First Steps in the Verified Software Grand Challenge

Jim Woodcock
University of York

The computer science research community is collaborating to develop verification technology that will demonstrably enhance the productivity and reliability with which software is designed, developed, integrated, and maintained.

"We guarantee that our software will perform substantially in accordance with the accompanying materials for a period of 90 days from the date of receipt, although the software may contain errors and the installation may not complete successfully."

This is a typical software warranty, which is essentially useless. Industry's other products come with an implied warranty of fitness for purpose, but not software. Ironically, while a piece of software might not have a proper warranty, its distribution CD probably does: If the CD is badly scratched or otherwise faulty, the supplier will replace it free of charge.

Tony Scott, the chief technology officer of General Motors, spends more than $3 billion annually on hardware and software. He says that his company would go out of business if it sold its cars the way we sell our software. In an *eWeek* interview (www.eweek.com), Scott called on software suppliers to offer a warranty that would cover any error that causes harm to a company's business. Thus, the supplier could be held accountable if it ships an application with a known security problem that subsequently crashes the purchaser's computers or affects its products. Currently, a customer can't hold the supplier liable for financial damages. Scott would want the supplier to fix the software and compensate the customer for the harm to its business.

Bugs have become an unpleasant fact for software producers. They don't offer warranties guaranteeing the absence of bugs in their software because they wouldn't be valid. The cost of these software bugs is staggering. According to a report from the US Department of Commerce's National Institute of Standards and Technology, the cost to the US economy could reach $60 billion a year. You can multiply this by three to get an estimate of the worldwide cost, a total that exceeds the gross national product of many countries.

What's to be done about this state of affairs? Can we ever expect software to come with warranties of fitness for purpose? In our vision of the future, we would expect *exactly* that.

A growing number of academic and industrial researchers believe that the way to revolutionize the production of software is by using formal methods, and they also believe that doing so is now feasible. In the near future, researchers will use formal methods to give software, even noncritical software, meaningful function and performance guarantees, and this will turn out to be the cheapest way to do it.

Given the right computer-based tools, the use of formal methods will become widespread, transforming the practice of software engineering. The international computer science community recently committed itself to making this a reality within the next 15 to 20 years.

WHY NOW?

Theoretical computer science is a mature scientific discipline, beginning perhaps with the publication of Kurt Gödel's incompleteness theorem 75 years ago, when the

0018-9162/06/$20.00 © 2006 IEEE

only existing general-purpose computers were Turing machines and electronics was in its infancy. Software development theories—widely studied as formal software engineering methods—have exploited fundamental results in computation theory, algorithm analysis, and programming language semantics. Developers use formal methods to produce precise documentation of software's purpose and to predict its behavior in practice. They also use these methods to document the interface to a software system or component as a contract between client and supplier. This documentation is amenable to rigorous mathematical analysis for compliance.

Using formal methods offers the opportunity to experiment with models, trying to find the best way to structure abstract descriptions of complicated systems. When done successfully, this provides a clearer picture of what the system is trying to achieve. This in turn leads to a cleaner architecture, which makes it easier for clients to use and understand the system.

After more than 25 years of research and application, formal methods have reached maturity.

After more than 25 years of research and application, formal methods have reached maturity. The Web contains a snapshot of the current state, providing a wealth of online resources. Universities around the world routinely teach formal methods, and a wide range of industrial projects use them successfully. What has changed recently is the arrival of a new wave of research using powerful mathematics hidden behind the scenes in program analysis and model-checking tools.

As an example, consider Microsoft's Static Device Verifier (SDV) project. At one time, Microsoft's flagship product, Windows—which is used on most PCs worldwide—was infamous because of its frequent crashes. SDV forms a small but interesting part of a large effort to improve Windows' reliability. Analysis revealed that device driver failures cause 85 percent of all Windows XP crashes. Mastering the API used to program device drivers is a complex and difficult task, and third-party manufacturers, who are not Windows kernel experts, write most device drivers.

To check whether a device driver uses the API properly, the SDV abstracts a small part of the driver code as a Boolean program, with the bits representing important observations about driver API usage. SDV then performs symbolic model checking for conformance to an abstract notion of correct API usage. For Microsoft to release SDV to third-party developers, it had to become a push-button tool, which thus embodies a large amount of domain knowledge.

SDV is a good example of sophisticated theory hidden from programmers in a usable tool. A device driver can fail for many reasons, and SDV doesn't try to find them all—it doesn't check array bounds or do performance checking. The result is a well-focused analysis tool that eliminates a certain class of error: It's like a very advanced kind of type checking.

A GRAND CHALLENGE

The Verified Software Grand Challenge is an ambitious, international, long-term research program for achieving a substantial and useful body of code that has been formally verified to the highest standards of rigor and accuracy. Tony Hoare initiated the Grand Challenge by calling on the computer science community to develop an integrated, automated toolset that developers can use to establish the correctness of software. A workshop on verified software was held in Menlo Park, California, in February 2005, and an IFIP Working Conference on Verified Software: Theories, Tools, and Experiments was held in Zurich in October 2005.

The Grand Challenge project is a 15-year research program to demonstrate the feasibility of using formal verification technology in industrial-scale software development. The program has three objectives. It will

- establish a unified theory of program construction and analysis;
- build a comprehensive and integrated suite of tools that support verification activities, including specification, validation, test-case generation, program refinement, program analysis, program verification, and runtime checking; and
- collect a repository of formal specifications and verified codes.

These programs will continue to evolve as verified code, and they may even replace their unverified counterparts in actual use.

Verification should be interpreted in a wide sense. There are generic properties of interest apart from total correctness, such as absence of runtime errors, data consistency, timing behavior, accuracy, type correctness, termination, translation validation, serializability, memory leakage, information hiding, representation independence, and information flow. There are also nonfunctional properties such as dependability and aspects of safety and security.

The goal for the verified software challenge is to develop verification technology to a point where it demonstrably enhances the productivity and reliability with which software is designed, developed, integrated, and maintained. This work will impact related areas of software development including software engineering, safety-critical systems, mathematical modeling, and artificial intelligence.

In the past, many people have said, often quite justi-

fiably, that we can't verify industrial software cost-effectively. At the end of the project, we'll be able to tell our skeptics, "You can't say anymore that it can't be done. Here, we've done it!"

PLANNING AND ROADMAP

The challenge's long-term goal is to ensure that science and practice converge, and that we routinely use and teach the principles of software specification, design, architecture, language, and semantics. In 15 years we'll have a well-developed theory, a comprehensive and powerful suite of tools, and a compelling body of experimental evidence demonstrating that we can engineer reliable software cost-effectively using formal verification techniques.

Our most important resource will be a joint research roadmap that establishes a long-term, coordinated, incremental research program.

In the first five years, researchers will lay the foundations for the work ahead through development of mature tools and standards. To do this, we must start by building cooperating communities of researchers who share the same ideals and objectives and are willing to collaborate and compete to achieve them. These communities will develop a research agenda for their specialist areas by tackling initial case studies that demonstrate the state of the art and highlight the shortcomings of current technologies. In the longer term, we'll see a closer engagement with industry and the development of a broader user base.

Our most important resource will be a joint research roadmap that establishes a long-term, coordinated, incremental research program. This will accelerate scaling of the performance, robustness, and functionality of basic verification technology. It will integrate and embed this technology in program development and verification methods and environments. It will ensure the relevance and inclusion of basic ideas in university curricula.

We will achieve these objectives through pilot projects that will evaluate feasibility and guide technology development, and through large-scale experiments that benchmark the technology. The challenges in the roadmap will be concrete, realistic, measurable, and verifiable, and we can achieve this only through international cooperation.

VERIFIED SOFTWARE REPOSITORY

An early step toward the realization of the Verified Software Grand Challenge will be the creation of a *Verified Software Repository*. The Repository will maintain the evolving collection of state-of-the-art tools, together with a representative portfolio of real programs and specifications for testing, evaluating, and developing the tools. It will contribute initially to the interworking of tools, and eventually to their integration. It will promote the transfer of relevant technology to industrial tools and into software engineering practice. It will build on the recognized achievements of practical formal development of safety-critical computer applications and contribute to the international initiative in verified software, covering theory, tools, and experimental validation. The Repository's most important contribution is that it will serve as a means of accumulating results and assessing progress throughout the project.

The Repository will be distributed among several sites around the world, each reflecting the particular interests and needs of local scientists in their contribution to the international effort. Its goals will be to

- accelerate the development of verification technology through the development of better tools, greater interoperability, and realistic benchmarks;
- provide a focus for the verification community to ensure that the research results are relevant, replicable, complementary, and cumulative, and promote meaningful collaboration between complementary techniques;
- provide open access to the latest results and educational material in areas relevant to verification research;
- collect a significant body of verified code (specifications, derivations, proofs, implementations) that addresses challenging applications;
- identify key metrics for evaluating the scale, efficiency, depth, amortization, and reusability of the technology;
- enumerate challenge problems and areas for verification, preferably ones that require multiple techniques;
- identify—and eventually standardize—formats for representing and exchanging specifications, programs, test cases, proofs, and benchmarks, to support tool interoperability and comparison; and
- define quality standards for the contents of the repository.

The Repository will form the hub of the project wheel. This is where researchers conduct the experiments that will drive the theoretical and technological innovations needed for the challenge project to be successful. The research community will be the force that drives the hub, and the challenge problems will be what motivate the community. Researchers have already proposed suitable problems, including verifying the Apache Web server, a reference implementation of the TCP/IP stack, and the Linux kernel.

MONDEX CASE STUDY

Researchers began the technical work on the Grand Challenge in January 2006 with the Mondex case study,

a yearlong pilot project. This short project is intended to demonstrate how different research groups around the world can collaborate and compete in scientific experiments and to generate artifacts to populate the Repository. We chose as our problem the verification of a key property of the Mondex smart card, a 10-year-old product, to examine the state of the art in proof mechanization.

Mondex is an electronic purse hosted on a smart card, one of a series of products developed to the high-assurance Information Technology Security Level E6 standard by a consortium led by NatWest, a United Kingdom high-street bank.

When you want to pay someone some money, you take some electronic cash out of your electronic purse and deposit it in theirs. To do this, the two purses interact with each other using a communications device to exchange the required value. Strong guarantees are needed to ensure that such transactions are secure, in spite of power failures and mischievous attacks, so that electronic cash can't be counterfeited.

Our challenge is to investigate the degree of automation that the correctness proofs can now achieve.

Completely distributed transactions compound the problem: There's no centralized control. Once released into the field, each purse must act individually, without any external arbitration, to ensure the security of all its transactions. The card must implement all security measures locally, without real-time external audit logging or monitoring.

Such products are seriously security critical. Needing convincing assurance of correctness, NatWest decided to use formal methods in its development process. The UK has a lot of experience with using Z—one of the notations recommended by the E6 evaluators—in the development of critical systems, so they chose it as the notation for modeling the Mondex protocols and proving them correct.

Susan Stepney and David Cooper of Logica carried out the work with consultancy from the University of Oxford. Together the team developed formal models of the implemented system and the abstract security policy, providing rigorous proofs by hand that the system design possessed all the required security properties.

The abstract security policy specification is about 20 pages of Z. The concrete system specification, including the *n*-step value-transfer protocol, is about 60 pages. The verification, presented at a level of detail suitable for external evaluation, is about 200 pages of refinement proof plus another 100 pages of derivation of the particular refinement rules the project needed.

Since the proof ran to several hundred pages, Stepney and Cooper had to carefully structure it to make sure it could be understood, either by a third party or by the original proof team later on. The various groups now working on Mondex have much appreciated this careful structuring.

The original proof was vital in successfully getting the required certification. It was also useful in finding and evaluating different models of the smart card. For example, during the proof, the team made a key modeling discovery of classifying certain partially completed transactions as *definitely aborted* or *maybe aborted*. This clarified the specification and the team's understanding of the protocol, and it probably would not have been discovered without doing the proof.

Finding the right abstraction meant that the security property could be stated precisely, and it explained why the protocol was secure. The value amount in a purse partway through the transfer protocol is not obvious. By having an abstract definition of the transfer, it was possible to state precisely the balance and thus show that the protocol could only create value, not transfer it. This description is useful even for someone who does not intend to read the Z specifications.

The original proof revealed a bug in the proposed implementation of a secondary protocol. The failed proof and the understanding of what had gone wrong gave a convincing counterexample that the protocol was flawed, and led to changing the design to correct it. The third-party evaluators also found a bug: One of the proofs needed to be tightened up to remove an undischarged assumption.

A commercially sanitized version of the Mondex documentation is publicly available. It contains the Z specifications of the security properties, an abstract specification, an intermediate-level design, and a concrete design, with handwritten correctness proofs of security and conformance of each design level. During the original project, mechanizing the proofs using a theorem prover for Z was not a question. Significantly, some strongly believed that mechanizing such a large proof cost-effectively was beyond the state of the art for the tools then available. Our challenge in this initial pilot project is to investigate the degree of automation that the correctness proofs can now achieve.

Several groups took up the Mondex challenge. They proposed using the following formalisms: Alloy at the Massachusetts Institute of Technology; Event-B at the University of Southampton; OCL (object constraint language) at the University of Bremen; Perfect Developer at Escher Technologies; Raise (rigorous approach to industrial software engineering) at the United Nations University, Macao, and the Technical University of Denmark; and Z at the University of York. They agreed to work for one year without funding. Separately, a group at the University of Augsburg worked on the ver-

ification using the Karlsruhe interactive verifier (KIV) and abstract state machines (ASMs).

Two distinct approaches quickly emerged in the early stages of the project. The *archaeologists* wanted to proceed by making as few changes as possible to the original documentation. According to this approach, these models have been very successful; surely, it would be cheating to change them just to make the verification easier? And if we did change the model and then found some bugs, how would we know that they had anything to do with the original specification?

The *technologists,* on the other hand, wanted to use the best proof technology currently available, but these new tools don't work for Z. They had two choices, translate the existing models into new languages or create new models better suited to new tools.

The current specification is the tenth version, comprising about 2,200 lines of RSL in 13 files, with 366 proofs, half of which were proved automatically.

Z at York

At the University of York, I worked with Leonardo Freitas to tackle the Mondex problem using the Z/Eves theorem prover. As *archaeologists*, our main objective was to mechanize all proofs, while remaining as faithful as possible to the original formalization.

We worked directly with the existing Z specifications and made changes in only two places to make explicit some information about finiteness. We succeeded in mechanizing the first 13 chapters of the monograph detailing the original work, about half of the total, and we expect to complete the remaining chapters in the near future. As this is not a full-time activity, the initial work took about a month to complete. We estimate that this initial effort took about seven working days using Z/Eves.

In addition to using the Z specifications, we also found the informal proofs useful. They indicated which theorems to prove and told us how to go about proving them using Z/Eves, which is not usually as obvious as it sounds. The later parts of the development have particularly thorough proofs that we followed closely in the mechanization.

Proving the correctness of the description of the Mondex protocols required proving about 140 *verification conditions*, and the difficulty of the proofs varies considerably. On average, each VC requires about five proof steps. Because Z/Eves has a considerable amount of built-in automation, some steps require little interaction with the prover. Other parts of proofs are repetitive steps from previous proofs, which can be abstracted into general lemmas with some effort. Some intermediate steps require familiarity with the way that Z/Eves works internally, while others are creative steps requiring domain knowledge, such as instantiating quantified variables. Finally, the process requires some general theories about language constructs (mostly finiteness results), which are sometimes hard to prove. In all, there are about 200 trivial steps, 400 intermediate ones, and 100 requiring creativity.

The work revealed some bugs. In the intermediate design level, missing properties mean that the system permits operations involving nonauthentic purses.

Our preliminary findings are very encouraging. The Z/Eves theorem prover hasn't changed over the past 10 years, so our mechanization could have been carried out during the original project, and the effort required would have been a matter of weeks, not months. What was lacking was motivation and expertise, not proof technology.

Raise at Macao and DTU

Chris George from the United Nations University, Macao, and Anne Haxthausen from the Technical University of Denmark used the Raise method and its specification language, RSL, which is inspired by the Vienna development method, communicating sequential processes (CSP), and ACT-ONE. Developers use RSL to express high-level abstract specifications as well as low-level designs, including using explicit imperative programming constructs such as loops. To verify RSL specifications, developers translate them into a prototype verification system (PVS).

The initial RSL specifications were transliterations of their Z counterparts, but the researchers soon felt inhibited by this. Because they weren't the specifications the researchers would have written most naturally in RSL, mechanizing them was awkward. They quickly changed course and created their own models directly in RSL.

They defined three levels of specifications. The abstract level describes Mondex simply as a problem in accounting. There are no purses or protocol messages; there are just three bottom-line values and some abstract operations that transfer money appropriately between them. At the middle level, there are abstract purses and concrete operations, but no details of the mechanisms that preserve the asserted invariant about the overall value not increasing. At this level, each operation is proved correct with respect to the abstract specification. Finally, the concrete level gives full details of the value-transferring protocol, and each operation is proved to implement its middle version.

The current specification is the tenth version, comprising about 2,200 lines of RSL in 13 files, with 366 proofs, half of which were proved automatically. Some proofs were particularly difficult. A typical invariant proof for the concrete level is about 300 prover commands (there are 11 of these proofs). Proving that the concrete invariant implied the abstract one was difficult

(150 prover commands). Proving that some sets defined by comprehension are finite also was difficult, a difficulty that the Z/Eves group also encountered.

The large amount of reworking of models was a feature of the original Mondex project. By starting afresh, the RSL group didn't benefit from using the modeling details that the original Mondex team had carefully worked out.

The biggest problem the group experienced with automating the proofs in RSL was identifying a suitable invariant. First, they would propose an invariant, then they would carry out a proof that the operations preserved at a particular level. Next, they would try to prove refinement to the next level, only to discover that the proposed invariant was too weak. Then they would find a stronger invariant that let them complete the refinement proofs, but then they would discover that it was too strong, and they couldn't prove that it really was invariant at the higher level. So they would weaken it again, only to find that the refinement proofs no longer worked. Thus, they were back where they started. Eventually, they were successful in finding the right invariant, but it's interesting to think how delicate the proofs are.

Although it seems worthwhile to generalize a typical proof as a reusable tactic, writing good tactics is difficult.

The RSL models have many large proofs with similar structure, and it's tempting to abstract common lemmas. Although it seems worthwhile to generalize a typical proof as a reusable tactic, writing good tactics is difficult. One incautious grind command in PVS eventually generated 1,580 subgoals.

The RSL group turned out to be technologists, preferring to construct entirely new models more suitable for their modeling and verification technology.

Perfect Developer at Escher

UK-based Escher Technologies chose to use Perfect Developer, a tool for the rigorous development of computer programs, starting from a formal specification and refining to code. PD supports the correctness-by-construction paradigm, in which static analysis verifies component interfaces to ensure that the components will strictly conform to their contracts at runtime. This work used the Perfect Specification Language, which has an object-oriented style, producing code in both Java and C++.

Escher researcher David Crocker wanted to achieve a fully automatic proof of the Mondex case study and to produce an implementation in Java. As a side effect, he wanted to learn more about how best to represent system-level specifications in Perfect and to understand and then overcome any limitations of the PD prover. Although this clearly marks him as a technologist, because the Perfect specifications are recognizable as translations of the original Z specifications, the model is faithful. Since the PD prover is fully automatic, the details of proofs are completely hidden from the user and thus don't obviously follow the originals.

The first step was to translate the concrete model from Z to PD. Crocker had to revise the refinement steps to make them more suitable for PD. Where PD did not automatically discharge the verification conditions, it provided additional assertions as hints, and where necessary, the PD team enhanced the prover. Finally, PD generated working code for the purse and other components.

Instead of starting from an atomic abstraction of the protocol, the PD team wanted to recognize that transactions are fundamentally nonatomic, and instead reformulated the security properties to consider this. The particular problem that they needed to solve was to account for value that has been debited from a sending purse, but has not yet been credited to the intended recipient. If the recipient is still expecting it, the transaction is in transit; if the recipient has recorded the transaction in its exception log, it is lost.

Currently, PD generates 213 verification conditions and proves 191 of them automatically. Some of the 22 unproven VCs are actually requirements on the environment; others are due to prover limitations. There are about 550 lines of Perfect, and the proof run takes about six hours. PD discharges all successful VCs in less than six minutes. The team spent about 60 hours on the project.

KIV at Augsburg

Led by Gerhard Schellhorn, the team from the University of Augsburg can claim the prize of being the first to mechanize the entire Mondex proof. They used the KIV specification and verification system and demonstrated that even though the handmade proofs were rigorous, they could still find small errors. They used ASM to provide an alternative formalization of the communication protocol. They are also technologists, but with a close eye on *archaeology*, as the original work clearly inspired the models and proofs.

The Augsburg team mechanically verified the full Mondex case study in KIV, except for the operations that transcribe failure logs from a smart card to a central archive. These are orthogonal to the protocol for money transfer. Z's relational approach is quite different from ASMs' operational flavor, and the two refinement theories have their differences. The Augsburg group decided to mimic the data refinement proofs faithfully to succeed in verifying the challenge, so they formalized Z's underlying data refinement theory.

The Augsburg group completed their work in four weeks. Of course, the level of expertise with formal verification influences the time needed for verification in general and with the KIV system in particular. The group required a week to familiarize themselves with the case study and to establish the initial ASMs. They took another week to verify the essential proof obligations of correctness and invariance for the ASM refinement. Specifying the Mondex refinement theory and generalizing the proof obligations to cope with invariants took another week. Finally, proving the data refinement and polishing the theories for publication took another week.

The existence of a nearly correct refinement relation helped to complete the work in four weeks. Usually, finding invariants and incrementally refining relations takes more time than verifying the correct solution. On the other hand, the group believes that sticking to ASM refinement would have shortened the verification time. The main data proofs for the Mondex refinement consist of 1,839 proof steps with 372 interactions.

The first task in the pilot project will be to capture a formal specification of the relevant parts of the standard.

The Augsburg work is interesting, both technically and organizationally. These researchers became aware of the challenge after other groups had started their work, and they completed their mechanization independently. This is limited but encouraging evidence that the verification community is keen to take up the challenges we're proposing.

NEXT STEPS

The Mondex case study shows that the verification community is willing to undertake competitive and collaborative projects, and that there is value in doing so. We call on other researchers and tool builders to join the project by reusing the Mondex example.

A series of pilot projects of increasing scope and complexity will follow the Mondex project. An ideal pilot project has the following characteristics:

- It is of sufficient complexity that traditional methods, such as testing and code reviews, are inadequate to establish its correctness.
- It is of sufficient simplicity that specification, design, and a dedicated team can complete the verification in two years.
- It will have an impact beyond the verification community.
- Existing documentation is freely available.
- It is amenable to different approaches.

Rajeev Joshi and Gerard Holzmann from NASA's Jet Propulsion Laboratory have proposed a verifiable file store as the next pilot project. This challenge satisfies the criteria given for ideal pilot projects:

- Many widely used and heavily tested file systems still contain serious bugs that can cause disastrous consequences, including deletion of the root directory.
- Most modern file systems conform to the Posix standard, and they use well-understood data structures and algorithms. This gives us confidence that a modest-sized team could accomplish the work within two years.
- Because file systems organize most electronic data, their correctness is of great importance when using computers.
- There is plenty of open source documentation for today's file systems.
- Researchers can approach the project in different ways: They can design a new file system from scratch using refinement or verify an existing open source file system. Model checking and theorem proving are both applicable.

The challenge is to produce the following: a formal behavioral specification of the functionality that the file system provides; a list of assumptions made about the underlying hardware; and a set of invariants, assertions, and properties concerning key data structures and algorithms in the implementation.

Posix uses careful informal prose to describe the behavior of functions, such as create, open, read, and write. Thus, the first task in the pilot project will be to capture a formal specification of the relevant parts of the standard. There are some existing partial formalizations, such as Carroll Morgan and Bernard Sufrin's Z specifications of the Unix filing system[1] and the Synergy file system from William R. Bevier and colleagues.[2] These efforts might be useful departure points for developing a more complete specification.

Providing a rigorous formal statement of the file system's properties, especially its robustness with respect to power failure, requires relying on certain behavioral assumptions about the underlying hardware. To make the file system useful, researchers must be precise about making assumptions regarding typical hardware, such as hard drives or flash memory. Ideally, the file system will be usable with different types of hardware, perhaps providing different reliability guarantees. Performance concerns dictate using caches and write buffers, which increase the danger of inconsistencies in the presence of concurrent thread accesses and unexpected power failures.

The implementation's proof of correctness will be formal descriptions of design properties, such as data structure invariants, annotations describing locking protection of data, and pre- and postconditions for library functions. Most typical file systems require using many common

data structures such as hash tables, linked lists, and search trees. The file system's proof of correctness will result in the development of libraries of formally stated properties and proofs of these data structures that will be useful in other verification efforts. These will be additional components that are useful for the Repository.

Joshi and Holzmann have a long-term task at JPL to solve the problem of building reliable software using automated verification tools rather than traditional ad hoc processes. As part of this task, they're building a reliable file system to use flash memory for use as nonvolatile storage on board future space missions. Flash memory is useful for this purpose because it has no moving parts, consumes low power, is easily available, and has been used on several recent NASA missions, such as the Mars Exploration Rovers and Deep Impact. Building a robust flash file system, however, is a nontrivial task.

The problem is compounded by certain faults that must be tolerated, such as arbitrary bit flips, blocks that unexpectedly become permanently unusable, and limited block lifetimes (typically 100,000 uses). In addition, a flash file system written for use on a spacecraft must obey additional constraints; for example, flight software typically is allowed to allocate memory only during initialization.

Awareness is growing in industry that something must be done about software reliability. Bill Joy, Sun's cofounder and former chief scientist, said, "I have a few more things I want to do: I still think the tools we have for building reliable software are inadequate" (CRN, Nov. 1999). *The International Technology Roadmap for Semiconductors* is an assessment of its industry's technology requirements that is intended to ensure continued performance enhancements in integrated circuits. The 2005 roadmap states, "Without major breakthroughs, verification will be a nonscalable, show-stopping barrier to further progress in the semiconductor industry."

What evidence do we have that large software companies will take any notice of this? Bill Gates described his company's progress in the area in a keynote address at the WinHec conference in 2002. Gates said, "software verification … has been the Holy Grail of computer science for many decades; but now in some very key areas, for example, driver verification, we're building tools that can do actual proof about the software and how it works in order to guarantee reliability."

This article describes the first few steps for the Verified Software Grand Challenge. If you want to get involved, visit http://qpq.csl.sri.com and join us. You can find more information by visiting http://vstte.ethz.ch or by Googling us.

References

1. C. Morgan and B. Sufrin, "Specification of the Unix Filing System," *IEEE Trans. Software Eng.*, Feb. 1984, pp. 128-142.
2. W.R. Bevier, R. Cohen, and J. Turner, *A Specification for the Synergy File System*, tech. report TR-120, Computational Logic Inc., 1995.

Jim Woodcock *is the Anniversary Professor of Software Engineering in the Department of Computer Science, University of York. His research interests include formal methods, unifying theories of programming, software archaeology, and large-scale industrial applications. Woodcock received a PhD in computation from the University of Liverpool. He is a Fellow of the British Computer Society and an executive member of the UK Computing Research Committee. Contact him at jim@cs.york.ac.uk.*