The Verified Software Initiative: A Manifesto

C.A.R. HOARE

Microsoft Research

JAYADEV MISRA

The University of Texas at Austin

GARY T. LEAVENS

Iowa State University

and

NATARAJAN SHANKAR

SRI International Computer Science Laboratory

1. INTRODUCTION

We propose an ambitious and long-term research program toward the construction of error-free software systems. Our manifesto represents a consensus position that has emerged from a series of national and international meetings, workshops, and conferences held from 2004 to 2007. The research project, the Verified Software Initiative, will attempt to construct over the next fifteen years: (1) a comprehensive theory of programming that covers the features needed to build practical and reliable programs, (2) a coherent toolset that automates the theory and scales up to the analysis of industrial-strength software, and (3) a collection of realistic verified programs that could replace unverified programs in current service and continue to evolve in a verified state.

This document summarizes the background of the initiative, its scientific goals, and the principles that underlie a worldwide collaboration to achieve them. We include an assessment of its strengths, weaknesses, threats and opportunities. A companion document will summarize a range of work packages, including developments in theory, tools, and experiments.

This draft manifesto is based on the input of a number of contributors and forms part of the Verified Software Roadmap. The list of authors may expand as the document evolves. We welcome your comments. Funded by NSF CISE Grant Nos. 0646174 and 0627284.

Authors' addresses: C.A.R. Hoare, Microsoft Research; J. Misra, Dept. of Computer Science, The University of Texas at Austin; G. T. Leavens, Dept. of Computer Science, Iowa State University; and N. Shankar, SRI International Computer Science Laboratory

DOI 10.1145/1592434.1592439 http://doi.acm.org/10.1145/1592434.1592439

2. EXECUTIVE SUMMARY

The scope of software in the modern world is unprecedented. Software is now woven into artifacts such as telephones, cars, and planes; it governs the infrastructure for systems in communication, air traffic control, and banking. Faulty software contributes to the unreliability of the products in which it is embedded and is a source of significant recurring costs and delivery delays. It exposes systems to malware attacks and raises the risk of catastrophic failure in critical applications.

Verification is the rigorous demonstration of the correctness of software with respect to a specification of its intended behavior. It can be applied at all phases of the software lifecycle from design to maintenance, and there is mounting evidence that it can reduce the cost in all these phases while delivering highly reliable software.

Automated verification technology has been advancing rapidly over the last few years. While manual verification has been successful for small critical applications, automation is required for large bodies of code. Automation makes verification results reusable, checkable, and robustly repeatable. When coupled with exponential growth in processor speed and parallelism, verification techniques can be applied to the design and analysis of large-scale software.

Verification complements and extends the capabilities of other software engineering techniques, such as requirements analysis and testing. For example, verification can be used to check the self-consistency of formally specified requirements, and bounded model checking can automate testing.

The Verified Software Initiative (VSI) outlines a program of research spanning fifteen years with the goal of establishing the viability of verification as a core technology for developing reliable software. The VSI can be summarized as follows:

- Researchers will collaboratively study the scientific foundations of software verification covering theoretical aspects, interoperable tools, and comprehensive verification experiments.
  - —Their theoretical work will focus on languages and logics for software specification, abstract models, and methods for systematically designing, analyzing, and building software, integrating these methods with programming languages, efficient verification algorithms, and theory unification.
  - —They will cooperatively vivify these theories in a seamless, powerful, and versatile toolset that supports all aspects of verified software development.
  - —Their research agenda will be driven by realistic experiments covering a range of small and large software systems.
- The software tools industry will adapt and build on the VSI toolset, incorporating verification technology into commercial tools.
- Industrial partners will contribute software artifacts as experimental material. These partners will benefit both from the VSI toolset and from the experimental verification of both substantial and smaller-scale artifacts. Researchers and graduates in verification technology will be employed in industry to apply, adapt, and extend the ideas and tools from VSI to specific needs and application domains.

3. MANIFESTO

We envision a Verified Software Initiative (VSI) to gain deep theoretical insights into the nature of correct software construction, to radically advance the power of automated tools for the construction and verification of software, and to benchmark the capabilities of such tools through convincing experiments. Our manifesto summarizes a broad

consensus of the views of a group of international leaders of the relevant computing research communities. It declares the resolve and commitment of this community to engage in an ambitious collaborative project that will realize its common scientific goals.

3.1. Background

As a society, we depend on software for business, banking, communication, transportation, and power, yet software as an engineered product comes without explicit warranties. Half of the time and resources used in software development are devoted to detecting and correcting mistakes. The resulting delays to the delivery of software are notorious. Software-controlled devices too frequently stop working, and must be rebooted to clear a software error. In some cases, the consequences are deadly serious. The reliability of software is too important to be left to chance and good intentions.

Other engineering disciplines are handicapped by the inevitable wear and tear of components due to aging and the unpredictability of the operating environment. Software does not age; it retains its freshness much like a mathematical theorem, and like a theorem, its correctness can be established with mathematical certainty. It has been known for nearly 40 years that software correctness can be cast as proving a theorem in mathematics, yet the scale and complexity of these proofs far surpasses the ability of humans and, currently, of computers.

A verified program is one that has been proved to satisfy certain desirable and clearly specified properties, and to be free of certain rigorously specified kinds of error. The proof itself is free from error, because it has been checked or even generated with the aid of software tools. Belief in program correctness is therefore as close to mathematical certainty as we can get.

Verification is applicable to all aspects of software construction and analysis: domain modeling, architectural design, formal specification, testing, refactoring, and maintenance, in addition to code verification. Even though software requirements cannot be verified against a customer's informal needs and desires, a great deal of clarity, insight, and precision can be gained by formalizing these requirements as a more precise specification. Once this is done, verification technology can be applied to the resulting formal specification, to investigate its consistency and to see if it captures important system properties such as safety or security.

Verification technology has made tremendous strides in the last decade. It has been extensively applied in hardware design, and in analyzing specific software systems—for example—device drivers, at the industrial level. Recent successes can be attributed to major theoretical advances, construction of better tools, and significant improvements in hardware capacity and performance. We anticipate that all these trends will continue.

3.2. Vision

We envision a world in which computer programmers make no more mistakes than other professionals, a world in which computer programs are always the most reliable components of any system or device. In this vision, the scientific foundations of software verification include comprehensive theories of programming supported by substantial experimental evidence of successful application. A software engineering toolset incorporating these theories plays the role of an intelligent development environment for verified software.

We envision a software industry that embraces verification and verification technology throughout the software life-cycle, including modeling, specification, design, coding,

code generation, composition, testing, and certification. This future industrial embrace of verification is supported not only by the VSI toolset, but by research into better languages and logics, semantic models (including those that unify diverse aspects of software development), and verification algorithms. In this future, society and business place justified confidence in the software industry and in its professional capabilities, enhancing the status of the software engineering profession.

3.3. Cost/Benefit

Current estimates of the cost of software error to the economies of the world are in the region of a hundred billion dollars per year. The cost is shared among the producers of software and its users. We believe that much of this cost could be recovered by improved application of verification technology.

Methods of verification are already being applied in industry, and they deliver immediate benefit in the detection and prevention of error. We expect that such applications will become more pervasive and more successful. They will be complementary to the longer-term scientific research of the VSI. Typically, industry will pursue research efforts that promise payoff in the short term for specific types of software such as floating-point algorithms and device drivers, whereas our research is a long-term effort to address the fundamental issues underlying correct software.

Industrial collaboration will be critical to the project in obtaining case material and experience. In return, industry will benefit from the project by the availability of new tools and an increase in the supply of recruits with the requisite motivation and skills.

The technology developed by the VSI will be delivered to the wider programming profession in the form of an integrated toolset for the development of verified software. We anticipate that the prototype toolset developed under the VSI will be enhanced by the software tools industry, before delivery to practicing software engineers.

3.4. Scientific Method and Ideals

This proposal is motivated by fundamental scientific curiosity to answer basic questions underlying the whole discipline of programming: What do programs do? How do they work? Why do they work? How do they interact with other programs and hardware subsystems? How can they be systematically constructed? What evidence can be given for their correctness? And how can the answers to these questions be used to improve the quality of software?

The work of the project will be complementary to the more immediately applicable industrial projects. It will be driven by fundamental curiosity, and by pursuit of scientific ideals far beyond the needs of any particular product or any particular market segment. It will aim at generality of theory, to cover the whole range of computer applications. It will seek certainty of knowledge, supported by wide-ranging and convincing experiment. Above all, the results of the research will be cumulative, because they are incorporated in scientific tools that make subsequent experiments easier and more effective.

The ideal of program correctness is like other scientific ideals—for example, accuracy of measurement for Physics, purity of materials for Chemistry, rigor of proofs for Mathematics. These absolute ideals may never be achieved in practice, or even in theory, as we can sometimes prove their impossibility. Nevertheless, scientists pursue their ideals to explore the limits of what is achievable far beyond any immediate market need for their results. Experience shows that eventually some ingenious inventor or entrepreneur will find a way to exploit these scientific advances for the benefit of the producer as well as the consumer of advanced technological products.

Commitment to scientific ideals is a factor contributing to the cohesion and the integrity of all scientific research. It is essential to the organization of a large-scale and long-term project like the VSI.

3.5. The scale of collaboration

Research on formal methods and their application is currently well represented in academia. Progress of individual researchers and teams is driven in the usual way by their scientific ideals, background, skills, and resources. The VSI project aims to build on this individual research, but will make more significant progress by organizing wide-ranging and long-term scientific collaborations.

Collaboration is also required to deal with research challenges that are growing beyond the capabilities of single scientists and research teams. Some challenges arise from the growing complexity of programming languages and libraries and the ambition of current computer applications. Others arise from the need to deal with multiple concerns, such as reliability, performance, safety, security, and human factors, and the necessity of applying research to all the different phases of the software development process. While no single academic research team can hope to handle the entirety of the problem, verification tools must cover this wide range in a seamless way.

The basic toolsets that will be developed must be supplemented by libraries of formal material (e.g., specifications, models, theories, proofs) specific to given application areas or to particular software architectures. This workload will be distributed by having research groups separately volunteer to cover each area.

3.6. Industrial participation

The programming tools developed in this project must be tested against a broad range of programs representative of those in use today. We expect that industry will contribute realistic challenge programs of various sizes and degrees of difficulty that can be freely released for scientific scrutiny.

The tools will be continuously improved in the light of experience of their use. Throughout the project, students will be trained to apply the tools to realistic programs, and many of them will obtain employment in industry that exploits their skills on commercially relevant products.

The scientists engaged in the project can volunteer any required scientific support and advice for the development of commercial tools that will spread the results of the project throughout the programming profession.

4. ASSESSMENT

The ideal of verification of software correctness dates back at least forty years. Before embarking now on a large-scale project to solve the problem, it is essential to understand and explain past failures and make a convincing case for future success.

4.1. Strengths

Verification is not another well-meaning panacea for software unreliability. It is the foundational discipline on which programming is based. Programs have properties that can typically be stated in the form of constraints on the inputs and conditions satisfied by the observable outputs. Verification makes these properties explicit, and rigorously determines whether the program has the expected property. The properties can then be used to construct further arguments involving larger programs and the relationship between a program and its external environment.

Verification technology is already widely exploited in advanced engineering industries—transport, aerospace, electronics, communications, defense, and national security. Specialized software verification tools have been developed in these areas, and are available from a number of startup companies, together with consultation and collaboration in their use. Other useful tools, such as program analyzers, are available for rapid detection of programming errors at the earliest possible stage.

Many research centers worldwide are already working towards this project's goals. Their research has yielded guarantees based on complete computer-checked verification for small microprocessors, operating systems, programming language compilers, communication protocols, several significant mathematical and logical proofs, and even the essential kernels of the proof tools themselves. We are witnessing accelerated growth in the scale and degree of automation of these verification efforts.

A number of verification tools in existence today have been developed through many years of experience. They have built up loyal communities of experimental users. We thus expect to find a large community of users for the toolset built by this project.

Improvements in the speed, capacity, and affordability of hardware are continuing at the traditional rate. There is evidence from case studies that hardware improvements over the last ten years have turned computer-aided verification into a practical and cost-effective option for real programs.

Acceleration in hardware capacity has been fully matched by improvements in the algorithms and programs that carry out the basic task of constructing proofs. In some of the basic inner loops of a theorem prover, a thousand-fold increase in speed and capacity has been achieved in roughly ten years, and this is in addition to increased hardware speeds. Both hardware and software improvements are likely to continue.

Established competitions exist between the builders of rival proof tools to evaluate their capabilities and speed. These are continuing drivers of measurable advance. The rules of a competition require the use of certain standard interfaces. These are already being further exploited to construct links for the interworking and cooperation of tools; they allow tool builders with varying backgrounds to collaborate in provision of services for their users. We expect that competition and cooperation will play significant roles in the VSI project.

The open source movement shows the power of collective idealism in the development of high-quality software. It gives an example of collaborative evolution of programs, which should be followed in the implementation and evolution of a verification toolset. The open source software provides a wealth of freely published source material, in the form of programs that can be subjected to analysis and verification.

In addition to correctness concerns, verification technology can aid other software engineering concerns, such as system security, reliability, safety, and general compliance with the system's requirements. Verification technology can be used to find and prevent known vulnerabilities that have significant economic costs, such as security holes.

4.2. Weaknesses

The main inhibiting factor during the early years of the project will be the lack of scientific staff with the right background, skills, and motivation. Another factor in the application of the results of the research will be the inexperience and lack of education of the programming profession, particularly in the difficult task of specifying what the program and its components are intended to do.

The lack of collaboration with other areas of computing is also a current weakness. Formal methods researchers must recognize the need to collaborate outside of the formal methods community in areas such as requirements analysis, testing, and fault-tolerant systems, in seeking ways to aid the cause of reliable software. They should

advance the state of programming practice by collaborating with researchers in programming languages, and improve software design tools by building on the insights of researchers in artificial intelligence, database management systems, and human-computer interfaces.

The Computing research community has little experience in long-term collaboration on an international scale. It will be necessary to learn from the experience of physicists and astronomers, who collaborate successfully on the construction and use of satellites and particle accelerators.

The current evaluation and reward structure in academic Computing departments favors rapid publication of conference papers expounding the latest research ideas of its authors. Such papers are rarely studied by practicing software engineers. In the modern world, results of research are typically made accessible, both to engineers and to scientists, through integrated toolsets. Work devoted to development and enhancement of scientific toolsets is insufficiently recognized by the current publication-based reward structures for computer scientists. Even less recognition is given to scientists who conduct experimental work based on tools and benchmarks developed by others.

The main scientific risk to the success of the project is that there is so much variation in computing application and technology that verification experience cannot readily be transferred from one program to the next; thus, no single tool will be able to meet all the demands placed on it.

4.3. Opportunities

Verification as a process of establishing properties of programs has deep repercussions for all aspects of computing including the design of hardware, the construction of operating systems and middleware, the coordination of concurrent activity, the safety of embedded systems and active networks, the autonomy of model-centric, self-healing systems, and information security. Verification addresses some of the most basic research challenges in computing including modeling, specification, design methods, type systems, language semantics, and verification algorithms.

The last two of decades of verification have yielded dramatic progress with the development of model checking, timed and hybrid systems, process calculi, SAT and SMT solvers, refinement calculi, specification languages, type theory, and abstract interpretation, along with the deep connections to control theory, game theory, artificial intelligence, operations research, and systems biology. In the next fifteen years, these ideas will have filtered into the mainstream, and new innovations and connections will be uncovered that radically redefine computational problem solving. The advances stimulated by the VSI challenge should eventually lead to a knowledge infrastructure that is founded on reliable software supported by powerful semantic tools.

Significant commercial opportunities exist for software verification products and services. A 2002 NIST report on software testing estimated the total sales of software for the year 2000 at $180 billion, and the size of the software testing market for 2004 at $2.6 billion. The arrival of multicore chips brings the challenge of programming and debugging concurrent software to the fore. Verification will be increasingly challenged to address issues of reliability, resilience, interoperability, security, and even usability.

4.4. Threats

Tool projects are often evaluated by assessing impact on industrial users. This can inhibit ideas that require longer periods of implementation, testing, and reflection. In other branches of science, the most advanced technologies are often incorporated first into scientific instruments for use by other scientists. In this way, the technology more quickly develops the maturity needed for industrial application.

Industrial users are often focused on rapid development and deployment to achieve market penetration. For such users reliability and correctness are secondary concerns, calling for methods and tools that are lightweight and adjustable.

Lightweight techniques and tools have been remarkably successful in finding bugs and problems in software. However, their success must not stop the pursuit of this project's long-term scientific ideals.

A significant inhibitor to advance in any branch of science is inadequate or intermittent or inappropriately directed funding. Continuity of funding usually requires a specific policy decision on the part of funding agencies. Such a decision has yet to be taken in favor of verification technology. The reasons for this neglect must be recognized, especially if they are likely to persist.

National funding agencies are rightly reluctant to make large-scale and long-term commitments, unless they follow certain politically acceptable goals such as interdisciplinarity. Verification calls on expertise from many schools and branches of Computing; unfortunately, it is not generally recognized as an interdisciplinary topic.

Many funding agencies rightly favor industrial collaboration in research, in the hope that rapid competitive advantage can be obtained. For such a project, the immediate needs of its industrial client are paramount. It is not widely recognized that Computing Science is a normal branch of science with a well-defined scientific agenda that transcends the pragmatic compromises needed for each of its many practical applications.

Some funding agencies rightly favor adventure in science. The VSI project as a whole is highly adventurous, and may well fail. Many individual advances that contribute to the challenge are essentially incremental—for example, to improve an existing tool or test it on a new application. These increments will have a compensating advantage: even if the project fails in its overall goal, the incremental improvements alone will more than justify the cost. The substantive advances will support novel ways of programming that can cope with the ever-increasing complexity and reliability demands of applications. The VSI challenge will surely provoke more than a few revolutionary innovations.

5. CONCLUSIONS

Correctness is a basic intellectual challenge for computing research. The Verified Software Initiative is a fifteen-year international program targeting progress in the science and technology of verification. The goal of the initiative is to demonstrate the viability of developing large-scale, bug-free software through the end-to-end use of verification technology. The research agenda is inclusive enough to accommodate all formal approaches to the development of certifiably reliable software. The goals are sufficiently ambitious that failure is a possibility. However, even partial success toward these goals can have a substantial impact on the security of the software infrastructure, the reliability of computer systems, and the scientific foundations of computing.