# 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 describe some of the highlights of our survey by presenting a series of industrial projects, and 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

In this paper we survey current practice and experience in the application of formal verification technology with a particular emphasis on industrial deployment.

Corresponding Author's address: Jim Woodcock, Department of Computer Science, University of York, Heslington, York YO10 5DD, UK; email: `jim@cs.york.ac.uk`.
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2009 ACM 0000-0000/2009/0000-0001 $5.00

We focus on the lessons learned from industrial applications of formal methods in software development over the last 20 years, identifying areas in which progress has been substantial and new challenges.

Formal methods are techniques for the development of computer-based systems, hardware and software. They are distinguished from traditional development methods by their use of notations with a highly rigorous, mathematical semantics to express design artefacts, whether abstract designs or implementation-level code. We use the term *verification* to refer to the process of confirming that a design artefact expressed in a formal notation has a specified property. The use of a formal notation means that verification can be performed symbolically, to a higher degree of assurance than is normally obtained from testing alone. Further, the availability of a formal semantics means that analyses can be carried out, in part, automatically.

There are many diverse formal methods and verification systems. It may be helpful to define this space along two dimensions: the analytic power of verification tools, and the expressive power of the formal languages in which specifications are expressed.

Considering the first dimension, there is a wide range of verification techniques supported by tools. There tends to be a trade-off between the level of automation achievable in verification and the complexity of the analysis performed. At one end of the range there are static analysers that identify potential program defects such as variable use before definition, constant conditions, and possibly out-of-range expressions. Examples include heuristic tools such as the 30-year-old Unix utility `lint` [Johnson 1978], which flags potential defects in C programs and more recent tools such as Splint [Evans and Larochelle 2002], which provides annotation-assisted lightweight static checking. These program analysis tools are in use in industry; see [Larus et al. 2004] for a short survey of their use within Microsoft. Modern analysers such as PREfix [Bush et al. 2000] and PREfast can analyse multi-million-line C and C++ programs for potential null pointer references, improper memory allocation and deallocation, uninitialised variable use, simple resource-state errors, improper library use and problematic programming idioms. Such tools often use formally unsound analytical techniques and so may produce false positives (identifying spurious errors), although efforts are made to minimise these. The SLAM tool on the other hand can certify a program free from a particular kind of error [Ball and Rajamani 2002] by checking that API usage follows certain sequencing rules given by the user. It has been used to find many bugs in Windows device drivers, and is distributed as the Static Driver Verifier, part of the Windows development kit. The ESP tool [Das et al. 2002] is similar to SLAM, but trades precision against large-scale tractability for analysing large code bases; it has been used to find buffer over-runs in large-scale system code and to validate an OS kernel's security properties.

Further along this verification spectrum there are tools such as model checkers and abstract interpreters. The SPIN automata-based model checker has been used to find logical design errors in distributed systems design, such as those used on the Deep Space 1, Cassini, Mars Exploration Rovers, and Deep Impact space missions [Holzmann 2004]. It checks the logical consistency of a specification, reporting on deadlocks, unspecified receptions, incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes. An example of an abstract interpreter is the ASTRÉE real-time embedded-software static analyser [Blanchet et al. 2003], which tries to prove the absence of all run-time errors in a C program. It has done this completely automatically for the primary flight-control software for the Airbus A340 and the electric flight control codes for the A380.

At the furthest end of the verification spectrum, theorem provers can verify arbitrary conjectures in a given logic with varying degrees of automation. For example, Vampire is an automatic theorem prover for first-order classical logic [Riazanov and Voronkov 2002]. KIV, on the other hand, is an interactive theorem prover with a user-definable object logic [Balser et al. 2000], and it has been used in a wide range of applications, from verifying protocols for medical procedures [Hommersom et al. 2007] to verifying protocols for smart cards [Haneberg et al. 2008].

Specification notations vary in their expressiveness. At one end of this scale we find a light sprinkling of special-purpose assertions, as with ESP [Das et al. 2002], mentioned above. More systematic specifications may be given for internal interfaces; for example, the Eiffel programming language [Meyer 1992] is based on the concept of design by contract. Routines are given preconditions and postconditions and classes are given invariants, and these contracts can be used to guide feature redefinition in inheritance, so that preconditions can be weakened and postconditions strengthened moving down the inheritance chain. Some Eiffel programmers find it useful to work with rather weak contracts, with well-documented preconditions and trivial postconditions. Eiffel has been used in many industrial projects, including applications in banking and finance, defence and aerospace, health care, networking and telecommunications, computer-aided design, and game programming.

Further along this scale there are special-purpose languages with automatic program generators. For example, SCADE [Esterel Technologies 2003] fills the semantic gap between control engineering models, such as those used with Simulink [MathWorks ], and the software implementation of mission and safety-critical applications. It is a commercial product produced by Esterel Technologies, and it is based on the synchronous data-flow programming language, LUSTRE [Caspi et al. 1987]. It can generate production quality C or Ada code, and it has been qualified as a development tool for DO-178B up to Level A. It is used most in aerospace and defence applications, although Esterel report many applications in the transportation, energy, and heavy equipment domains. Finally, there are languages for the full functional specification of a system. These include: VDM [Jones 1990], Z [Spivey 1989], Alloy [Jackson 2002], ASM [Börger and Stärk 2003], and B [Abrial 1996], all of which have strong records of industry application.

The points in the plane mapped out by our two dimensions represent different couplings of specification systems with verification tools. For example, the Alloy formalism is closely linked with a model-checker; Z has been used with the Eves proof system and VDM has been used with both the HOL and PVS provers. It can be highly advantageous to move about this plane, even within a single project. For example, the recent application of the RAISE specification language to the Mondex challenge [George and Haxthausen 2008] used the SAL model checker to identify a proportion of invalid conjectures before attempts at proof are made using the

PVS proof system. As discussed in later sections, advances in the pragmatics of formal methods tools, such as the use of plug-in architectures, is enabling a greater freedom of movement in this plane.

In this paper we assess the state of the art in formal methods application, and we appraise the extent of industrial use of this technology. We first revisit several influential surveys of the use of formal methods and verification technology in industry (Sect. 2). We present the results of a new survey of industrial practice in Sect. 3. In Sect. 4, we describe selected industrial projects from the last 20 years, representing 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, based on the survey findings and highlighted projects. A weakness in the current situation is lack of a substantial body of technical and cost/benefit evidence from applications of formal methods and verification technology; in Sect. 6, we describe the Verified Software Repository that is being built in response to this challenge. Finally, in Sect. 7, we draw some conclusions about the current practice and experience. A list of acronyms is provided in Sect. 7.

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

### 2.1 Surveys of formal methods practice

The transfer of verification and formal methods technology into industry 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. A majority of responses analysed (126) reported on the use of model-oriented formalisms and concentrated on specification 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 which was claimed to be "an industrially useful scale".

In spite of the optimistic conclusions of some surveys, a round-table article in IEEE Software in 1996 [Saiedian 1996] showed the divergence of opinion on whether formal methods were delivering hoped-for improvements in practice. 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 formal methods adoption. 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. In spite of the differences in approach, there is some agreement on significant challenges to successful industrial adoption. We examine several such challenges below.

## 2.2 Deployment Strategies: Lightweight or Heavyweight Formal Methods?

Formal methods, while believed by their more experienced users to be within the capabilities of typical engineers [Snook and Harrison 2001], seem to present a significant barrier to potential adopters because of their rigour and dissimilarity to more conventional approaches. 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.

## 2.3 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 industrialisation 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 tools are not always accompanied by advances in methodology or theory.

### 2.4 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.

### 2.5 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 help gain further understanding of trends and advances against the challenges identified in the papers described above, we undertook a survey to gather information in a consistent format on a number of industrial projects known 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 that can be used in an industrial context. The data collection was targeted towards gathering information regarding the current state and trends in formal methods uptake. We were interested in gaining a better understanding of the answers to the following questions:

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

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

—Where in the development process are they being used?

—What is the trend over time of the application of formal methods?

The questions we asked concerned:

—The type of software developed and the application domain.

—The timescale and size of the project undertaken.

—The tools and techniques used for different development activities.

—The experience and training provided to the personnel involved.

—The satisfaction with the adequacy of the techniques used.

### 3.2 Data Collection

Using a structured questionnaire, data was collected on 54 industrial projects known from the published literature, mailing lists and personal experience, to have employed formal techniques. Whilst recognising that this was a highly biased sample, both in the way the projects were selected, and in that its respondents would be subject to acquiescence and social biases, the uniform way in which the data was collected did allow comparisons to be made between the projects in the selected group.

Data was collected over 1 year between November 2007 and October 2008. 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, separate questionnaires were completed for each project. In 48 of the 54 projects, data was collected directly from individuals who had been involved those projects and in the remaining 6 cases questionnaires were completed by the present authors from information available in the literature.

*The application.* Data was gathered on the application domain and type of application to which 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 the corresponding life-cycle activities.

*Personnel.* Data was collected on the composition and experience of the project team. Respondents were asked to state the roles that 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 of formal techniques on the outcome of the project in each of these dimensions, and 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 group of project studied it is impossible to generalise the conclusions too widely, we still consider it valuable to summarise the responses as this gives some insight into current practice and long term trends in the use of formal methods.

*The Application.* Figure 1 presents the application areas to which the projects related. The largest single application domain was transport (26%) followed by financial (22%). Other major sectors were defence (14%), telecommunications (10%), and office and administration (10%). Other areas with two or fewer 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

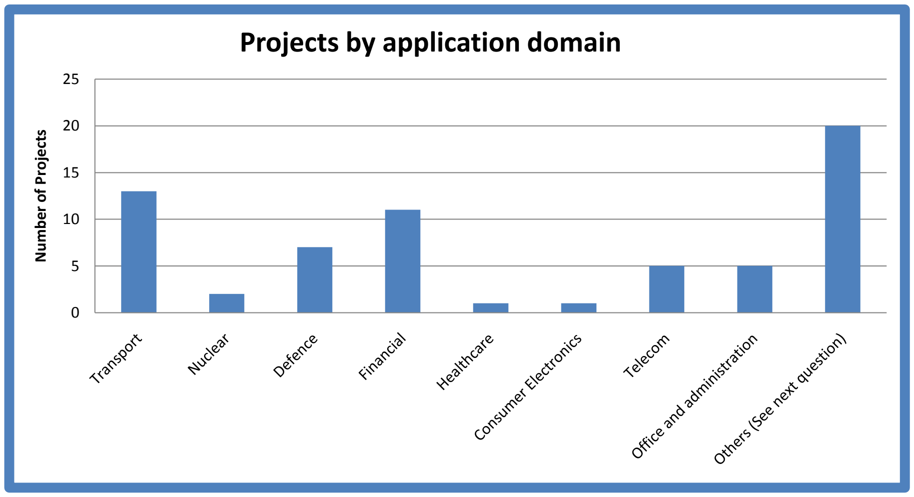

Fig. 1. Application Domains

applications within the domain such as high performance computing, runtime code optimisation, file system replication, access control, communications protocols, microcomputer design.

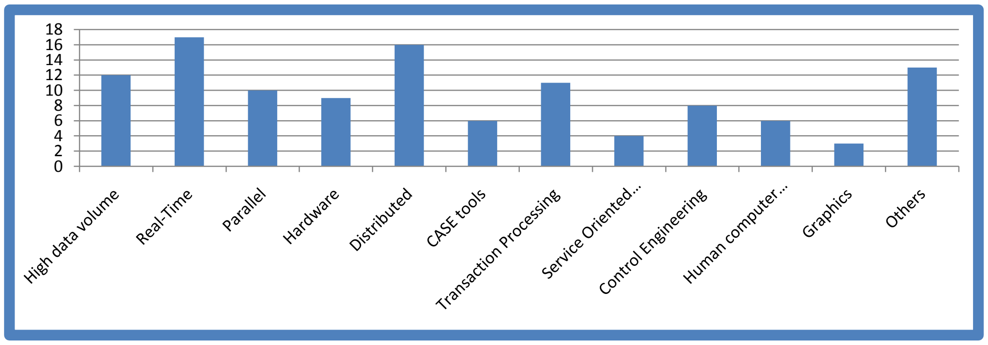

Fig. 2. Application Types

The types of applications are presented in Figure 2. The largest groups of application types were real-Time applications (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 (8%).

Certification standards were indicated as 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. After taking account of the fact that ongoing projects are unlikely to be represented in the survey, it would seem that 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 readily obtained.

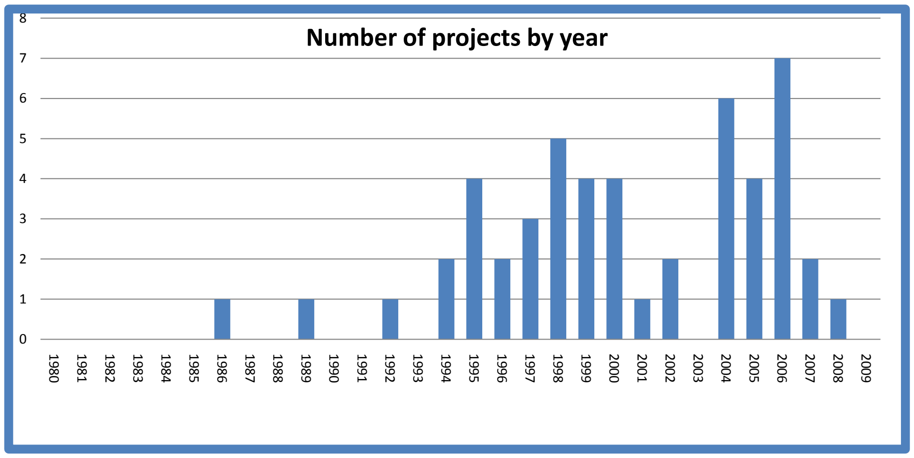

Fig. 3. Project start date

*Software size.* As shown in Figure 4, 56% of respondents gave an 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%).

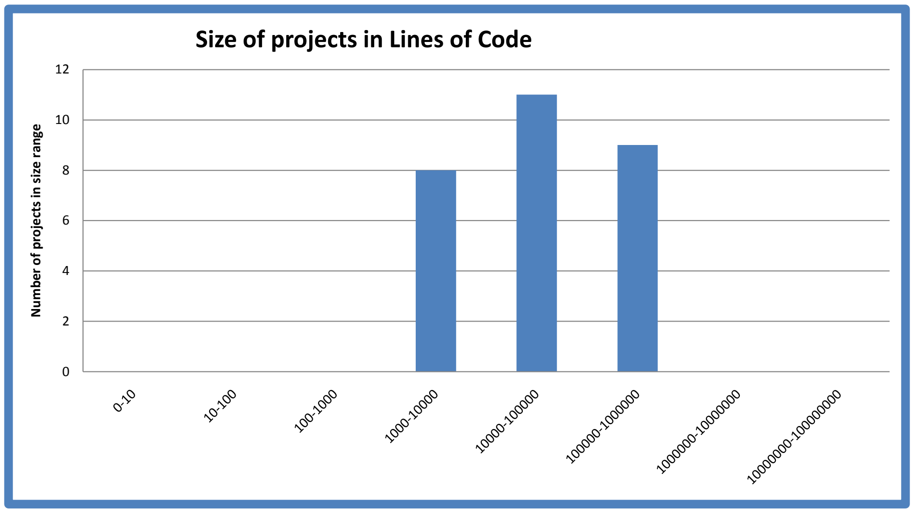

Fig. 4. Project size in Lines of Code

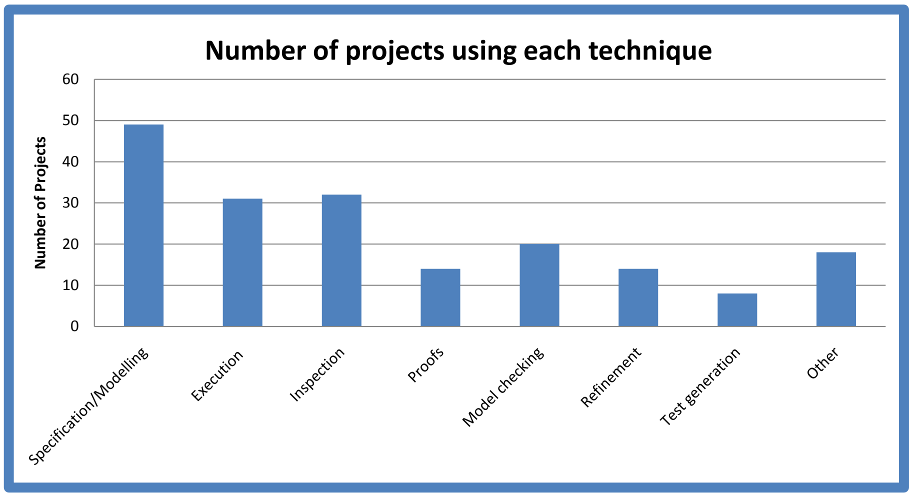

Fig. 5. Techniques used

*Methodology.* Figure 5 presents the rates of use of various techniques such as specification/modelling, execution (specification interpretation/testing), inspection (of specification/model).

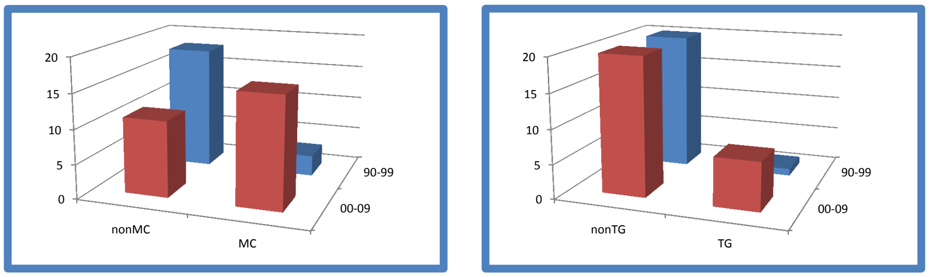

Fig. 6. Change in use of Model Checking and Test Generation by decade

The use of these techniques was correlated against the date of the project (Figure 6 and it was found that a greater proportion of reported applications had used model checking in this decade compared to the older projects. Model Checking has increased from 14% (3 out of 21) in the 1990s to 59% (16 out of 27) in this decade. This is a highly significant change (Fishers exact test, double tailed, p=0.003). The use of Proof has decreased slightly in this period from 33% to 22% and the use of refinement 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 moderately from 5% to 26% (p= 0.064).

### 3.4 Techniques used for different Application Domain and Software types

We looked for dependencies between application domain and type of software developed and found only one significant correlation which was a high representation of transaction processing software in the financial domain (Fishers exact test, double tailed, p=0.02). However, without comparison to general data relating software type and application domain, this can only be assumed to reflect a trend in software as a whole, rather than being related to the application of formal techniques.

Similarly, no significant dependencies at all were found between the techniques used for different application domains and only a few, reasonably mild, correlations were observed between the techniques used for different type of software. These correlations were a higher than expected use of model checking in consumer electronics and higher than expected use of execution and inspection in transaction processing software (Fisher Exact test, double tailed, p=0.04 for each).

The fact that there were so few significant correlations may itself seem slightly surprising as it would seem to imply a rather uniform penetration of formal techniques into various potential software markets and may indeed indicate a general lack of focus in the targeting of the use of formal techniques.

*Personnel.* Interestingly, when asked to indicate which roles were part of the project team from a predefined 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 expertise in the techniques used, 50% reported "Considerable previous experience", 48% reported "Some previous experience" and 16% reported "No previous expertise". The total adds up to more than 100% because some respondents reported 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 these techniques.

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

*Outcomes—the effect on Time, Cost and Quality.* Figure 7 shows the responses concerning the overall effect of the use of formal techniques on time, cost, and quality.

*Time.* The effect of the use of formal techniques on time taken to do the work was on average beneficial. Of those cases where a view was expressed (40%), three times as many reported a reduction in time (30%) than an increase (10%). Many responses (60%) indicated that it was difficult to judge the effect on time taken, although several noted increased time in the 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"

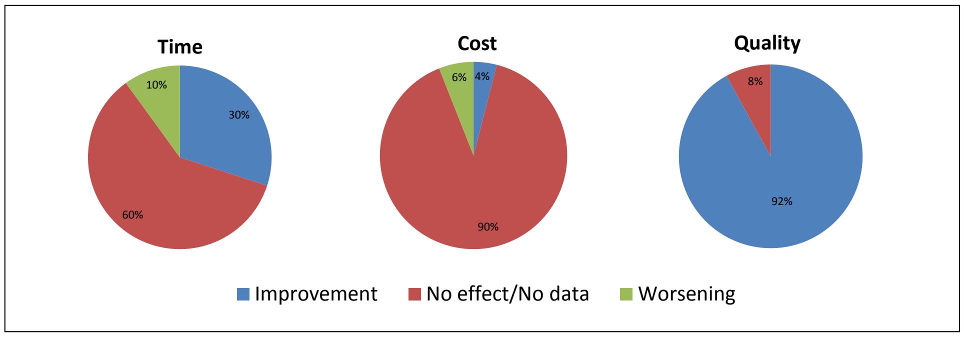

Fig. 7. Did the use of formal techniques have an effect on time, cost and quality?

> "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 by respondents 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 most 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%).

*Respondents' Conclusions.* As shown in Figure 8 the survey respondents have generally been positive about the success with the use of formal methods in the projects that have been reported, although, due to the bias in the selection process described above, one would expect stakeholders that did not see value in the use of formal methods to be under-represented among the responses. Figure 9 illustrates that the respondents in general were satisfied with the formal techniques used in their projects, whereas figure 10 shows that in 10% of the projects, the tools applied have not fully been able to live up to expectations. Finally, Figure 11 also demonstrates that a majority of the respondents wish to use a similar technology again on new projects.

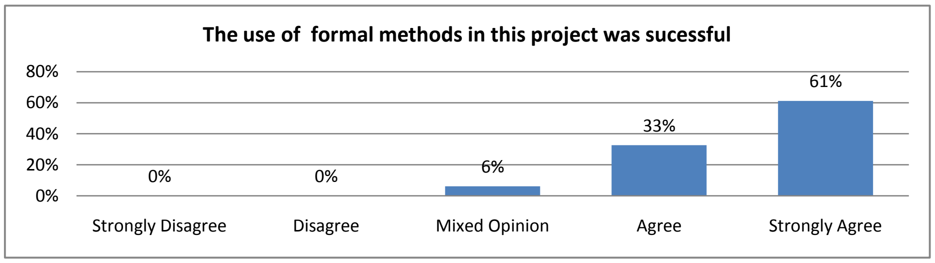

Fig. 8. Overall satisfaction with the used of formal techniques

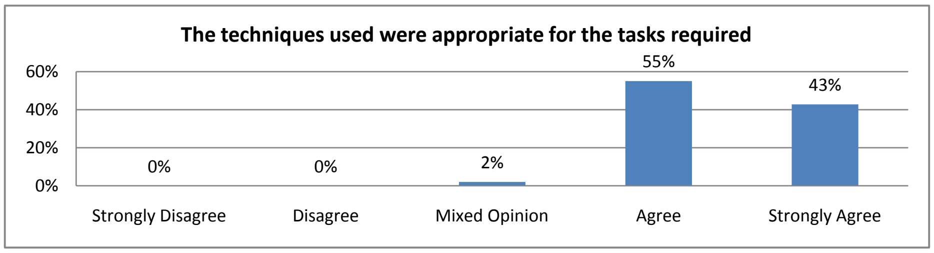

Fig. 9. Overall satisfaction with the techniques used

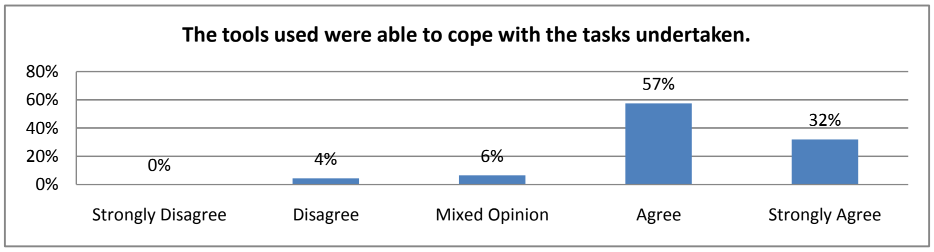

Fig. 10. Overall satisfaction with the tools used

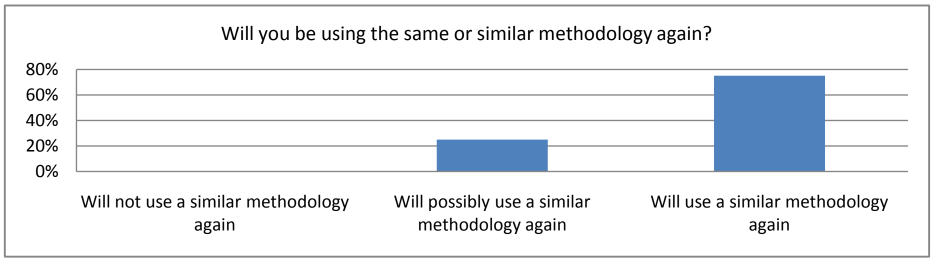

Fig. 11. Intention to use formal techniques again

## 3.5 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 all of the application domains listed in the questionaire. Similarly, there are examples of the 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 that 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 market 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 characterised by extensive and comprehensive analysis, significant manual intervention in the formal analysis, and therefore a major change to the informal techniques. On the other hand, some projects were more interested in "broad" analysis, hoping for some benefit from little or no manual effort.

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 that 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 formal methods disappearing "under the hood" in due course. Such a trend is epitomised by the following quotation from a project that employed significant levels of model checking:

> "To be useful during product development, formal methods 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 inarguably replacing specialist expertise 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 follows:

> "Tools for formal methods are 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, whether relative or absolute, of employing formal techniques. On the other hand, the overall satisfaction with their use amongst the respondents is clear (Figures 8 and 11).

> "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 an improvement in value for money. Arguably, this approach to decision making may be being taken because the estimation and assessment of the 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. In this setting, automated analyses that offer "something for nothing" are perceived as useful even if techniques that require a more substantial investment cannot be justified. The cost-benefit of the use of particular software development techniques, whether they be formal or informal, is clearly an area which could benefit from more quantitative research.

## 4. HIGHLIGHTED PROJECTS

We present a series of industrial formal methods projects, chosen to show a cross-section of work 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. Each has been included in the survey described in the last section.

### 4.1 The transputer project

The transputer series of microprocessor chips were designed specifically for parallel processing [Inmos Ltd 1988b]. Gibbons describes the development of one of the transputers: the T800 floating-point unit [Gibbons 1993], which combined a 32-bit reduced instruction set CPU, some memory, four bidirectional communications links on a single chip, and a floating-point arithmetic unit. Its successor, the T9000, was rather more sophisticated, with richer connectivity, memory model, and pipelined processor. A transputer based on T9000 technology, the ST20, 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].

Inmos started to develop the T800 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 natural language IEEE-754 standard for floating-point arithmetic [IEEE 1985] was formalised in Z [Spivey 1989; Woodcock and Davies 1996]. The specification is described in [Barrett 1987; 1989], and revealed some problems in the standard. For example, the standard requires that diagnostic information about the results of invalid operations (such as the square root of a negative number) be propagated through further operations. But this is not always possible. The next task was to show that a floating-point package written in occam and used in previous transputers was a correct implementation of IEEE-754. The attempted verification using Hoare logic [Hoare 1969] 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 and used 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 faster than the alternative informal development that ran concurrently [Barrett 1990]. 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 the translation program that converted the micro-level occam into assembly code for the chip. The second was introduced by a hand optimisation of this assembly code. Oxford University and inmos jointly received a Queen's Award for Technological Achievement in 1990 "to recognise and encourage outstanding achievements advancing process or product technology in the UK".

### 4.2 Railway signalling and train control

In 1988, GEC Alsthom, MATRA Transport, and RATP (the Parisian public transport operator) started working on a computerised signalling system for controlling the RER regional express network commuter trains in Paris (reported in [Bowen and Stavridou 1993]). The objective of the project was to increase network traffic by 25%, while preserving existing safety levels. The resulting SACEM system with embedded hardware and software was delivered in 1989 and has controlled the speed of all trains on RER Line A in Paris, involving seven billion passenger journeys, 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 B [Abrial 1996] and the proofs were done interactively using automatically generated verification conditions for the code. The validation effort

| | **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.

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 (the creator of B) reports on two further railway projects carried out using B [Abrial 2007]: Line 14 of the Paris Métro, a system in use since October 1998 [Behm et al. 1999]; and the driverless Paris Roissy Airport shuttle, in use since 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. No unit tests were performed for the Line 14 or Roissy Shuttle projects; they were replaced by some global tests that were all successful. This reduced the overall costs significantly.

## 4.3 Mondex

In the early 1990s, the National Westminster Bank and *platform seven* developed a smartcard-based electronic cash system, Mondex, suitable for low-value cash-like transactions, with no third-party involvement, and no cost per transaction. 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]. It was crucial that the card was secure, otherwise money could be electronically counterfeited, so *platform seven* decided to certify Mondex to one of the very highest standards available at the time: ITSEC Level E6 [ITSEC 1991], which approximates to Common Criteria Level 7 [CCRA 2006]. 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 and the lower-level concrete architectural design, and to provide a formal proof of correspondence between the two, in order to show that the concrete design obeys the abstract security properties. The evaluation was carried out by the Logica Commercial Licenced Evaluation Facility, with key parts of the work subcontracted to the University of York to ensure independence.

The target platform smartcard 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. Power could be withdrawn at any point during the processing of a transaction. Logica was contracted to deliver the specification and proof using Z [Spivey 1989; Woodcock and Davies 1996]. They had 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 would have made the abstraction much easier, but was thought to be too expensive to implement, as the parallel implementation work was already well beyond that point. The 200-page proof was carried out by hand, and revealed 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. The extensive proofs that were carried out were done manually, a decision taken at the time to keep costs under control. Recent work (reported below) 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 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. No errors were found in any part of the system subjected to the use of formal methods.

Mondex was revived in 2006 as a pilot project for the Grand Challenge in Verified Software (described further in Sect. 6). 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 using the following formal methods: Alloy [Ramananandro 2008], ASM [Haneberg et al. 2008], Event-B [Butler and Yadav 2008], OCL [Kuhlmann and Gogolla 2008], PerfectDeveloper, $\pi$-calculus [Jones and Pierce 2007], Raise [George and Haxthausen 2008], Z [Freitas and Woodcock 2008]. The main results showed that mechanising the proof of correctness of Mondex was clearly within the state of the art in 2006. The cost of mechanising the Z proofs of the original project was 10% of the original development cost, and so did not dominate costs as initially believed. Interestingly, almost all techniques used in the Mondex pilot achieved the same level of automation, producing similar numbers of verification conditions and requiring similar effort.

## 4.4 Rockwell Collins

Miller reports experiences from several formal methods projects carried out at Rockwell Collins [Miller 1998]. One of their earliest experiments was to specify and refine a micro Real Time Executive $\mu$RTE in the RAISE notation, RSL [RAISE Language

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 available were syntax and type checkers, with no support for developing and managing 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 complex processing units, such as a Lookahead Fetch Unit. It contains approximately 500,000 transistors and provides performance somewhere between an Intel 386 and 486. Rockwell Collins 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 for 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 application-oriented 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.

The latest project involves the AAMP7 processor, which has a separation kernel built into the micro-architecture. Rockwell carried out a proof in ACL2 that a line-by-line model of the microcode adheres to a security policy about how partitions are allowed to communicate [Wilding et al. 2001; Greve and Wilding 2002]. What is hard about proofs of this kind is the complexity of dealing with pointer-laden data structures. The work received National Security Agency certification as a Multiple Independent Levels of Security device for use in cryptographic applications, at EAL-7 of the Common Criteria.

## 4.5 SCADE

Airbus have used SCADE for the last ten years for the development of DO-178B Level A controllers for the A340-500/600 series, including the Flight Control Secondary Computer and the Electric Load Management Unit. A summary of these and other industrial applications is described in [Berry 2008]. Esterel reports the following benefits: (i) A significant decrease in coding errors: for the Airbus A340 project, 70% of the code was generated automatically. (ii) Shorter requirements changes: the SCADE tool suite manages the evolution of a system model as requirements change, and in the Airbus A340 project, requirements changes were managed more quickly than before, with improved traceability. (iii) Major productivity improvement: Airbus reported major productivity gains, in spite of the fact that each new Airbus programme requires twice as much software as its predecessor.

Having achieved significant savings on the overall design cycle, Airbus adopted SCADE for A380 projects, where most on-board computers developed by Airbus and its suppliers benefit from SCADE technology. The SCADE Suite is used for the development of most of the A380 and A400M critical on-board software, and for the secondary flying command system of the A340-500/600 aircraft, in operational use since August 2002. The A380 and A400M Cockpit Control and Display System and the On-board Airport Navigation System Display have been developed using SCADE Display, oriented towards the specification of graphical interfaces.

## 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 computing system. In terms of the international standard IEC 61508 [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 [Spivey 1989; Woodcock and Davies 1996], and this was embedded into a Promela model describing control, and designs were validated using the SPIN model checker [Holzmann 2004]. 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 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 200 KLOC for the operational system and 250 KLOC 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 than 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.

### 4.7 Tokeneer

The Tokeneer ID Station (TIS) project [Barnes et al. 2006], carried out by Praxis High Integrity Systems 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 re-developed 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, a subset of Ada with an accompanying toolset, which was specifically designed for writing software for high integrity applications [Barnes 1997]. 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

| | **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.

| **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.

project required 260 person-days of effort, comprising three part-time staff working over one year. The number of defects found in the system during independent system reliability testing and since delivery in 2006 is two, both discovered by code inspection after the completion of the project (see Spinellis's blog for an account of the second bug [Spinellis 2008]). 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 correspondence between representations. Other aspects were out of scope, such as delivery and operational support.

### 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].

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.

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 KLOC including comments, of which approximately 60 KLOC are test cases formulated in VDM++). The implementation was approximately 110 KLOC of C/C++, including comments.

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

Previous surveys and our recent review indicate that there have been successes in the application of verification and formal methods to problems of industrial scale and significance, and within industrial settings. 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.

In spite of their 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 in certain domains. Indeed, we may expect diffusion of rigorous verification and design technology to take place gradually and not necessarily to result in explicit adoption of "formal methods" as a distinct technology.

In this section we revisit concerns identified in previous surveys (Sect. 2) and identify progress, trends and remaining challenges in the light of more recent projects.

## 5.1 Successful applications are "lightweight", carefully focused and tools-based

Our review does not suggest that there is a single strategy for the successful application of verification and formal methods technology. However, a "lightweight" approach, as described in Sect. 2.2, appears to underpin many recent industry applications. Fully formal techniques are used, but are typically focused on specific subsystems and on the verification of particular properties. The success of this approach depends on, and is limited by, the quality of tool support.

he significant developments in tools and tool chains might be expected to affect the forms of deployment of formal techniques in future. Where the achievement of certification is a significant driver, specialist practitioners may be used to construct formal models, formulate conjectures for verification, guide and interpret the results of semi-automated formal analyses. However, the increasing capability of automated formal analysis makes it possible to have an impact on larger-scale developments, not necessarily critical, that are driven by the need to reduce time to market, defect rates or costs of testing and maintenance. In such projects, we might expect to see widespread use of enhanced static analysis tools incorporating model checking or proof capabilities.

## 5.2 Verification tools are more capable and more rugged, but present usability challenges

Several studies listed in Sect. 2 and around a quarter of the projects surveyed in Sect. 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 research community has focused a considerable effort on the development of new theories to underpin verification, the improvement of tools and, to a limited but increasing extent, tools integration. The highlighted work on Mondex originally involved the manual construction of proofs; its more recent "revisiting" gives an indication of the improvements in the level of tool support in recent years, whether as a result of improved capability or the underlying "Moore's Law" increase in processing power. In our 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 and there remain many interesting challenges in the underpinning theories of verification as well as in user interaction with verification tools. We must remark that some of the free comments received in the new survey indicate that tools are still not usable, in the words of one respondent, "by mere mortals".

## 5.3 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, for example, suggests that a targeted approach can yield significant benefits on an exiting code base, when the tools are carefully integrated with existing development environments.

Successful take-up of formal verification technology involves "packing specific analyses into easier to use but more restricted components" [Bloomfield and Craigen 1999]. Such automatic analyses must return results that are comprehensible to the user [Arvind et al. 2008]. 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, and 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].

Integration with existing development processes is also significant. The recent survey suggests that 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.

## 5.4 Cost-benefit evidence is still incomplete

The surveys and highlighted projects provide anecdotal evidence for the claim that formal techniques can be used to derive low defect density systems. However, there appears only to be limited evidence on the effect on development 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 projects is 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.

Evidence on the technical and cost profiles of commercial applications may not be publicly accessible, but nonetheless, it suggests that pilot studies in verification technology are not being conducted with a view to gathering subsequent stages in technology adoption. Pilot applications of verification technology should be observable and the observations made should be relevant to the needs of those making critical design decisions. 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][1] explicitly addresses the movement from innovators to early adopters by means of studies measuring the effects of verification tools integrated into existing development processes.

There have been many calls in previous surveys for improved evidence on which to base adoption arguments. However, it is worth bearing in mind that the decision to adopt development technology is risk-based 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 the construction of 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.

## 6. 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 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]. 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

[1] `www.deploy-project.eu`

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. 6.1 is inspired by a space-flight application. FreeRTOS in Sect. 6.2 is a real-time scheduler that is very widely used in embedded systems. The bidding process for the Radio Spectrum Auctions described in Sect. 6.3 has been used in auctions with bids ranging from several thousands of dollars to several billions. The cardiac pacemaker in Sect. 6.4 is a real system, and is representative of an important class of medical devices. Finally, the hypervisor in Sect. 6.5 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.

## 6.1 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.

## 6.2 FreeRTOS

Richard Barry (Wittenstein High Integrity Systems) has 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.

There are really two challenges here. The first is to analyse the program for structural integrity properties. The second is to make a rational reconstruction of FreeRTOS, starting from an abstract specification, and refining down to working code, with all verification conditions discharged with a high level of automation. These challenges push the current state of the art in both program analysis and refinement of pointer programs.

### 6.3 Radio Spectrum Auctions

Robert Leese (Smith Institute) has 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. 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.

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. As well as the computational problems, there are challenges in getting a trustworthy infrastructure.

### 6.4 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.

### 6.5 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.

## 7. CONCLUSIONS

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, although significant advances in verification technology have yet to filter through to more widespread use.

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 tools: 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. Our impression is that times have changed: today many people feel that it would be inconceivable not to use some kind of verification tool. Whether they are right or not, there has been a sea-change amongst verification practitioners about what can be achieved: people seem much more determined to verify industrial problems. This change in attitude, combined with the increase in computing capacity predicted by Moore's Law, and the dramatic advances in research in verification technology (described elsewhere in this issue of *Computing Surveys*) mean that the time is right to attempt to make significant advances in the practical application of formal methods and verification in industry.

A significant trend is towards *invisible formal methods*, proposed by Rushby [Tiwari et al. 2003], who described it in the context of MathWorks' Simulink and StateFlow, which are 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* provide 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.

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.

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. 1996. *The B-book: assigning programs to meanings.* Cambridge University Press.

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

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.

Ball, T. and Rajamani, S. K. 2002. The slam project: debugging system software via static analysis. In *POPL*. 1–3.

Balser, M., Reif, W., Schellhorn, G., Stenzel, K., and Thums, A. 2000. Formal system development with kiv. In *FASE*, T. S. E. Maibaum, Ed. Lecture Notes in Computer Science, vol. 1783. Springer, 363–366.

Barnes, J. 1997. *High Integrity Ada: The SPARK Approach.* Addison-Wesley.

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 PRG-58, Oxford University Computing Laboratory.

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.

Berry, G. 2008. Synchronous design and verification of critical embedded systems using scade and esterel. In *Formal Methods for Industrial Critical Systems.* Lecture Notes in Computer Science, vol. 4916. Springer.

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.

Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. 2003. A static analyzer for large safety-critical software. In *PLDI*. 196–207.

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.

Börger, E. and Stärk, R. 2003. *Abstract State Machines: A Method for High-Level System Design and Analysis.* Springer-Verlag.

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.

Bush, W. R., Pincus, J. D., and Sielaff, D. J. 2000. A static analyzer for finding dynamic programming errors. *Softw., Pract. Exper. 30,* 7, 775–802.

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

Caspi, P., Pilaud, D., Halbwachs, N., and Plaice, J. 1987. Lustre: A declarative language for programming synchronous systems. In *POPL*. 178–188.

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.

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.

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. 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. 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.

Das, M., Lerner, S., and Seigle, M. 2002. Esp: Path-sensitive program verification in polynomial time. In *PLDI*. 57–68.

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.

Esterel Technologies. 2003. *SCADE Language Reference Manual*.

Evans, D. and Larochelle, D. 2002. Improving security using extensible lightweight static analysis. *IEEE Software 19*, 42–52.

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.

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., 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.

Greve, D. and Wilding, M. 2002. Evaluatable, high-assurance microprocessors. In *NSA High-Confidence Systems and Software Conference (HCSS), Linthicum, Md., March 6–8*.

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.

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.

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.

Holzmann, G. J. 2004. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley.

Hommersom, A., Groot, P., Lucas, P. J. F., Balser, M., and Schmitt, J. 2007. Verification of medical guidelines using background knowledge in task networks. *IEEE Trans. Knowl. Data Eng. 19,* 6, 832–846.

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 1985. *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985 ed. IEEE.

Inmos Ltd. 1988a. *occam2 Reference Manual.* Prentice Hall.

Inmos Ltd. 1988b. *transputer Reference Manual.* Prentice Hall.

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

Jackson, D. 2002. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol. 11,* 2, 256–290.

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

Johnson, S. 1978. lint, a C program checker, unix Programmers Manual. Tech. Rep. 65, AT&T Bell Laboratories.

Jones, C. 1996. A rigorous approach to formal methods. *IEEE Computer 29,* 4 (April), 20–21.

Jones, C. B. 1990. *Systematic Software Development Using VDM*, 2nd ed. Prentice-Hall International.

Jones, C. B. and Pierce, K. G. 2007. What can the *pi*-calculus tell us about the mondex purse system? In *ICECCS*. IEEE Computer Society, 300–306.

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.

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 phones. 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.

Larus, J. R., Ball, T., Das, M., DeLine, R., Fähndrich, M., Pincus, J. D., Rajamani, S. K., and Venkatapathy, R. 2004. Righting software. *IEEE Software 21,* 3, 92–100.

MathWorks. Simulink.

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

Meyer, B. 1992. *Eiffel: The Language.* Prentice Hall.

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.*

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.

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.

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.

RAISE Language Group. 1992. *The RAISE Specification Language.* The BCS Practitioners Series. Prentice-Hall.

RAISE Method Group. 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.

Riazanov, A. and Voronkov, A. 2002. The design and implementation of vampire. *AI Commun. 15,* 2-3, 91–110.

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.

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

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.

Spinellis, D. 2008. A look at zero-defect code. `www.spinellis.gr/blog/20081018/`.

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.

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.

Wilding, M., Greve, D., and Hardin, D. 2001. Efficient simulation of formal processor models. *Formal Methods in Systems Design 18,* 3 (May), 233–248.

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

## Acronym List

| | |
|---|---|
| ASM | Abstract State Machine |
| CA | Certification Authority |
| CADE | Conference on Automated Deduction |
| CASE | Computer Aided Software Engineering |
| CENELEC | The European Committee for Electrotechnical Standardization |
| CICS | Customer Information Control System |
| CMM | Capability Maturity Model |
| CSP | Communicating Sequential Processes |
| EAL | Evaluation Assurance Level |
| EMIC | European Microsoft Innovation Center |
| FDR | Failure Divergence Refinement |
| FSPM | Formal Security Policy Model |
| FTLS | Formal Top Level Specification |
| HCI | Human Computer Interface |
| IEC | International Electrotechnical Commission |
| FM | Formal Method |
| ITSEC | Information Technology Security Evaluation Criteria |
| KIV | Karlsruhe Interactive Verifier |
| kLoC | kilo Line of Code |
| LFU | Lookahead Fetch Unit |
| MBD | Model Based Development |
| MC | Model Checking |
| MIT | Massachusetts Institute of Technology |
| MXI | Mondex International |
| NSA | National Security Agency |
| OCL | Object Constraint Language |
| PROSPER | Proof and Specification Assisted Design Environments |
| RODIN | Rigorous Open Development Environment for Complex Systems |
| PVS | Prototype Verification System |
| RTE | Real Time Executive |
| RTOS | Real Time Operating System |
| SCADE | Safety Critical Application Development Environment |
| SDV | Static Driver Verifier |
| SIL | Safety Integrity Level |
| SLAM | Social Location Annotation Mobile |
| UML | Unified Modelling Language |
| VDM | Vienna Development Method |
| VSI | Verified Software Initiative |