

Why Are Formal Methods Not Used More Widely?

John C. Knight Colleen L. DeJong Matthew S. Gibble Luís G. Nakano

(knight | cld9h | msg7y | nakano)@virginia.edu

Department of Computer Science

University of Virginia

Charlottesville, VA 22903

Abstract

Despite extensive development over many years and significant demonstrated benefits, formal methods remain poorly accepted by industrial practitioners. Many reasons have been suggested for this situation such as a claim that they extend the development cycle, that they require difficult mathematics, that inadequate tools exist, and that they are incompatible with other software packages. There is little empirical evidence that any of these reasons is valid. The research presented here addresses the question of why formal methods are not used more widely. The approach used was to develop a formal specification for a safety-critical application using several specification notations and assess the results in a comprehensive evaluation framework. The results of the experiment suggests that there remain many impediments to the routine use of formal methods.

1 Introduction

For many years, academics have claimed that the use of formal methods in software development would help industry meet its goals of generating a better software process and increasing software quality. The benefits that have been cited include finding defects earlier, automating checking of certain properties, and decreasing rework. Despite their popularity in academia and these claimed benefits, formal methods are still not widely used by commercial software companies. Industrial authors have expressed frustration in trying to incorporate formal technologies into practical software development for many reasons including: the perception that they add lengthy stages to the process; they require extensive personnel training; or they are incompatible with other software packages. Experts in formal methods have tried to analyse the situation and provide systematic insight into the reasons for this lack of acceptance [4, 7, 9].

The goals of the research presented here are to investigate this disparity between research and industry, and to determine what steps might be taken to increase

the benefits realized by industry from formal methods. The initial hypothesis for the relative lack of use of formal methods was that industrial practitioners were reluctant to change their current methods and hence they overlooked the benefits that formal methods could provide. However, upon attempting to apply several formal techniques to a significant application in a case study, several shortcomings that are actually well-known impeded progress dramatically right at the outset. Examples of the difficulties encountered were that a single specification language could describe only a relatively small part of the system, and necessary tools were either not available, not compatible with other development tools, or too slow.

A new hypothesis was formulated in response to these findings. This second hypothesis was that formal methods must overcome a number of relatively mundane but important practical hurdles before their benefits can be realized. These practical hurdles arise from the current state of software-development practice. While the methods used in industry are rarely formally based, they are reasonably well-developed and understood. In order to be incorporated into industrial practice, formal methods must meet this current standard.

After formulating this second hypothesis, we set out to characterize these practical hurdles. A variety of evaluations have appeared in the literature written largely by researchers and with conclusions that tend to praise formal methods. However, further investigation of the evaluations found them lacking. The criteria used for evaluation tended to be vague and ambiguous. They were often derived from the author's experience with a particular project, with little substantiation that the list of criteria was in any sense complete or even applicable to a range of projects. In addition to defects in the criteria themselves, the methods of evaluation were subjective. All of this resulted in little insight into the general characteristics or utility of the formal method.

In this paper we summarize an evaluation framework for formal methods and present results of applying the framework to several formal techniques. The complete framework provides a comprehensive list of evalu-

ation criteria together with the rationale for each. The basis of the evaluation framework is the need for any software technology, including formal methods, to contribute to one overriding goal—the cost-effective development of high-quality software. The results come from the development of formal specifications in several notations for a safety-critical application together with the application of a theorem-proving system to the application.

In the next section we summarize previous work both in the use of formal methods in software development and their evaluation. Then we present a summary of our evaluation framework, and we follow that with a summary of the results of its application in a case study¹. Finally, we present our conclusions.

2 Previous Work

2.1 Formal Specification

Formal methods have made some inroads into industrial practice. A fairly large number of projects have been undertaken using formal specification in notations such as Z, VDM, PVS, and Statecharts. The most comprehensive report on such work is the well-known study by Craigen, Gerhart, and Ralston [2].

iLogix gives summaries of some of the industrial applications in which the Statemate family of tools has been used [12]. Cardiac Pacemakers, Inc., a unit of Guidant Corp., used Statemate to speed up development of defibrillators and pacemakers. Animations of the Statechart models allowed them to examine interactions between features before building a prototype and to receive feedback on the design from physicians. AOA Apparatebau used Statemate to design a new waste system for the Airbus A330 aircraft. Animation of the system allowed them to easily test single and multiple failures. Boeing used Statecharts in the development and validation of electrical, mechanical, and avionics systems as well as in their integration [14].

The Hursley Park laboratory of IBM UK has used Z in two major projects involving CICS [11]. The first of these was the development of a new release of the system and this release showed quality improvements corresponding to the sections which were formally specified. The second project was the formal specification of the application programming interface.

SCR is a formal method developed at the Naval Research Laboratory during an effort to re-engineer the flight control software for the A-7 aircraft [8]. Since its

1. A complete report of the research can be found elsewhere [3, 6].

introduction, the SCR methodology has been expanded and more formally defined. It has been used on several industrial projects, such as a submarine communications system [10] and the certification of the shutdown system for a nuclear generating station [2], but never without the involvement of research or academic experts.

2.2 Formal Verification

Some industrial applications of formal verification have been reported using tools such as HOL, Nqthm, EVES, and PVS. Despite the large number of research projects that have used formal methods, the number of industrial projects that have utilized formal verification is quite small. Of the industrial projects that have taken place, the majority are research projects as opposed to actual practice producing real products.

By far the largest application of formal verification has been in hardware verification. Although hardware verification is not the subject of this paper, we note that successful application to hardware design is a strong indication that similar success with software is possible.

Specification analysis is the area within the software domain where theorem provers are being used. Lutz and Ampo applied mechanical analysis tools, specifically PVS, to the requirements analysis of critical spacecraft software [13]. This project consisted of specifying and analyzing the requirements for portions of the Cassini spacecraft's system-level fault-protection software. This project was more of an experimental study examining the applicability of formal methods and mechanical analysis to industrial software practices.

2.3 Evaluation of Formal Methods

Various authors have proposed evaluation criteria for formal methods and used them in a variety of ways. Rushby introduced some ideas intended to help practitioners select a verification system and also offered a set of evaluation criteria for specification notations [15]. Faulk also proposed a set of evaluation criteria for specification notations [5].

A comprehensive approach to evaluation and some results were presented by Ardis et al [1]. In this work, a set of criteria was established for the evaluation of formal specification notations and the set was then applied to several notations. The evaluation was performed on a sample problem from the field of telecommunications.

3 Evaluation Framework

3.1 Framework Basis

Our objective was to evaluate formal methods in a systematic manner, and an evaluation *framework* enabled us to generate a clear and complete set of evaluation criteria. The alternative was merely to develop a list of seemingly relevant criteria, but such an ad hoc list, though it might appear useful, leaves three important questions unanswered:

- Where did the criteria on the list come from?
- Why are the criteria on the list considered important?
- Is the list complete?

Questions such as these are not answered readily from a list of criteria. An investigation of the development of the criteria could answer these questions, but the framework summarized here provides a defensible list of criteria for the evaluation of specification languages and mechanical analysis tools.

The basis of our evaluation framework is *software development* and the associated *software lifecycle*. In other words, we seek to discover how formal methods contribute to software development and how they fit into the software lifecycle. The criteria used for an evaluation of formal methods should ultimately return to the question, “How will this help build better software?”, where the definition of better is not restricted to a certain set of goals. There are two aspects to this question—first, what is needed to build software and, second, how can the use of formal methods augment the current development practices of industry to help build “better” software?

The question of what is needed to build software leads us to examine current practice. Current methods of software development divide the activities into lifecycle phases. Such a division focuses the developers’ attention on the tasks that must be completed. But the lifecycle alone is not sufficient to describe the current process of building software since development is guided by program management activities. These activities continue throughout the lifecycle, monitoring and directing it.

An evaluation of formal methods technologies must examine their *compatibility* with current practice and the *actual benefits* they realize over the entire lifecycle. In order to be accepted by industrial practitioners, formal methods have to meet certain objectives:

- They must not detract from the accomplishments achieved by current methods.
- They must augment current methods so as to permit

industry to build “better” software.

- They must be consistent with those current methods with which they must be integrated.
- They must be compatible with the tools and techniques that are in current use.

A further difficulty is that each project has different practical requirements. For instance, if the goal of a project is to be the first commercial vendor to develop a certain networked Java application, the reliability is less important than the speed of production. In this context, “better” software would probably imply a faster time to market, whereas in a safety-critical system, “better” would refer to greater reliability of the software.

We present here only a sample of the framework because of space limitations. The complete framework is in two major parts—one for formal specification and the other for mechanical analysis techniques. The framework is structured by the six phases of the software lifecycle—requirements analysis, requirements specification, design, implementation, verification, and maintenance—and for each phase a set of criteria that must be met have been identified and documented along with the rationale for each.

As an example of the way in which the framework operates, consider the oft-cited criterion that a formal specification language must be “readable”. In practice, this is completely inadequate as a criterion because it is imprecisely defined and is untestable. In practice, a formal specification is read by engineers with different goals, skills, and needs in each phase of the lifecycle. What is readable to an engineer involved in developing a specification is not necessarily readable to an engineer using the specification for reference during implementation or maintenance. Thus, there are in fact *many* important criteria associated with the notion of a readable specification—the criteria are determined by the lifecycle phase and their relative importance by the product goals.

A selection of the criteria used for formal specification in the framework is presented in the next subsection. For brevity here, they are not broken down by lifecycle phase. In addition, they were chosen for illustration and are in no sense complete. In general, we use criteria for illustration that have not been noted by others, and we include the rationale for each.

3.2 Criteria for Formal Specification

- *Coverage.*
Real systems are large and complex with many aspects. For a specification notation to serve well, it must either permit description of all of an application

or be designed to operate with the other notations that will be required.

- *Integration with other components.*

A specification is not developed in isolation, but rather as part of the larger software development process. Any specification technology must integrate with the other components of this process, such as documentation, templates, management information, and executive summaries. Often a system database and version control system are used. A part or all of the specification might be inserted into another document, so the specification must have a common file format. There will almost always be the need for a printed version of the specification. It should be easy to print the entire specification, including comments and non-functional requirements, in a straightforward manner. The formal method technology must be suited to the larger working environment.

- *Group development.*

Every software project involves more than one person. During the development of the specification, it must be possible for several people to work in parallel and combine their efforts into a comprehensive work product. This means that any specification technique must provide support for the idea of separate development—a notion equivalent to that of *separate compilation* of source programs in languages such as Ada that have syntactic structures and very precise semantics for separately compiled units.

- *Evolution.*

A specification is not built in one effort and then set in concrete; it is developed and changed over time. The specification technology must support the logical evolution of specification and ease its change. Incompleteness must be tolerated. Functionality such as searching, replacing, cutting, copying, and file insertion must be provided. Modularity and information hiding must be facilitated, so that for example a change in a definition is automatically propagated to every usage of it. Large scale manipulations must also be supported, such as moving entire sections or making them subsections.

- *Usability.*

The ability to locate relevant information is a vital part of the utility of a specification. The ability to search, for example with regular expressions is valuable, but not sufficient. The specification is intended to serve as a means of communication. Annotating the specification with explanations, rationale, or assumptions is important for both the use of the specification in later phases and for modifications of the specification. This annotation must be easy to create

and access, and it must be linked to a part of the specification, so that changes effect the corresponding annotation. The formal method should also provide structuring mechanisms to aid in navigation since the specification document is likely to be large. In a natural language document, the table of contents and index assist in the location of information; many tools allow them to be generated automatically from the text. Another useful capability seen in text editing is the use of hypertext links to a related section or glossary entry. Formal methods must address the usability of the resulting specification documents.

- *Compatibility with design tools.*

A very strong relationship exists between the specification of a system and its design, therefore the tools should also be closely related. It should not be necessary for the designer to re-enter parts of the specification that are also part of the design. Either the specification tool must also fully support the design phase or it must be compatible with common design tools.

- *Compatible with design methods.*

Just as the specification technology must be the same as or compatible with popular design tools, it must also be compatible with popular design methods. A difficult transition from a formal specification to say an object-oriented design is an unacceptable lifecycle burden.

- *Communicate desired system characteristics to designers.*

In order to design the system, the designer must be able to read and understand the specification. Naturally, the specification should describe the normal operating procedure, any error conditions and the response that is appropriate, and non-functional requirements. The specification has to answer every question raised about the system by the designer (who is not likely to be an author of the specification).

- *Facilitate design process.*

The more easily a design can be developed from the specification, the better. The use of a formal methods could speed up the design process by describing the system clearly and precisely. The designer must take the abstract description in the specification and describe how a real system is going to implement the specification. Information hiding must be maintained and the ability to view the system at varying levels of abstraction must be provided.

- *Implementation performance.*

Implementation is hindered by any lack of clarity in the specification (and design) and misunderstandings

that cause rework. The more complete, precise, and detailed the specification and design are, the more smoothly implementation will go. An improvement in implementation efficiency is expected, therefore, from the use of formal specification because of its ability to achieve clarity and precision. This efficiency improvement is a critical element in the overall cost effectiveness that is realized by introducing formal specification into the lifecycle.

- *Support for unit testing in implementation phase.*

A precise, complete, and accurate specification can greatly aid in the formulation of a unit test suite, perhaps through automatic generation. It should also minimize rework, since the requirements are well defined and unambiguously stated in the specification. Again, this expected benefit is a critical element in the overall cost effectiveness argument.

- *Maintenance comprehension.*

An engineer responsible for maintenance should be able to study the specification and gain either a broad or detailed understanding of the system quickly. The documentation of non-functional requirements and design decisions is vital to a complete understanding of the system. The specification should be easy for the maintainer to navigate, accurate, complete, and easy to reference to find answers to questions. Structure, support for information hiding, and the ability to view the specification at different levels of abstraction are essential.

- *Maintenance changes.*

When a change is made to an operational software system, the specification, the design, the implementation, and the verification must be changed. This is clearly facilitated if the different work products are carefully linked together so the changes needed in the code, for example, are very similar to those in the specification. In many current developments the specification is changed as an afterthought or not at all. Ideally the specification should be changed first to examine the effects of the change on the system. This requires that the specification be easily changed and that the document remains well-structured. Once changed, formal methods could allow static analysis, animation, or even the establishment of proofs of properties on the new specification before the change is propagated to the code. Clearly both validation and verification of a maintenance change are important and lifecycle support is required.

4 Experimental Evaluation

To evaluate the utility of a formal technique in industrial practice with any degree of statistical rigor, the technique must be tested in a large number and variety of projects. The projects chosen for study should encompass a wide range of application areas with a variety of goals. The population of engineers involved should consist of experienced industrial software practitioners, including clients, managers, designers, developers, technical writers, and maintainers. Finally, projects should be followed from conception through a period of maintenance, and measurements of productivity and product quality made before and after the addition of formal methods to the development process, so that a comparison can be made.

Unfortunately, a study with these characteristics would require many years, the cooperation of thousands of people, and is beyond the scope of this endeavor. The study reported here is quite modest and the results correspondingly modest. What we report: (a) is based on a single application of a particular type; (b) comes from a single development activity; (c) involves specifications that have not yet proceeded to implementation; and (d) is based on specifications that were not developed by experts.

We have applied the evaluation framework to a small but realistic safety-critical application in order to obtain assessments of various formal techniques. The application is the *University of Virginia Reactor (UVAR)*, a research reactor that is used for the training of nuclear engineering students, service work in the areas of neutron activation analysis and radioisotope generation, neutron radiography, radiation damage studies, and other research [16].

The experimental evaluation was conducted by first developing three separate specifications for part of a control system for the reactor in three specification languages—Z, PVS, and Statecharts—and establishing proofs of safety properties of the PVS specification using the PVS system. During the creation of these artifacts, various observations and measurements were made by those involved in the development. Once the artifacts were complete, a second set of observations and measurements were made by the developers, computer scientists not involved in the development, and nuclear engineers and reactor operational staff.

4.1 The Case Study Application

The UVAR is a “swimming pool” reactor, i.e., the reactor core is submerged in a very large tank of water. The core is located under approximately 22 feet of water on an 8x8 grid-plate that is suspended from the top of the

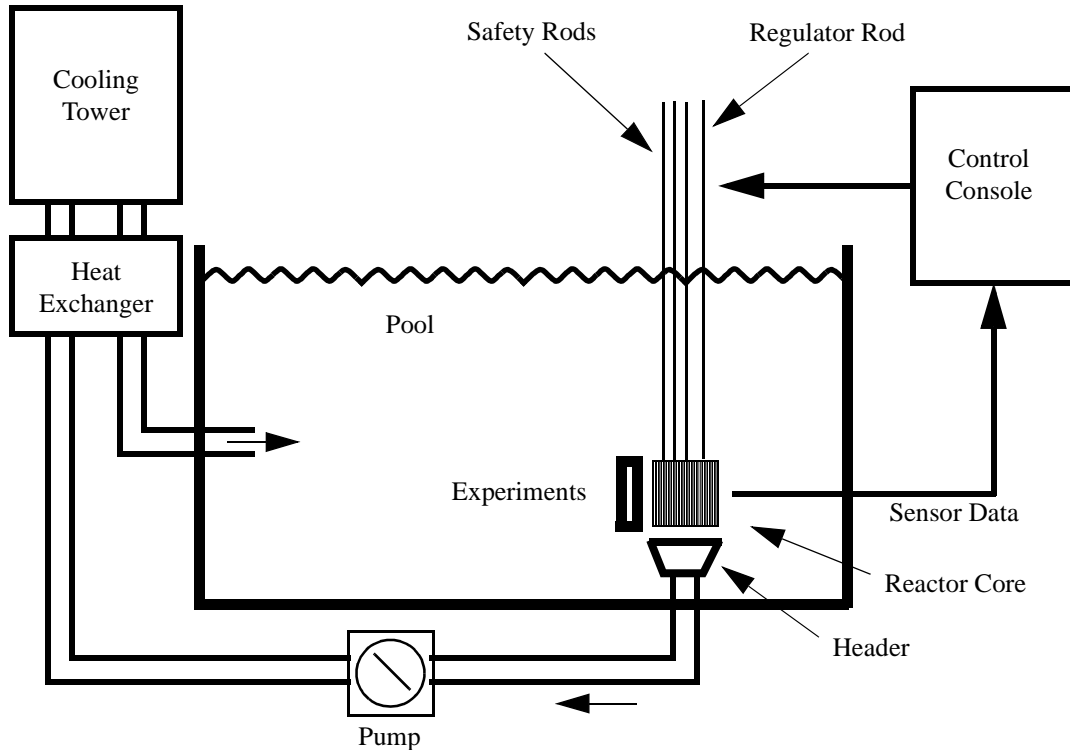


Fig. 1. - The University of Virginia reactor system.

reactor pool. The reactor core is made up of a variable number of fuel elements and in-core experiments, and always includes four control rod elements. Three of these control rods provide gross control and safety. They are coupled magnetically to their drive mechanisms, and they drop into the core by gravity if power fails or a safety shutdown signal (known as a “SCRAM”) is generated either by the operator or the reactor protection system. The fourth rod is a regulating rod that is fixed to a drive mechanism and is therefore non-scramable. The regulating rod is moved automatically by the drive mechanism to maintain fine control of the power level to compensate for small changes in reactivity associated with normal operations [16].

The heat capacity of the pool is sufficient for steady-state operation at 200 kW with natural convection cooling. When the reactor is operated above 200 kW, the water in the pool is drawn down through the core by a pump via a header located beneath the grid-plate to a heat exchanger that transfers the heat generated in the water to a secondary system. A cooling tower located on the roof of the facility exhausts the heat and the cooled primary water is returned to the pool. The overall organization of the system is shown in Fig. 1.

The evaluation that we undertook involved the development of formal specifications for the following

three components of a proposed new digital control system:

- the alarm system that alerts the operator of conditions needing attention;
- the logic associated with shutting the reactor down in the event of a possible safety problem (the SCRAM logic); and
- the activities undertaken by the operator to start the reactor operating.

For the sake of brevity, we only summarize the results of the study¹ in the following subsections. The first subsection address the specific evaluation criteria outlined earlier and reflect the experience of the developers. The next subsection itemizes specific results obtained from the nuclear engineers. The last subsection documents results obtained from computer scientists

4.2 Specification Assessment By Developers

- *Coverage.*
Our experience with the UVAR specifications is similar to that of others—many things that have to be

1. Further details can be found elsewhere [3, 6].

specified are not covered by any of the notations we are using. A particularly significant example is the user interface. For systems like the UVAR, the user interface is complex, absolutely critical, and must be formally specified. Even though a model-based specification notation, like Z for example, is not really suitable for such specification, its integration with other notations is essential.

- *Integration with other components.*

There is a complete lack on compatibility of the tools for the three notations with common text preparation systems. It is remarkably difficult, for example, even to get a printed copy of a specification in any of these notations. Worse is the fact that non-formal elements of a specification cannot be included in a specification and manipulated in a consistent way. A complete specification is more than the formal part. In the UVAR specification, for example, extensive technical background material has to be included.

- *Group development.*

Statemate offers some support for version control and access control but neither Z nor PVS provide either. The latter is actually preferable because artifacts using the notations can then be handled by existing tools. The Statemate approach to projects and users is completely inconsistent with that which large projects are likely to be using for other purposes.

Support for separate development (akin, as noted above, to separate compilation of source code in languages like Ada) is completely absent from all three notations.

- *Evolution.*

The structure of both standard Z and the PVS specification notation offer no support for building specifications with any structure that facilitates evolution. Even the elementary notion of information hiding is absent. Statecharts offer some limited support using the hierarchical chart facility. In the UVAR specification, for example, there is extensive material related to physical devices that might change over time. Similarly, since the digital system is experimental, the concepts it includes are subject to change.

- *Usability.*

Both the structure and the tool support associated with these three notations provide essentially no support for navigation and searching of a specification. The PVS specification for the UVAR, for example, defines literally dozens of identifiers. Reading, navigating, and changing a specification of even the UVAR's moderate size is difficult and error prone.

- *Compatibility with design tools and design methods.*

Although specification and design are supposed to be separate activities, there is always a lot of overlap. The SCRAM logic for the UVAR, for example, is a significant part of the specification and a clear implementation structure is implied by the basic functionality required. Despite this, neither Z nor PVS provides a systematic link to any design methods or tools, nor do they explicitly avoid doing so in an effort to support generality. Statemate, on the other hand, embeds the notion of data flow diagrams into the basic specification structure and thereby biases designs towards structured design, an approach that is not universally preferred.

4.3 Specification Assessment By Nuclear Engineers

The following results were obtained by interview. For each of the specification notations, the notation and the associated specification were presented to a nuclear engineer and then the engineer was asked a series of questions derived from the evaluation framework. This process was repeated twice for each notation (making a total of six interviews in all).

The presentation of the notation was informal and brief, intended only to permit the nuclear engineer to understand the subsequent presentation of the specification. The presentation of the specification was intentionally very much like an inspection. As a result, we were able to get very specific information about how understandable the specifications were to domain experts. This is an important issue since, for the most part, human inspection is the primary vehicle for specification validation.

The results were quite unexpected and the detailed discussion resulting from the interviews was more enlightening than the specific answers to the framework questions. The majority of the following are observations that resulted from these discussions. The first three points are general and the remainder are language specific.

- *The role of the specification has to be understood.*

Communicating with people from a different field of expertise is always difficult. In this experiment, a particularly troublesome issue was the role of the specification in software development—the nuclear engineers were not familiar with this role. One of the participants considered one of the specifications to be source code and wanted to see the execution to check correctness. Another considered it a summary that should be easy to read and not contain many details. The lesson learned was that it is vital that application

engineers understand the role of a specification before trying to read or manipulate one.

- *Direct and indirect influence on the system are difficult to distinguish.*

A common difficulty for the nuclear engineers in understanding the specifications was with the difference between direct and indirect influence on the state of the system. The nuclear control system is reactive—it is constantly making alterations in response to input received from sensors. A change in the height of a rod causes changes in the sensor values. The height of the rod can be altered directly by the system, but the sensor values change indirectly as a result of the movement of the rod.

The formal specification notations designate parts of the system that can be influenced directly differently than those that cannot, for example Z uses primed identifiers to indicate items that are changed directly by operations. These designations were a constant source of questions because, along with the changes in the system from direct influence, there are expected indirect changes in the state of the system. By no means is this an argument to abolish the separate designations for items that can be directly influenced, rather to point out a difficulty in understanding these notations that is forgotten once the notation is familiar.

- *The use of symbolic constants is problematic.*
An interesting anecdote involves the use of constants. It is customary, in fact preached, in computer science that constants should be defined in one place and given symbols so that no “magic” numbers are used throughout the rest of the system. The reasons are first that the numbers are unexplained, and second that every location of use has to be found if the constant is changed. To most of the nuclear engineers, this organization was clear and desirable since they did not have the constants memorized and the values would have to be checked against other documentation in an effort separate from the general perusal of the specification. However, one participant was confused by the use of constant identifiers rather than numbers because the specific values have important meanings in the context of the application.
- *There is no road back to natural language specification.*
Once the nuclear engineers had experience with one or more of the formal specification notations, they said they would never trust a natural language specification again. They were impressed by the level of understanding of the system that was required to write the specifications and felt that with natural lan-

guage they could never be sure that the words were not just copied down with little understanding of the system. While they would have liked some natural language to accompany the formal specifications, they wanted to retain the formalisms.

4.3.1 Z

- *Effective for communication.*
The Z specification was described as meaningful and useful for communication by the nuclear engineers. One participant felt comfortable with the notation after a short period of time, no longer needed full translations of the schemas, and began to find errors in the specification. This participant felt that, after a few iterations of discussion and correction of the specification, he would feel that there was a mutual understanding of the system.
- *Mathematical notation is not familiar.*
A surprising discovery was that the mathematical notation used in Z was not familiar to the nuclear engineers. One participant expressed the desire for a glossary of symbols, including for all, there exists, and implies. Another asked why words, which are universally understood, were not used in place of the symbols.
- *Validation by inspection was effective.*
In this case, the presenter of the Z specification was not the author, but another computer scientist familiar with the project, and the process of explaining the specification to the nuclear engineers uncovered errors. This helps to substantiate the generally accepted view of the community that inspection is valuable and cost effective.

4.3.2 PVS

- *Looks like source code.*
The first impressions of the PVS specification were that it looked like source code, it was too long, and there was too much text. One participant said he did not even want to try to read it. Another criticism from another participant was that there were too many variables leading to confusion.
- *Validation by inspection was effective.*
Although one of the participants was not comfortable reading the PVS notation, a detailed explanation of the specification facilitated useful discussions that identified errors in the specification and in the specifiers’ understanding of the system. One way that this occurred was that the nuclear engineer asked questions to check the model. He identified a misunderstanding of the power levels of the reactor that necessitated the redesign of a section of the specifica-

tion. If this error had not been found until the system had been implemented, it would have been impossible to increase the power level of the reactor above about half of the value at which it is licensed to operate. The use of meaningful variable names was key to the understanding of the specification.

In addition to errors found by the nuclear engineers, presenting the specification caused the specifier to discover an error in his own specification.

4.3.3 Statecharts

- *Effective for communication.*
After less than an hour of introduction to the Statecharts notation and specification, one participant was no longer intimidated by the notation and was able to understand the specification without assistance. The graphical notation was appealing, as well as the obvious flow of the system following the arrows. The cliché “a picture is worth a thousand words” was used repeatedly. The structure of the specification was much more evident in Statecharts than the other two notations because of its hierarchical nature.
- *Difficult to search and navigate.*
In a very detailed examination of the specification, participants complained of the difficulties of knowing the state of the whole system at once and of identifying the results of actions since the actions could affect any page of the specification. Whenever the details of a state were included in the diagram of that state rather than being saved in another file, the lack of abstraction seemed to be confusing.
- *Easy to learn.*
Within two hours of discussion of the specification, the participants displayed the desire to learn the syntax of the notation in order to understand the subtleties of the specification. A large number of errors were identified during the discussion of the specification and the need for additional robustness was evident. The participants found the specification easy to understand with the explanation from the specifier and felt that they could then continue to study it alone. They also felt comfortable enough with the notation that, if there were changes to be made to the system, they felt they could write Statecharts of the proposed changes.
- *Specification is superior to existing documentation.*
The participants from the nuclear reactor staff felt that the specifiers understood the system better than most of the operators. They felt that they could eventually come to an agreement that the Statechart specification correctly described the system and did not feel that they would have the same confidence with

an English document. They said that this specification had the potential to be used in the training of their operators and perhaps even to replace their SAR which describes the control of the nuclear reactor. These are significant comments.

4.4 Specification Assessment By Computer Scientists

The participants in this portion of the study were seven computer science students. There was one undergraduate, four students working toward or finished with a master’s degree, and two Ph.D. candidates. Two participants had a year or less work experience developing software, three had one to five years experience, and two had more than five years of work experience. All had knowledge of the C programming language. Regarding their experience with formal specification methods, four had no experience prior to this study, two had a segment of a course, and one had an entire course. All had some, but not extensive, knowledge of basic science and engineering and little to no knowledge of nuclear reactors.

4.4.1 Z

- *Fairly easy to understand, navigate, and search.*
The Z specification was generally well-structured and this aided the participants in understanding and searching the specification. However, one participant expressed difficulty locating the definitions of types since they are not defined near their use and another suggested that the specification would be easier to search, navigate, and use for reference if there were a table of contents. The participants felt strongly that Z would aid communication about the system, however they considered it only average for use in the maintenance phase as an introduction to the system and as a reference document about the system. Familiarity with logic symbols, the smallness and simplicity of the notation, and the natural language descriptions aided the participants in understanding the specification.
- *Reasonably easy to learn.*
None of the participants felt very confident in their ability to use Z after this short introduction. A few of the participants felt that Z was harder to learn than a programming language, but most felt that it was as easy or easier to learn. Difficulties in learning Z were attributed to the mathematical notation, the unusual delimiters of inputs and outputs, and the unfamiliarity of the notation in general. No one thought that Z was too large a notation and almost everyone thought the complexity of the notation was appropriate for speci-

fication.

- *Implementable.*

After a thorough inspection of the description of the SCRAM logic in the specification, everyone saw ways that it could be implemented. No one was sure that the description was complete, however. Some participants found errors in the specification. Upon quick perusal of the rest of the specification, almost everyone felt that all the features of the notation were implementable. It was practically unanimous that Z provided the appropriate level of detail about the system for a specification.

4.4.2 PVS

- *Difficult to understand, navigate, and search.*

Although PVS is structured a lot like source code in C (of which all participants claimed a lot or extensive knowledge), it received low ratings in the areas of structure, understandability, and searching. One participant cited the formatting as hindering understanding. It was deemed average to bad for use during the maintenance phase as an introduction to the system or as a reference document. The answers were widely varied as to whether PVS would aid communication between people involved in the software development process.

- *Ease of learning mixed.*

None of the participants felt confident using PVS after this short introduction. Most felt that PVS was as easy or easier to learn than a programming language, but a few felt that it was harder to learn. No one thought that the PVS notation had too few features and most people thought that it had the appropriate amount of complexity, while a few felt that it was too complex. Difficulties in learning the notation were attributed to the size and complexity of the notation and the difficulty in understanding the keywords and constructs. However, some participants felt that the keywords and constructs were easy to learn and PVS was similar to other notations with which they were familiar.

- *Implementable.*

After examining the scram logic in the PVS specification, everyone saw ways that it could be implemented, but a few saw some problems. No one was certain whether the description of the scrams was complete. After a quick inspection of the rest of the specification, the participants felt that everything was implementable. There was a wide range of responses when asked whether PVS provided the appropriate level of detail for a specification.

4.4.3 Statecharts

- *Easy to understand.*

Statecharts was described as well-structured and this aided the participants in understanding the specification. Difficulties in understanding the specification were attributed to the global nature of events and the division of the specification over many pages. The responses indicated strongly that Statecharts would aid communication between people in the development of a software product.

- *Difficult to navigate and search.*

The structure of Statecharts aided in searching, but one participant noted that the specification would be easier to navigate, search, and use as reference, if it had a table of contents. It was deemed average for use in the maintenance phase as an introduction to the system and as a reference document.

- *Fairly easy to learn.*

The participants did not feel confident in their ability to specify a system using Statecharts at this point. Difficulties in learning Statecharts were attributed to the notation being unlike any notation they had seen before and the constructs being difficult to understand. However some people felt that Statecharts was easy to learn because the notation was familiar, graphical, small and simple, and the constructs were easily understood. Most of the participants thought that Statecharts was as easy or easier to learn than a programming language.

- *Implementable.*

After studying the scram logic described in the Statecharts specification, everyone saw ways to implement it, however no one was certain the description was complete. After a quick survey of the specification, almost every participant thought that all the features of the notation were implementable. It was almost unanimous that Statecharts provided the appropriate level of detail about the system. Most of the participants thought that Statecharts notation contained the appropriate level of complexity.

4.5 Mechanical Analysis With PVS

The PVS specification was subjected to limited analysis with the PVS theorem-proving system. The purpose was to evaluate the difficulties involved in dealing with this modest sized specification and to learn what the practical issues might be that are limiting the wide-spread application of mechanical theorem proving. This part of the study was performed by the authors.

The conclusions from this part of the study fall into two basic categories. The first concerns the “method”

part of formal methods. Devising the requisite theorems and developing a proof strategy for them proved to be a significant challenge and there is no real “method” that can assist the specifier.

The second category of conclusions is in the area of tool performance. Although the PVS system is very powerful, this power is difficult to use. Some of the difficulties with the tool are the following:

- *Syntax and type checking are laborious because the system reports errors individually.*
- *The specification interface is very awkward to use since, for example, it does not permit many display items to be customized, does not provide status information conveniently, and lacks expressivity.*
- *Navigation through a specification using the toolset is extremely labored.*
- *The variation in delays that occur with different user actions makes interactive use very difficult.*
- *The theorem prover interface is awkward to use since, for example, information is not displayed conveniently during proof attempts, proofs are re-displayed after invalid commands, and certain commands generate an overabundance of output.*

These and many other observations lead us to conclude that the practical adoption of mechanical theorem proving by industrial practitioners is being severely limited by one major problem—the difficulty of determining what should be proved to gain confidence in a specification, and one relatively minor problem (or at least a problem that should be minor)—the relatively poor usability of the toolset.

5 Conclusions

Our assessment of the formal technologies that we used is that there are many practical barriers to their routine use in industrial software development projects. In most cases, this will not be “news” either to the developers of the techniques or the community at large. In fact, some developers have been quite open in their discussion of the pragmatic weaknesses of their technologies. Thus, we offer little specific new information. However, the accumulation of all the different criteria in our framework together with their systematic development provides a clear picture of what is needed to achieve success in industrial applications. It is important to keep in mind that the criteria are not sufficient, merely necessary.

Several of our results are surprising but two are repeated here because of their significance. Both of

these comments arose during the interviews with the nuclear engineers:

They felt that they could eventually come to an agreement that the Statechart specification correctly described the system and did not feel that they would have the same confidence with an English document.

Once the nuclear engineers had experience with one or more of the formal specification notations, they said they would never trust a natural language specification again.

These are very positive comments although when reading them it must be kept in mind that the nuclear engineers involved had been exposed to this technology for only a short time. However, these remarks provide strong motivation for continued work in the area of formal methods.

Perhaps the most important conclusion to be drawn from this work is that the framework provides a detailed research agenda for workers in this field. The potential is tremendous but unless the criteria in the framework are met by specific formal methods, their chance of widespread acceptance is remote at best.

6 Acknowledgments

It is a pleasure to acknowledge those who patiently described the requirements of the UVAR control system including Tom Doyle, Bo Hosticka, Don Krause, Bob Mulder, and Reed Johnson. The participants in this study also deserve recognition: Roger Rydin, Reed Johnson, William Dixon, Jim Kelly, Bob Mulder, Tom Doyle, Emily West, Dale Newfield, Russ Haddleton, Craig Chaney, Cassie Trontoski, Meng Yin, and Chenxi Wang. This work was supported in part by the National Science Foundation under grant number CCR-9213427, in part by NASA under grant number NAG1-1123-FDP, and in part by the U.S. Nuclear Regulatory Commission under grant number NRC-04-94-093. This work was performed under the auspices of the U.S. Nuclear Regulatory Commission. The views expressed are those of the authors and do not necessarily reflect the position or policy of the U.S. Nuclear Regulatory Commission.

References

- [1] Mark A. Ardis, John A. Chaves, Lalita J. Jagadeesan, Peter Mataga, Carlos Puchol, Mark G. Staskauskas, and James Von Olnhausen. A Framework for Evaluating Specification Methods for Reactive Systems: Experience Report. *IEEE Transactions on Software Engineering*, 22(6):378–

- 389, June 1996.
- [2] Dan Craigen, Susan Gerhart, Ted Ralston. *An International Survey of Industrial Applications of Formal Methods*. U.S. Department of Commerce, March 1993.
- [3] Colleen L. DeJong, Matthew S. Gible, John C. Knight, and Luís G. Nakano. Formal Specification: A Systematic Evaluation. Technical Report CS-97-09, Department of Computer Science, University of Virginia, Charlottesville, VA, June 1997.
- [4] David Dill and John Rushby. Acceptance of Formal Methods: Lessons from Hardware Design. *IEEE Computer*, 29(4):23-24, April 1996.
- [5] Stuart Faulk. Software Requirements: A Tutorial. *Technical Report NRL/MR/5546—95-7775*, Naval Research Laboratories, November 14, 1995.
- [6] Matthew S. Gible and John C. Knight. Experience Report Using PVS for a Nuclear Reactor Control System. Technical Report CS-97-13, Department of Computer Science, University of Virginia, Charlottesville, VA, June 1997.
- [7] Anthony Hall. What is the Formal Methods Debate About? *IEEE Computer*, 29(4):22-23, April 1996.
- [8] Kathryn L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering*, 6(1):2-13, January, 1980.
- [9] C. Michael Holloway and Ricky W. Butler. Impediments to Industrial Use of Formal Methods. *IEEE Computer*, 29(4):25-26, April 1996.
- [10] Constance Heitmeyer and John McLean. Abstract Requirements Specification: A New Approach and Its Application. *IEEE Transactions on Software Engineering*, 9(5), Sept. 1983.
- [11] I. Houston and S. King. CICS Project Report: Experiences and Results from The Use Of Z In IBM. *VDM '91. Formal Software Development Methods, Vol. 1: Conference Contribution*. Lecture Notes in Computer Science, Volume 552, Springer Verlag, 588-596.
- [12] Some industrial uses of iLogix tools can be found on-line at: <http://www.ilogix.com/company/success.htm>, 1997.
- [13] Robyn Lutz and Yoko Ampo. Experience Report: Using Formal Methods For Requirements Analysis Of Critical Spacecraft Software. In *Proceedings of the 19th Annual Software Engineering Workshop*, pp. 231-248, Greenbelt, MD, December 1994. NASA Goddard Space Flight Center.
- [14] C. R. Nobe and W. E. Warner. Lessons Learned from a Trial Application of Requirements Modeling using Statecharts. In *Proceedings the Second International Conference on Requirements Engineering*, pp. 86-93, April 15-18, 1996.
- [15] John Rushby. Formal Methods and the Certification of Critical Systems. *Technical Report CSL-93-7, SRI International*, December 1993.
- [16] University of Virginia Reactor, The University of Virginia Nuclear Reactor Tour Information Booklet can be found on-line at: <http://minerva.acc.Virginia.EDU/~reactor/>, 1997.