verified software: Theories, tools, and experiments: Vision of a grand challenge project. In *Verified Software: Theories, Tools, and Experiments. First IFIP TC2/EG2.3 Conference, Zurich, October 2005*, B. Meyer and J. Woodcock, Eds. Lecture Notes in Computer Science, vol. 4171. Springer, 1–18.

Houston, I. and King, S. 1991. Experiences and results from the use of z in ibm. In *VDM '91: Formal Software Development Methods*. Lecture Notes in Computer Science, vol. 551. Springer, 588–595.

IEC. 1997. Functional safety of electrical/electronic/programmable electronic safety-related systems. Tech. Rep. IEC 61508, International Electrotechnical Commission.

IEEE POSIX Working Group. 1995. Interface Requirements for Realtime Distributed Systems Communication. Tech. Rep. IEEE P1003.21, IEEE. Jul.

Institute of Electrical and Electronic Engineers 1985. *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985 ed. Institute of Electrical and Electronic Engineers.

ITSEC. 1991. Information technology security evaluation criteria (ITSEC): Preliminary harmonised criteria. Tech. Rep. Document COM(90) 314, Version 1.2. June.

Jackson, D. and Wing, J. 1996. Lightweight Formal Methods. *IEEE Computer 29,* 4 (April), 22–23.

Jones, C. 1996. A Rigorous Approach to Formal Methods. *IEEE Computer 29,* 4 (April), 20–21.

Jones, C. 2000. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley.

Jones, C. B., O'Hearn, P. W., and Woodcock, J. 2006. Verified software: A grand challenge. *IEEE Computer 39,* 4, 93–95.

Jones, G. and Goldsmith, M. 1988. *Programming in occam2*. Prentice Hall.

Josey, A., Ed. 2004. *The Single UNIX Specification Version 3*. Open Group. ISBN: 193162447X.

Joshi, R. and Holzmann, G. J. 2007. A mini challenge: Build a verifiable filesystem. *Formal Asp. Comput. 19,* 2, 269–272.

Kang, E. and Jackson, D. 2008. Formal modeling and analysis of a flash filesystem in alloy. In *ABZ2008: Abstract State Machines, B and Z, First International Conference, ABZ 2008,*

London, September 16–18, 2008, E. Börger, M. Butler, J. P. Bowen, and P. Boca, Eds. Lecture Notes in Computer Science, vol. 5238. Springer, 294–308.

Kars, P. 1997. The Application of PROMELA and SPIN in the BOS Project. In *The SPIN Verification System: The Second Workshop on the SPIN Verification System; proceedings of a DIMANS Workshop, August 5, 1996*. DIMANS Series in Discrete Mathematics and Theoretical Computer Science, vol. 33. 51–63.

Knight, J. C., Dejong, C. L., Gibble, M. S., and Nakano, L. G. 1997. Why are formal methods not used more widely. In *Fourth NASA Formal Methods Workshop*. 1–12.

Kuhlmann, M. and Gogolla, M. 2008. Modeling and validating Mondex scenarios described in UML and OCL with USE. *Formal Asp. Comput. 20,* 1, 79–100.

Kurita, T., Chiba, M., and Nakatsugawa, Y. 2008. Application of a Formal Specification Language in the Development of the "Mobile FeliCa" IC Chip Firmware for Embedding in Mobile Phone. In *FM 2008: Formal Methods*, J. Cuellar, T. Maibaum, and K. Sere, Eds. Lecture Notes in Computer Science. Springer-Verlag, 425–429.

Larsen, P. G. and Fitzgerald, J. 2007. Recent Industrial Applications of VDM in Japan. In *FACS 2007 Christmas Workshop: Formal Methods in Industry*, B. Boca and Larsen, Eds. BCS, eWIC.

Larsen, P. G., Fitzgerald, J., and Brookes, T. 1996. Applying Formal Specification in Industry. *IEEE Software 13,* 3 (May), 48–56.

Macedo, H. D., Larsen, P. G., and Fitzgerald, J. S. 2008. Incremental development of a distributed real-time model of a cardiac pacing system using vdm. In *FM 2008: Formal Methods, 15th International Symposium on Formal Methods, Turku, Finland, May 26-30, 2008, Proceedings*, J. Cuéllar, T. S. E. Maibaum, and K. Sere, Eds. Lecture Notes in Computer Science, vol. 5014. Springer, 181–197.

May, D., Barrett, G., and Shepherd, D. 1992. Designing chips that work. *Philosophical Transactions of the Royal Society A 339*, 3–19.

Meisels, I. and Saaltink, M. 1997. *Z/Eves 1.5 Reference Manual*. ORA Canada. TR-97-5493-03d.

Miller, S. and Srivas, M. 1995. Formal verification of the aamp5 microprocessor. In *Workshop on Industrial-Strength Formal Specification Techniques (WIFT95), April 5-8, Boca Raton, Florida*.

Miller, S. P. 1998. The industrial use of formal methods: Was darwin right? In *2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*. 74–82.

Miller, S. P., Greve, D. A., and Srivas, M. K. 1996. Formal verification of the aamp5 and aamp-fv microcode. In *Third AMAST Workshop on Real-Time Systems, Salt Lake City, Utah, March 6-8, 1996*.

Morgan, C. and Sufrin, B. 1984. Specification of the UNIX Filing System. In *Transactions on Software Engineering*. Vol. SE-10. IEEE, 128–142.

NASA. 1997. Formal Methods, Specification and Verification Guidebook for the Verification of Software and Computer Systems. Vol II: A Practitioner's Companion. Tech. Rep. NASA-GB-001-97, Washington, DC 20546, USA. May. Available from http://eis.jpl.nasa.gov/quality/Formal_Methods/.

NASA. 1998. Formal Methods, Specification and Verification Guidebook for the Verification of Software and Computer Systems. Vol I: Planning and Technology Insertion. Tech. Rep. NASA/TP-98-208193, Washington, DC 20546, USA. December. Available from http://eis.jpl.nasa.gov/quality/Formal_Methods/.

Overture-Core-Team. 2007. Overture Web site. http://www.overturetool.org.

Owre, S., Rushby, J. M., and Shankar, N. 1992. Pvs: A prototype verification system. In *CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, June 15–18 1992*. Lecture Notes in Computer Science, vol. 607. Springer, 748–752.

Parnas, D. 1998. "formal methods technology transfer will fail. *The Journal of Systems and Software 40,* 3 (March), 195–198.

Place, P. 1995. POSIX 1003.21—Real Time Distributed Systems Communication. Tech. rep., Software Engineering Institute @ Carnegie Mellon University. Aug.

RAISE Method Group, T. 1995. *The RAISE Development Method*. The BCS Practitioners Series. Prentice-Hall International.

Ramananandro, T. 2008. Mondex, an electronic purse: Specification and refinement checks with the Alloy model-finding method. *Formal Asp. Comput. 20,* 1, 21–39.

RODIN-Project-Members. 2007. RODIN Web site. http://rodin.cs.ncl.ac.uk/.

Romanovsky, A. 2008. DEPLOY: Industrial Deployment of Advanced System Engineering Methods for High Productivity and Dependability. *ERCIM News 74*, 54–55.

Roscoe, A. W. and Hoare, C. A. R. 1988. The laws of occam programming. *Theoretical Computer Science 60*, 177–229.

Rushby, J. 1993. Formal Methods and the Certification of Critical Systems. Tech. Rep. CSL-93-7, Computer Science Laboratory, Menlo Park CA 94025 USA. December.

Rushby, J. 2000. Disappearing formal methods. In *High Assurance Systems Engineering, 2000, Fifth IEEE International Symposium on. HASE 2000*. IEEE.

Saaltink, M. 1999. *Z/Eves 2.0 User's Guide*. ORA Canada. TR-99-5493-06a.

Saaltink, M. 2003. *Z/Eves 2.0 Mathematical Toolkit*. ORA Canada. TR-03-5493-05b.

Saiedian, H. 1996. An invitation to formal methods. *IEEE Computer 29,* 4 (April), 16–30. Roundtable with contributions from experts.

Shankar, N. and Woodcock, J., Eds. 2008. *Verified Software: Theories, Tools, Experiments, Second International Conference, VSTTE 2008, Toronto, Canada, October 6-9, 2008. Proceedings*. Lecture Notes in Computer Science, vol. 5295. Springer.

Shepherd, D. 1988. The role of occam in the design of the IMS T800. In *Communicating Process Architectures*. Prentice Hall.

Shepherd, D. and Wilson, G. 1989. Making chips that work. *New Scientist 1664*, 39–42.

Snook, C. and Butler, M. 2006. UML-B: Formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Methodol. 15,* 1, 92–122.

Snook, C. and Harrison, R. 2001. Practitioners Views on the Use of Formal Methods: An Industrial Survey by Structured Interview. *Information and Software Technology 43*, 275–283.

Spivey, J. M. 1989. *The Z Notation: a Reference Manual*. International Series in Computer Science. Prentice Hall.

Srivas, M. and Miller, S. 1995. Applying formal verification to a commercial microprocessor. In *IFIP Conference on Hardware Description Languages and their Applications (CHDL'95), Makuhari, Chiba, Japan*.

Stepney, S., Cooper, D., and Woodcock, J. 2000. An electronic purse: specification, refinement, and proof. Technical Monograph PRG-126, Oxford University Computing Laboratory. July.

inmos ltd. 1988a. *occam2 Reference Manual*. Prentice Hall.

inmos ltd. 1988b. *Transputer Reference Manual*. Prentice Hall.

Thomas, M. 1992. The industrial use of formal methods. *Microprocessors and Microsystems 17,* 1 (January), 31–36.

Tiwari, A., Shankar, N., and Rushby, J. 2003. Invisible formal methods for embedded control systems. *Proceedings of the IEEE 91,* 1 (jan), 29–39.

Tretmans, J., Wijbrans, K., and Chaudron, M. 2001. Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods. *Form. Methods Syst. Des. 19,* 2, 195–215.

Wijbrans, K., Buve, F., Rijkers, R., and Geurts, W. 2008. Software engineering with formal methods: Experiences with the development of a storm surge barrier control system. In *FM2008: Formal Methods*, J. Cuellar, T. Maibaum, and K. Sere, Eds. Vol. 5014. Springer-Verlag, 419–424.

Woodcock, J. 2006. First steps in the verified software grand challenge. *IEEE Computer 39,* 10, 57–64.

Woodcock, J. and Banach, R. 2007. The verification grand challenge. *J. UCS 13,* 5, 661–668.

Woodcock, J. and Davies, J. 1996. *Using Z: Specification, Refinement, and Proof*. International Series in Computer Science. Prentice Hall.

Woodcock, J., Stepney, S., Cooper, D., Clark, J. A., and Jacob, J. 2008. The certification of the Mondex electronic purse to ITSEC Level E6. *Formal Aspects of Computing 20,* 1, 5–19.

Received NN

Contents