

Formal Methods For Developing High Assurance Computer Systems: Working Group Report

Mats P.E. Heimdahl
Department of Computer Science and Engineering
University of Minnesota, 4-193 EE/CS Building, Minneapolis, MN 55455

Constance L. Heitmeyer
Center for High Assurance Computer Systems
Naval Research Laboratory, Washington, DC 20375-5337

1 Introduction

The Second International Workshop on Industrial-Strength Formal Techniques (WIFT'98) was held in October, 1998, in Boca Raton, Florida. At the workshop, four different discussion groups investigated various topics. This report summarizes the discussions conducted on the topic "Formal Methods for Developing High Assurance Systems."

High assurance computer systems are computer systems where convincing evidence is required that the system satisfies a collection of critical properties. To operate correctly, these systems must satisfy properties such as safety and security. Examples of high assurance systems include flight control systems, medical systems, and control systems for nuclear plants. In addition, increased reliance on communications is moving many communications systems, such as telephone networks and cellular and satellite communications systems, into the domain of high assurance systems.

The aim of the 1998 discussion was to revisit and continue a discussion began in the working group with the same name at the first WIFT in 1995. A report describing the discussions at WIFT'95 is available at the web site:

<http://www.cse.msu.edu/WIFT98/>

2 Discussion Points

During WIFT'95, the following topics were discussed:

Advantages: Where and how have formal methods had utility in the development of high assurance systems?

Barriers: What are the major barriers to applying formal methods in the development of high assurance systems?

Strategies for Success: How can we obtain an industrial-strength process for developing high assurance computer systems?

Due to time constraints, some topics scheduled for discussion at WIFT'95 never were addressed. The goal of

the WIFT'98 discussion was to revisit the above topics and also to address the following:

Uses of Formal Methods: How are formal methods being used in the development of high assurance systems?

Overcoming the barriers: How can the barriers to widespread use of formal methods be overcome?

The summary below is organized around these two topics and roughly follows the flow of the discussion in the group.

3 Uses of Formal Methods

A formal method may be described as an approach to developing computer systems that includes 1) a notation with a well-defined syntax and semantics, 2) some guidelines and procedures for using the notation, and 3) techniques for analyzing specifications expressed in the notation. Traditionally, formal methods have been used for formal specification and formal verification. Recently, the use of formal methods for refutation has also gained widespread use in some segments of industry.

Formal Specification

Formal specification of the required system behavior has many benefits. For example, a formal specification is unambiguous and precise, can be checked for syntactic correctness, and can help ensure that a specification is well-formed.

Formal Verification

A formal specification enables formal verification. In formal verification, a proof is constructed, often with mechanical support, that the specification satisfies properties of interest. Several projects have successfully applied both formal specification and "light-weight" analysis to verify well-formedness properties (e.g., type correctness, no missing cases, no unwanted non-terminism). By light-weight analysis, we mean mechanical analysis that is largely automated or semi-automated. Examples of formal methods supporting both formal specification and light-weight analysis include SCR (Software Cost Reduction) [1, 2], which

Ontario Hydro and NRL have applied in real-world projects, and RMSL, which has been used in the development of TCAS II, a collision avoidance system for aircraft [3, 4, 5].

Usually more difficult and complex than verification of well-formedness properties is formal verification of application properties, such as security and safety properties. Despite the complexity of developing formal proofs of application properties, however, there have been numerous successes. Examples include the verification of the floating point algorithm for division in the AMD5k86 microcode [6], verification of clock synchronization [7], and verification of a collection of representative instructions for the AAMP5 processor [8].

Refutation

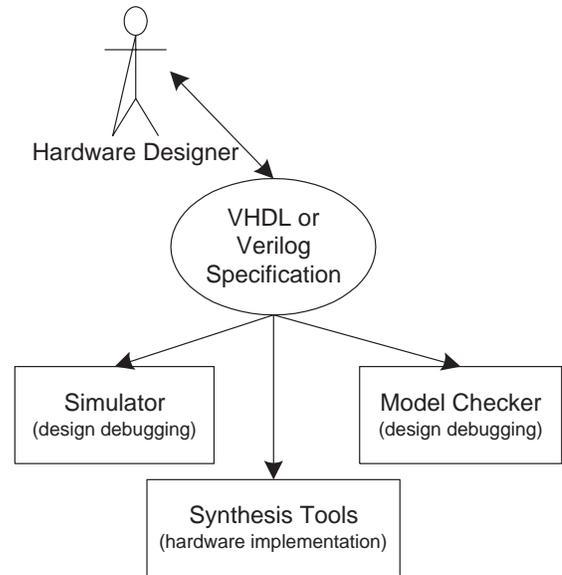
While the goal of verification is to demonstrate that a specification is correct with respect to some property, the goal of refutation is to find errors in the specification. A technique called model checking, which automatically checks a finite-state machine model for specified properties, has been highly successful in detecting errors. The use of model checkers to detect defects in microprocessor designs is now common in industry, and several commercial tools are available to perform this task [9]. In addition, some microprocessor manufacturers have in-house research groups developing proprietary technologies. The success and utility of formal methods in the analysis of chip design is undisputed. Reports of successful application of model checking, largely to detect hardware errors, are abundant (see, e.g., [10]).

Why the success in hardware?

Three factors related to the success of formal methods in microprocessor design stand out: 1) the high cost of design mistakes, 2) the availability of standard notations, and 3) the availability and use of tools, such as simulators.

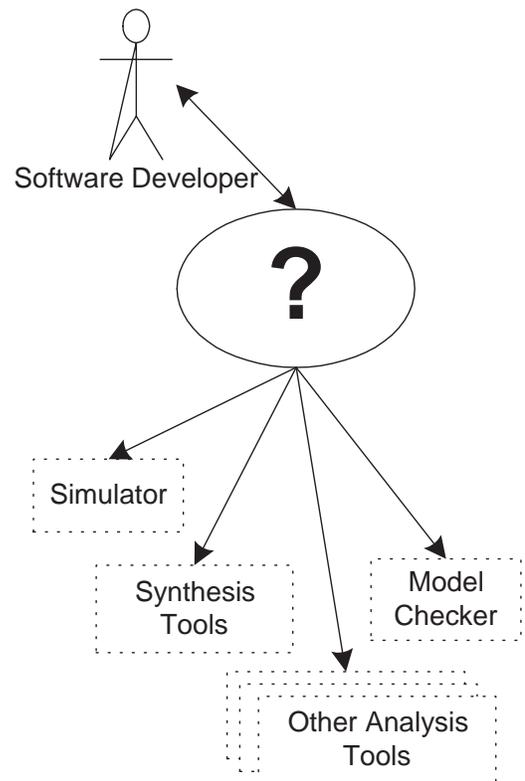
First, the cost of detecting and correcting a mistake in a chip design during the design phase is low; in contrast, detecting and correcting a mistake when the mask is produced or when the chip is manufactured is very high. Thus, there is a clear cost savings associated with each mistake detected and corrected using a formal method. This cost savings provides a compelling business case for using formal methods technology during design.

Second, nearly all hardware designers use one of two design languages, VHDL or Verilog (see Figure 1), or a variant. Hence, many engineers are trained to use these languages. Further, tools, including simulators and code synthesizers, have been developed for designs expressed in these languages. This defacto language standardization provides a focal point for research and commercial development of formal analysis tools.



Ref: C. Heitmeyer. Adopted from an invited talk at the 1998 Symposium on Formal Techniques for Real-Time Fault-Tolerant Systems [13]

Figure 1: Formal methods in hardware design.



Ref: C. Heitmeyer. Adopted from an invited talk at the 1998 Symposium on Formal Techniques for Real-Time Fault-Tolerant Systems [13]

Figure 2: Formal methods in software development.

In the domain of software development the picture is very different. First, although a number of studies have shown that significant cost savings can be achieved by detecting software errors early in the development process (see, e.g., Boehm [11]), this fact is not widely accepted among software practitioners. The common approach is to detect errors during the later development stages (e.g., coding, testing, and initial operation). Often, the solution to a software bug is to make a fix available on a web site and the only effect is a minor public relations setback. Thus, an explicit business case is lacking for using formal methods in software development.

Second, the number of different modeling, design, and implementation languages used in software development is very large; nothing close to a standard approach to software development is available (Figure 2). Further, tools such as simulators and formal analysis tools are rarely used during the initial stages of software development. Unfortunately, the costs of training engineers and of tool development are very high.

Who should use formal methods?

Formal methods may be used in two ways: 1) software practitioners may rely on formal methods experts to apply the methods, or 2) practitioners may apply the formal methods and their support tools directly. Currently, the former is the dominant approach. In industry, formal methods are perceived as very difficult to use. (At this point in the discussion, industry representatives pointed out that they are, in fact, software practitioners, and that they and their colleagues use formal methods. Admittedly, these were fairly sophisticated industry representatives from Praxis Critical Systems and Collins Commercial Avionics, but they pointed out several examples of engineers not extensively trained in formal methods using sophisticated formal methods technology. For example, formal modeling of the AAMP-FV microprocessor at Collins was performed in large part by the microprocessor designers.)

For formal methods to have a large impact on software development, this reliance on formal methods experts must be reduced and the use of formal methods must become standard in software development. Thus, we should strive for a broad dissemination of formal methods knowledge and, like the hardware community, make formal modeling and formal analysis part of standard practice.

4 Overcoming the Barriers

To make lasting inroads in industry and to put formal methods techniques and tools in the hands of the software developers, there are several barriers that must be overcome: 1) the notations must be more accessible and easy to use, 2) robust tools must be produced, 3) education must be improved, and 4) cost-effectiveness must

be demonstrated.

Notations and tools

For industrial success, the traditionally abstruse notations and difficult-to-use tools produced by formal methods research must be abandoned. There is little doubt that the software developers in industry are capable of using formal methods; if they can learn to effectively use C++, they can learn formal modeling and analysis. What is needed is a reduction in the unnecessary complexity introduced by poorly designed languages and tools.

The importance of improved notation is well known, and several research groups are addressing the problem [2, 3, 12, 13]. We expect new and easy to use formal notations to emerge in the next few years.

Providing robust and well-engineered tools is a more challenging problem. The small market for formal methods tools does not justify the high cost of quality tool development. Most tools are developed in research environments, and stability and ease of use are not the highest priorities. Nevertheless, as the penetration of formal modeling in industry increases, the availability of quality tools can be expected to follow.

A few suggestions tool developers should consider include the following:

- Provide more automation.
- Provide good feedback when errors are detected. When an analysis tool detects a problem, the error report must clearly identify the problem in the specification.
- Provide tool integration and tool interoperability. Different tools are good for different tasks and must also be usable in concert. There should be no need to use more than a single notation.
- Provide simulators, i.e., tools that symbolically execute the system based on the specification. Integrating a simulator into a formal method provides additional inducement for industrial users to apply a formal method.
- Integrate methods and tools into the software development process.

Education

Finding the time for formal methods education in the undergraduate curriculum is a challenge. Computer science and engineering are broad topics, and many students (and to a large extent industry) are simply not interested in formal methods education. Courses in topics such as graphics, multimedia, and Internet software development are more popular and are perceived

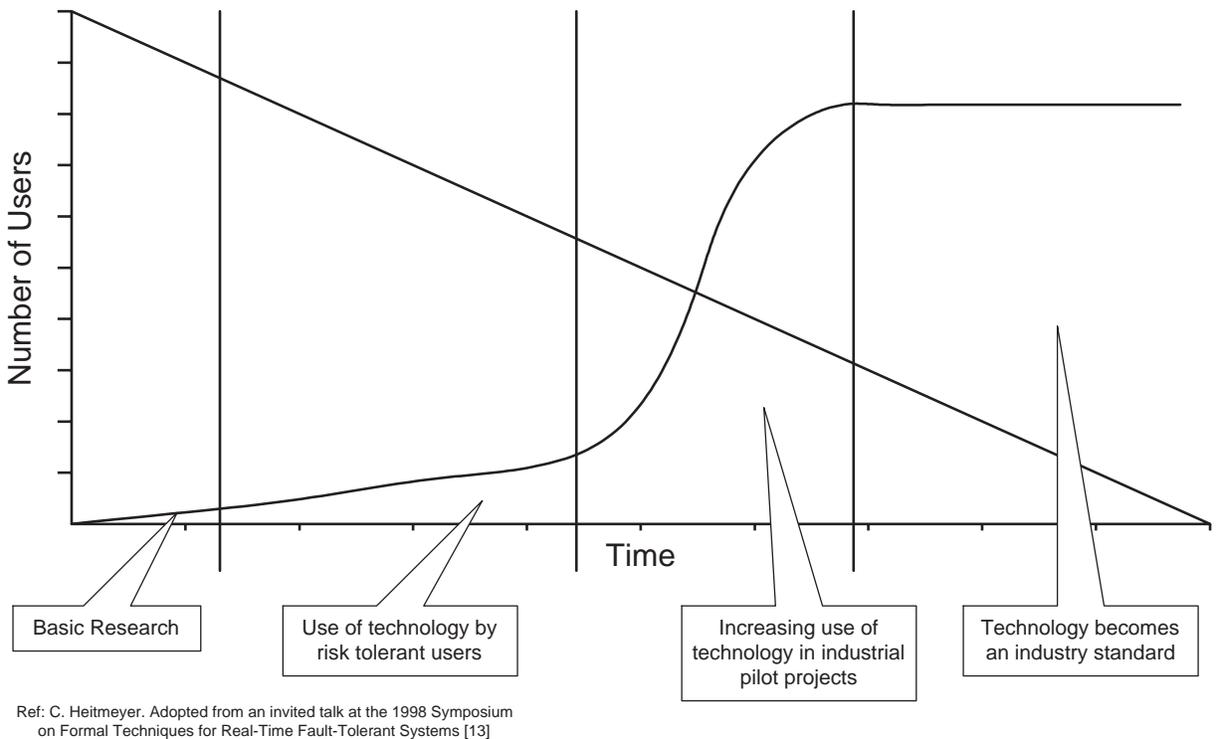


Figure 3: Technology adoption in industry.

to have more direct relevance to a student's future career. Rigorous software development (with or without formal methods) is a strenuous, time-consuming, and non-glamorous activity. Convincing students that this activity is important to software development is not an easy task.

We see no easy solution to this problem. Hopefully, as formal methods prove their worth in industry, the demand for and availability of formal methods education will increase.

Provide a business case for adoption

If we can show that the use of formal methods reduces development costs, shortens time to market, and increases productivity, management is much more likely to adopt the approach. The success of model checking and equivalence checking in the hardware community can be partially credited to the well-known high cost of classes of design problems that can be eliminated using these techniques. If we can provide similar evidence in the software community, widespread adoption of formal methods in industry is more likely to occur.

5 Summary

The consensus of the group was that formal methods are mature enough to be applied in software development. The methods have proven their worth in numerous industrial projects, and there is little doubt that they have

an important place in the software development process.

Transferring formal methods technology to industry is largely a non-technical problem (it is often a culture clash) and the transfer is happening (slowly).

The group agreed that there have been few major breakthroughs in formal methods usage since WIFT'95. The one notable exception is the increased use of model checking technology in microprocessor design. In the hardware community, the use of formal methods has moved into the rapid adoption stage, and some formal analysis tools have become part of the standard practice.

The use of formal methods in software engineering is currently limited to the early adopters (the risk-tolerant users in Figure 3). The challenge for the software community for the next few years is to follow the lead of the hardware community by successfully transferring formal methods technology into the development of high-assurance software systems.

Acknowledgements

We thank the members of our working group, Perry Alexander, Yi Deng, Detlef Fehrer, Patrice Godefroid, Anthony Hall, Steve Miller, and Ralph Johnson, for participating in the discussions and helping to extend and refine the ideas in this report. Some of the above material was presented originally by C. Heitmeyer in an

invited talk at the 1998 Symposium on Formal Techniques for Real-Time Fault-Tolerant Systems [13].

REFERENCES

- [1] M. Viola. Ontario hydro's experience with new methods for engineering safety-critical software. In *Proceedings of SafeComp'95*, 1995.
- [2] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bhara. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Transactions on Software Engineering*, 24(11), November 1998.
- [3] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, pages 684–706, September 1994.
- [4] Mats P. E. Heimdahl and Nancy G. Leveson. Completeness and consistency in hierarchical state-base requirements. *IEEE Transactions on Software Engineering*, pages 363–377, June 1996.
- [5] D.Y.W. Park, J.U. Skakkebæk, M.P.E. Heimdahl, B.J. Czerny, and D.L. Dill. Checking properties of safety critical specifications using efficient decision procedures. In *Proceeding of FMSP'98: Second Workshop on Formal Methods in Software Practice*, March 1998.
- [6] M. Kaufmann and J. Moore. An industrial strength theorem prover for a logic based on common lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, April 1997.
- [7] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of pvs. *IEEE Transactions on Software Engineering*, 21(2), February 1995.
- [8] S.P. Miller and M. Srivas. Formal verification of the AAMP5 microprocessor. In *Proceedings of the International Workshop on Industrial Strength Formal Techniques*, pages 2–17, 1995.
- [9] R. Kurshan. Formal verification in a commercial setting. In *Proceedings of the Design Automation Conference*, June 1997.
- [10] E. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, June 1996.
- [11] B. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [12] N.G. Leveson, J.D. Reese, and M.P.E. Heimdahl. Spectrm: A cad system for digital automation. In *Proceedings of the 17th Digital Avionics Systems Conference*, November 1998.
- [13] C. Heitmeyer. On the need for “practical” formal methods. In *Proceedings of the Symposium on Formal Techniques for Real-Time Fault-Tolerant Systems (FTRTFT '98)*, September 1998.