# Managerial Issues for the Consideration and Use of Formal Methods

Donna C. Stidolph[1] and James Whitehead[2]

[1] Snaptrack Inc., Campbell, California
donnas@qualcomm.com
[2] University of California, Santa Cruz
ejw@cse.ucsc.edu

**Abstract.** The introduction of formal methods into the commercial community has been slow. This might, in part, be due to lack of guidance for program managers responsible for execution of a program, who must justify added expense or time to delivery. This paper first provides managers with guidance in deciding if a particular project is a good candidate for the use of formal methods. After the go/no go conditions are considered, the paper describes some of the management and reporting issues that may arise if the decision is made to use formal methods on a program.
**Keywords** Program management, schedule, formal methods, specification, requirements, cost

## 1 Introduction

The formal methods community recognizes that formal methods are not often used in commercial software developments or well understood by most practicing software engineers [7, 24]. There appear to be a few commonly recognized reasons for this [33, 35, 2, 7, 12, 4]

- Application of formal methods requires the explicit use of discrete math skills.
- The effect of the use of formal methods on schedules is not well understood.
- The development tools are awkward and may not integrate with standard industry design tools.
- An expert is necessary to get started.

Each of the items above effectively add to program costs or risks. There are many concrete examples of this. British Aerospace reports a factor of seven decrease in programmer productivity going from a non-safety critical development project to a full formal methods development. Other surveys report that productivity is reduced by half [5, 3]. There are published cases where overall time to product delivery on a formal methods project is equivalent to that predicted for a traditional delivery. In these cases, though, the application of formal methods shifts the labor curve toward the end of the project. This potentially conflicts

with a common business model in which costs are recovered based on customer identified milestones, which puts a premium on getting to actual code production. Many organizations that have tried formal methods, even in small research areas, seem to find the process too painful to repeat, so the work is abandoned and the small efforts are not scaled up to provide adequate real-world examples.

Other software methodologies have overcome introductory hurdles similar to those listed above and ultimately have been accepted in the production software world, so why not formal methods? There seem to be a number of reasons, but they all reduce to lack of return on investment. There have been very few instances where the use of formal methods has resulted in even the perception of cost savings [19]. It is enlightening to consider that one of the most accepted uses of formal methods in software is in the production of protocols and algorithms, where the cost of the formal methods can be amortized across all uses of the algorithm or protocol [24, 27]. Although formal methods can result in improved safety and reliability of software products, few organizations assign dollar values to improvements in those areas. If there is no measurement in place for the improvement, it is perceived as a cost with no resulting benefit.

However, the business environment is changing in ways that make formal methods more attractive:

- Use of formal methods are being mandated in some safety or security critical applications [32, 6].
- Litigation as a result of failed software is becoming more common [5]. Use of formal methods could demonstrate the developer's concern for reliability and safety.
- As e-commerce moves toward large scale business collaborations, it will become increasingly important to provide all participants with some proofs of integrity and formal methods may provide them.

In light of this, it is natural to ask if there are rules a software manager could follow to select appropriate uses for formal methods and to increase the chances of a successful application of formal methods.

This paper provides some guidelines for application of formal methods. A literature search was performed to find case studies involving use of formal methods in the software industry. Although very few formal methods projects were documented with the appropriate data and in sufficient detail for management comparisons, what information was available was used to start developing some rules of thumb to help on deciding when and how to use formal methods. As a result, our contributions are ones of synthesis, surveying the known literature with an eye towards managerial issues of adoption, and managerial guidelines for use of formal methods on projects. This paper does not introduce any new formal methods techniques or tools, instead focusing on certain of the managerial issues surrounding their use. Management of formal methods projects is one of a cluster of topics, including engineering curricula, in-practice training, and empirical validation, that are critical for crossing the gap between theory and practice in formal methods.

The management guidance provided by this paper is divided into two parts, deciding whether to use formal methods at all, and then, assuming they are used, rules of thumb for their application. The next section of this paper identifies conditions that must exist for formal methods use to succeed. Next, the paper provides a series of managerial issues to be considered in the event that formal methods are used on a project. The paper concludes with a brief summary of results.

## 2 Go - No Go Conditions for Application of Formal Methods

In order for a formal methods effort to succeed, the developing organization must have a compelling reason to use them, the effort should be structured so that the developing organization can control the amount invested in the effort, and management must be behind the effort. Each of these conditions is justified and discussed in more detail below.

### 2.1 You Have a Really Good Reason

A really good reason would be that your customer requires them. This isn't restricted to customers who explicitly require them, such as Great Britain's Ministry of Defense (MoD); you may be in a business that doesn't have a customer base that is sophisticated enough to realize the risks incurred by depending on software. If that's the case, you may decide to apply formal methods because it's the right thing to do. This might be the if your company develops heavy equipment assembly line control software or other types of manufacturing software; if there is a personal safety risk to employees, formal methods might be appropriate. Finally, there may be a competitive advantage if you do it first in security-aware sectors such as finance.

Another good reason would be that you expect this software to be around virtually forever. In the case of power plant control software [11] or an enterprise business product such as IBM's CICS [16], the expected life cycle of the software is decades, with continuous maintenance and upgrades. It is in your best interest to lock down every assumption and every corner case on every test because your experts won't be there to tell you what they were thinking in 25 years, particularly after 25 years of continuous maintenance. In the shorter term, having a fully proven test suite allows software maintenance to be approached with much greater confidence, since the likelihood of detecting unintended consequences is much higher. This results in lower design/analysis costs for the maintenance efforts and lower domain knowledge requirements for the maintenance programmers, both significant productivity enhancements.

### 2.2 The Development is Internal, or on a Shared-Risk Contractual Vehicle

There are virtually no metrics for estimating cost or schedule for formal methods projects [15]. In the 66 cases studied, only 7 claimed equal or better cost numbers as compared to traditional developments [45, 35]. In the rest of the cases, the cost was not specifically addressed. For the cost/benefit analyses which would be required to justify incorporation of formal methods, cases where cost is not reported are not useful. It should also be noted that published case studies are probably not representative because they are usually domain specific and implemented by various All Star Teams. Finally, most of the projects are on "toy-sized" applications, not the millions of lines of code that is common in commercial software. Experienced formal methodologists insist that cost and schedule estimation techniques are unsatisfactory and will remain so until a large body of experience becomes available [4].

Since the cost/schedule impact of formal methods can't be predicted, it seems unwise to volunteer to use them unless you can control the amount you are willing to invest in them. If the proposed use is internal development or some type of contractual vehicle in which the customer shares risk (cost plus fixed fee, cost plus award fee, etc.), then formal methods can be considered. In these cases, the financial risk can be somewhat bounded. Interestingly, in one case study even though the acquirers needed help interpreting the formal system description, the use of formal methods was "important in the acceptability of the proposal" [13], meaning that the customer saw enough value to invest in being educated.

### 2.3 You Have Management Buy In

Use of formal methods might be expected to extend the initial phases of development (analysis, requirements and specification) and compress the design, code and test phases of a program schedule. Since this affects the investment profile for a program, your management must be aware of this and accept that initial investment will probably be higher on a formal methods program.

Figure 1 compares the time spent in each phase of development on a development program for a satellite control system and the (very successful) formal methods development for a rework of that software[3] [42].

Note that at the end of the specification phase, more than twice the hours were expended on the formal methods development as on the traditional development, but by the end of the test phase, the formal methods approach had proven much less expensive. Changes such as those shown in Fig. 1 are of serious concern for two reasons:

[3] An argument could be made that the effort on the rework should be lower because of the knowledge gained the first time. In this case, the task was designed as an experiment in applying formal methods, so the implementers selected had no previous domain knowledge or project-specific knowledge. However, they were formal methods practitioners.

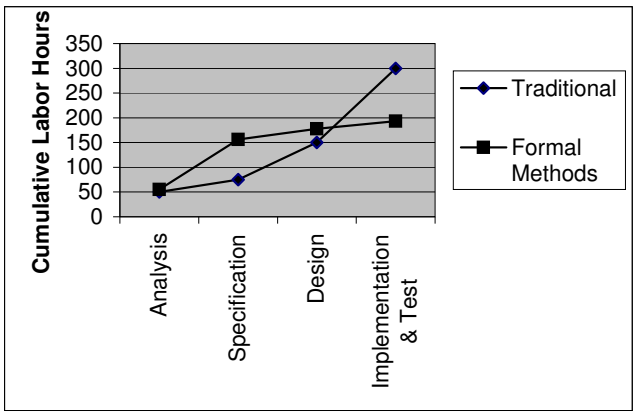


**Fig. 1.** Comparison of labor investment over time on a formal methods development versus a traditional investment

- The costing profile of the program is altered; that is, money is spent at different times in the program than in a traditional development.
- The drop dates of the interim deliverables are changed.

If an organization is working with a contract based on a traditional development model, these two factors may combine to make the developing organization's investment significantly higher for formal methods programs. Milestone payments, a common method of payment distribution in development contracts, are based on completion of various program phases, signified by the completion of a deliverable item, or by acceptance of some work product. Since the customer (internal or external) wants his investment to be proportional to the amount of useful "product" he currently holds, the payment schedule is usually back-loaded, so that the payments for requirements analysis and specification are small in comparison to those for design, code, and test.

In essence, the use of formal methods shifts the workload toward the beginning of the program. As can be seen, the investment by the developing organization is significantly higher for the formal methods development until completion of the design phase, even though total spending is decreased. As a result, the developing organization may find itself going into debt in the initial phases of a formal methods program, even a remarkably successful effort, rather than making an initial profit. If management is unaware of this, the program might be red-flagged early, getting saddled with the "costs of panic" that always accrue to a program that is thought to be in trouble, due to increased reporting requirements.

## 3 If You Still Think You Might Do It . . .

If you have a project that cries out for formal verification, you have the power to cut your losses if the process isn't working, and you have management buy in to try [28, 46], here are some guidelines for proceeding:

### 3.1 Plan on Using Consultants

Most of the formal methods programs studied used expert formal methods teams, and all of them had expert consultants available. In the Formal Methods Europe (FME) database (www.fmeurope.org), 90% of the projects were done either by academic institutions, or in direct cooperation with one. If this is a one-time or few-time effort, like many formal methods projects are [44], you'll have a sharply limited opportunity to recover your training investment. Unless you believe that your organization is going to make a habit of using formal methods, the benefits accrued by training engineers to be experts in formal methods probably don't justify the cost of the training. In light of this, you might consider out-sourcing the entire effort [28, 17, 18]

Even if you decide to train your own staff, there is another reason you need consultants: to meet a commercial development schedule, you need instant expertise to quickly analyze the selected problem, identify the appropriate level of rigor and type of formal method to apply, and select the tool to match the problem. There are many method/tool/language combinations that have dedicated adherents, and each has different technical and cultural strengths, just as with programming languages. Just because a formal method has a large support base does not mean it is the right choice for your program. On the other hand, selecting the perfect tool may be a bad choice if the only expert is its author and her graduate student, who both live in time zones far, far away. The formal methods community isn't geared to provide general purpose tools, so you need an expert to select them, as well as to run them.

As always, there are disadvantages to outsourcing. First, your experts may come with a strong commitment to an existing formal method. Many of the formal methods were developed in research settings, and many experts became experts by developing new tools or by applying an academic tool to an industrial problem. As the saying goes "When you have a hammer, everything looks like a nail." The FME website has a current list of methods and a high level roadmap for the selection process. Use this website or something like it to inform yourself a little, so you can at least ask questions to ensure that your expert is open to considering multiple alternative solutions.

The major disadvantage of outsourcing is that the formal methods expert will probably not be a domain expert, and will require training about the domain. Communication between the formal methods experts and the domain experts was cited as a problem in several of the case studies [6, 15]. In some cases, the formal methods team was required to come up with an alternative method of presenting their specification in order to get it properly reviewed with respect to domain-specific technical content. Alternately, several of the participants cite

misunderstanding of the domain by their formal methods team as a serious impediment to their progress. Viewing that from a different perspective, Easterbrook et al. [18] points out that the process of educating their formal methods experts resulted in detection and resolution of several requirements problems, as well as exposing areas that lacked clarity in the requirements, both valuable outcomes.

### 3.2 Formal Methods Leverage the Existing Software Process, Not Replace It

In most cases in which formal methods were successfully applied, the formal methods were used to augment an existing successful process to meet a new requirement[4]. Formal methods cannot be used to develop software, only to improve the chance of getting good software. For example, some formal methods tools support continuous refinement of design through code generation, but the requirements had to be identified before they could be formally specified, and the resulting code has to be tested and its configuration managed, all functions included in a mature software process. Another characteristic of organizations with successful software development processes is that there is a process for introduction of change, and recognition that different types of projects require different development models and tools [39]. Organizations such as these are likely to try new approaches, such as the addition of formal methods, in carefully selected pilot programs, with identified baselines for expected technical and schedule results. When change is planned for, it can be accommodated. As Fitzgerald and Larsen point out,

> In introducing such organisations to formal techniques, we are not in a position to require radical changes in development processes in the short term. This has to be done gradually as part of a process development plan.

The application of formal methods delays the beginning of the coding phase of a program. In light of this, formal methods will probably not be successful in an impatient organization, and patience is usually a characteristic of an organization with a mature process [28].

### 3.3 Expert Consultants Must Be Available

As mentioned previously, all the formal methods application programs reported on here were accomplished either by teams made up entirely of experts, or had

[4] One exception was a Tektronix oscilloscope development. In this case, formal methods were introduced to help bring clarity to the architectural design process. Although the use of formal methods was abandoned, this instance of use was judged to be a success based on the fact that the resulting product line was successful in a flat economy, and several generations of oscilloscopes have been built on the architectural base.

ready access to expert consultants. NASA has concluded that, at the least, expert consultants should be available to formal methods programs, and ideally, at least one team member should be an expert [25]. Surveys of participants in software projects that had used formal methods [44, 40] also included this as a lesson learned. So, even if you send your people to schools, expect them to need support and be prepared to provide it.

### 3.4 You Need "Early Adopters"

There must be a community of "early adopters"[5] on the program so their enthusiasm can be used to bridge through the training and initial frustration. Formal methods have to be introduced in the same way as any new technology. If you don't have someone eager to try the technology and support it when frustration sets in, the introduction won't be successful [46]. In the published cases, the implementers entered the project with confidence in the effectiveness of formal methods. For your formal methods project to be successful, you need to find someone to inculcate that confidence and enthusiasm in your own team and in the customer's - or do it yourself.

### 3.5 Know Exactly Where You Need The Extra Effort

Very few of the projects applied formal methods to the entire design, and those that did elected to start analysis several levels of detail down from the top [32] in order to avoid combinatorial explosion of proofs [24, 41].

Another reason to restrict the scope as much as possible is to keep the analysis timely and synchronized with development. Easterbrook [18] uses the term "lightweight formal methods" to "indicate that the methods can be used to perform partial analysis on partial specifications, without a commitment to developing and baselining complete, consistent formal specifications." Easterbrook describes the use of lightweight formal methods used selectively on identified problems with the results fed back into the existing process. NASA has used this process successfully in several cases, and a group at Nortel has concluded that selecting the right problem is critical to successful use of formal methods [48].

Scalability is another unknown in formal method developments. Most of the projects documented were relatively small, ranging from 1 KLOC to 200 KLOC, but many commercial and custom development projects run to millions of lines of code. Even if the formal methods tools and techniques worked superbly on the samples, there is no guarantee that they will work well on a large program. We all know that changes in size can cause changes in kind as well as quantity. With this in mind, risk can be bounded by carefully selecting the right, small portion of your program logic for formal methods application.

[5] Or "Early Majority-ers" as they are called in Geoffrey Moore's book Crossing the Chasm, which Craigen discusses in Craigen, D. (1999) Formal Methods Adoption: What's Working, What's Not., SPIN'99, Toulouse, France, Springer-Verlag.

### 3.6 Select an Appropriate Program Phase

Several of the projects that used formal methods for specification, design, and verification noted that the majority of the value came in using formal methods to develop a specification [23, 29]. The projects surveyed provide little evidence of value added by formally proving a program, either manually or with an automated theorem checker. There are probably a couple of reasons for this. One reason is that the formal methods tools for design don't appear to be as mature as the tools for specification. Another reason is requirement collection and specification are weak points in many processes. As a result, formal methods might provide the most value-added in the specification phase of the program by forcing rigor.

Using this as a guide, formal methods could be appropriate either in new development programs during the requirements/specification phase, or in the reverse engineering phase of re-engineering an existing system from its as-built configuration. In either case, the process of formally specifying the system can provide an unambiguous description of system requirements that can ensure the internal consistency of the requirements set and can be used for generating test cases. In the case of a greenfield system, it can be used to expose weaknesses in the natural language specification. In the reverse engineering application, creating a formal specification from the existing code allows the user to compare the as-built with the original specifications and also to verify what the code actually does, while forming a basis for a new requirements specification.

It should be realized that the formal methods should be applied after user requirements are relatively firm and the essential requirements are identified. It is difficult to express the concept of "desirable but not essential" in these languages [22].

If applied after the development is done, use of formal methods to reverse engineer the as-built specification can cost up to 50% of the original development [31]. If the system has been in the field for some time and the actual body of code has lost synchronization with the documentation, formal specification may be worth a look. If the alternative is maintaining the software with inaccurate documentation and resolving inconsistencies between the documentation and the code via testing, formal methods may actually represent a cost saving. "Specification Mining" is a new development that may help with this problem [1]. This approach assumes that a fielded program is mostly correct and infers the formal specification based on the behavior of the existing, running code.

There have also been successful pilot projects using code generation tools, in which the code was automatically generated from a formal specification and integrated with manually developed code [26]. This approach allows the formality developed in the specification stage to be "automatically" floated to the coding phase of a program.

### 3.7 You'll Probably "Lose" Your Domain Experts to the Development

Your domain experts may become absorbed in the formal methods effort and they may be lost as cross-organizational assets. Usually the domain experts who are most valuable to any single project are valuable to multiple projects, and using formal methods, even with consultants, will require them to learn enough formal methods to talk to the consultants. The learning process will cause more of the domain expert's time to be taken than a normal, intra-organization consultation does, since it involves, essentially, learning a new language [28]. Be prepared to defend your asset.

### 3.8 Select the Right Languages and/or Tools

Tool selection will be a critical decision, which should be based on suitability for your intended use [22, 10] and to match your existing process [20]. As mentioned above, it's probably best to have expert help making this decision. Just as in programming languages, it is important to match the formal method language to the problem. Also, again as in programming environments, the contents of formal methods tool suites are inconsistent, so you may be limited in your selection of tools by your application.

If you aren't driven to a single language or tool by your application, selection should be based on availability of experts and/or training and ease of learning. Furthermore, one project may require multiple notations for different aspects of the program [27, 22].

When assessing the formal methods tools, be aware that one of the major self-criticisms of the formal methods community is that the tools they produce are not usually robust, reliable, scalable or stable. The tool developers seem to try to extend the power of the tool, rather than making the present version more usable in a production environment [25, 22]. This is why access to an expert is so crucial. Your team needs to know quickly whether it's them, their theorem, or the tool that's misbehaving.

Finally, can the tool be qualified? "Qualification" is the process of meeting a guideline for development and maintenance to help ensure a given level of stability, reliability and performance. In certain application domains, all software, hardware, firmware and the tools to create each of them, such as compilers, have to be qualified. For example, in the US, FAA DO-178B is used as a qualification guideline for software for airborne systems. To be included on a US built aircraft, software developers must publish a plan that addresses essentially all phases and aspects of software development and maintenance. After the plan has been accepted, specific procedures must be written to implement the plan. Finally, the developer must create an audit trail to prove that the plan has been complied with [43, 14]. Guidelines such as DO-178B exist for many safety-critical domains.

If you're working in an environment that requires hardware and software qualification, it seems appropriate that you would want your formal methods

tool or environment to be at least as trustworthy [2]. In fact, DO-178B requires that tools that automate a software task typically done by humans must be qualified. If a tool has not been qualified by its vendor, the user of the tool is responsible for qualifying it. Selecting a previously qualified tool, or one with extensive, accurate documentation, will make this process much less expensive.

A relatively new possibility in the area of tool selection is the translation of a formal specification from one notation to another, allowing use of the most appropriate tool/notation for the problem or the aspect of the problem that is currently under investigation [30]. The technical problem with this is that the translation or the translation tool needs to formally verified. The management problem is that there are relatively few people competent to help with the matching of problem aspect to formal approach.

### 3.9 Formal Methods Must Complement Other Techniques in Use

To be successful, formal methods need to be integrated into existing processes and support, and be supported by, other design notations and artifacts. Formal methods have been used successfully in combination with data flow diagrams, entity-relationship diagrams and various object modeling forms. The participants stated that the diagrams were useful for identifying system boundaries and providing a starting point when identifying interface relationships. There is ongoing work in formalizing UML and object oriented modeling techniques [9]. Andrew Butterfield, who has worked on a number of formal methods applications, says [10]

> "Do not rely on one model alone. Products of sufficient complexity give rise to different views. Checking for cross consistency will usually identify errors in understanding the requirements."

Good advice whether or not one of the models is a formal method.

Many formal methods practitioners feel that the output of the formal methods tool alone was not sufficient to communicate with either other developers or the customer [36, 47]. Since a formal methods specification does not provide an overview of the system [21], it is necessary to provide natural language expansions and explanations along with the formal methods product, particularly in the non-functional areas of discussion. A particularly bothersome aspect of this arises when your customer has to report on your project within her organization: in some cases [21], the customers had difficulty communicating about their system within their own organization, due to the use of formal notation. If your customer can't explain your project status to her boss, your program is in trouble.

In one of the most interesting developments, Heitmeyer has used domain specific front ends on simulators to allow validation of specifications by domain experts, combining the advantages of prototyping and formality [24]. The IBM team that developed the rehearsal scheduler, also developed a documentation style that embedded user interface information into the specification [45].

### 3.10 Estimating the Cost Of Formal Methods

There is little or no history available for the cost and schedule impact of using formal methods. What history is available is of low value to most organizations due to differing processes, application domains, changes in tool suites, differing levels of in-house expertise, etc. The normal view of costing on formal methods projects is summed up by Koob [34] discussing the results of several short pilot programs using a formal methods framework called VSE-Tool:

> "Formal development, even using VSE, takes considerably more effort. The result might be a better product but it might also be too late and significantly more expensive than conventionally developed products. The solution to this still has to be worked out."

That said, one of the case studies used parallel development teams on the same task. The cost distribution was slightly different, but the two teams finished in nearly the same amount of time and spent nearly the same amount of money. And IBM's development of CICS actually resulted in an estimated 9% savings in development costs [8]. In a case where an Air Traffic Control (ATC) system was being built, the percent of engineer time spent in each phase was within 4% of the COCOMO prediction [21], thus highlighting that formal method use can result in schedules comparable to traditional development schedules. In the development of power plant control software, significant cost savings were achieved in comparison with the estimated cost of doing a traditional development [11]. All of these cases were developments where the teams had software processes in place and were already successful at working within a documentation structure.

An additional factor to consider is that there might be long term cost benefits not captured by current metrics. On the ATC system, although the number of defects found during system testing was similar, the number of post-delivery problems found was lower for the portions of the system that had been subjected to formal methods [22]. Since both types of code passed system test with the same number of defects, this implies that the code developed using the formal methods more closely matched the end user's expectations for characteristics that weren't documented by the requirements specification. Strengthening this argument, an IBM representative estimated that there is a 40% decrease in post-delivery failures on their software developed using formal methods when compared to their software developed using other methods [44]. Since the goal of all documentation and design is to enable us to meet or exceed customer expectations, this might be an extremely significant result.

Alternately, there are long term costs associated with the use of formal methods. For example, for formally specified and proved systems, the cost of change is may be large when compared to a traditionally specified system. If the investment has been made to formally specify and/or prove a system, to preserve formality, any change has to be inserted with the same level of formality. Furthermore, if the change affects interface properties of the module, the proofs that depend on those have to be reproven and little mention is made of this problem

in the literature, except to note that it exists [37]. If the cost is excessive, this will result in fewer change proposals and less profit made on upgrades.

However, this may be a perceived problem, rather than a real problem: the life cycle cost may actually be less since the cost of maintaining an undocumented code base is uncertain. Parnas refers to a common software phenomena he calls "ignorant surgery", in which changes are made to a software baseline without deep understanding of the purpose and underlying design of the code base [38]. As a result, "repairs" and "enhancements" may result in the loss of key features or, possibly worse, an unexpected change in their behavior, resuling in large financial losses due to unintended and adverse effects on customers operations. Companies could avoid that risk by investing in rigorous and unambiguous specification - through use of formal methods. It is possible that the actual cost of performing maintenance could drop.

Another long term concern/cost that may be more applicable to formal methods than to other engineering approaches is that if tools were used in the development, the tools must be maintained in order to use the previous work as a starting point for changes. Since the formal methods tool market has hardly been established, much less wrung out, we cannot anticipate that the company that provided our tool will be around in 10 years, much less issuing updates on it for new hardware and software platforms. This puts the responsibility on the using organization to preserve the tool and its environment.

## 4 Conclusions

Use of formal methods in industry is still in the investigation stage and that fact must be recognized by any manager using them. The tools are not mature and a critical mass of users has not been created, much less an audience of informed consumers. As a result, education and resource management for both tools and people must have relatively larger allocations than in a normal development.

Formal methods appear to be most useful in the specification of systems, either during development or when re-specifying during a reverse engineering effort. The formal methods use should be deferred until there is a firm, intuitive understanding of the product and the user interface. The methods are designed to root out ambiguity, so they don't do well modeling "should" or "might" statements. Experts in the use of formal methods are essential on any effort; learning by doing is effective, but only when guidance is ultimately available. If experts aren't available, your people will get frustrated and your budget will get burned.

Formal methods are a tool to be used in an existing software process. If the organizational or project culture doesn't have a disciplined approach to software development, it is unlikely that formal methods will be successful.

The types of systems where formal methods appear to have the most success are in safety critical applications, such as large scale power management or flight and mission control systems. It is suspected that organizations that create the software for these tasks have the necessary supporting culture in place and have

a requirement to prove the reliability of their products, a constraint that few other development teams labor under.

Although much of this paper has been dedicated to warnings about the difficulties of using formal methods, many factors are making them more attractive: tools are getting more robust, the community is working toward making their output more accessible, governments and standards bodies are recognizing the value of formal methods, and systems are getting unimaginably complex.

Finally, one of the studies [44] noted that post-delivery failures were decreased on a formal methods program. Post-delivery failures are those that are not detected during unit or system test, so they might be problems in areas that were unspecified or were thought to be of little concern. This a common problem when developing a system for a new purpose; neither the developer or the customer can visualize all the consequences of the use of the new system, so sometimes the wrong things are emphasized or ignored. If formal methods can be shown to help stretch the imaginations of the users and developers during analysis, this might be a huge step forward for software development.

## References


1. Glenn Ammons, Rastislav Bodik, and James R Larus. Mining specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.
2. Mark R. Blackburn and Robert D. Busser. Requirements for industrial-strength formal methods tools. In *Workshop on Industrial Strength Formal Specification Techniques*, pages 137–8, Boca Raton, Fl., 1998. IEEE.
3. J. P. Bowen. Formal methods in safety-critical standards. In *1993 Software Engineering Standards Symposium*, pages 168–177. IEEE Computer Society Press, 1993.
4. J. P. Bowen. Ten commandments of formal methods. *IEEE Computer*, 28(4):56–63, 1995.
5. Jonathan Bowen and Victoria Stavridou. The industrial take-up of formal methods in safety critical and other areas: a perspective. In *FME'93: Industrial Strength Formla Methods*, volume First International Symposium of Formal Methods Europe Proceedings, pages 183–195. Springer-Verlag, 1993.
6. Jonathan Bowen and Victoria Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, 8:189–209, 1993.
7. J.P. Bowen and M.G. Hinchey. The use of industrial-strength formal methods. In *Twenty-First Annual International Computer Software and Applications Conference*, pages 332–7, Washington, DC, USA, 1997. IEEE Comput. Soc.
8. J.P Bowen and M. G. Hinchley. Seven more myths of formal methods. *IEEE Software*, 12(4):34–41, 1995.
9. Jean-Michel Bruel. Integrating formal and informal specification techniques. why? how? In *Proceedings of the 2nd IEEE Workshop on Industrial-Strength Formal Specification Techniques (WIFT'98)*, pages 50–57, Boca Raton, Florida, 1999. IEEE Computer Press.
10. A.. Butterfield. Introducing formal methods to existing processes. In *IEE Colloquium on Industrial Use of Formal Methods*, London, UK, 1997. IEE.
11. E Ciapessoni, E. Crivelli, and P Mirandola. From formal models to formally based methods: an industrial experience. *ACM Transactions on Software Engineering and Methodology*, 8(1):79–113, 1999.

12. G. Cleland and D. MacKenzie. Inhibiting factors, market structure and the industrial uptake of formal methods. In *Workshop on Industrial-Strength Formal Specification Techniques*, pages 46–60, Boca Raton, FL, USA, 1995. IEEE Comput. Soc. Press.
13. Tim Clement. Re dust expert. Private correspondence, 2002.
14. Software Productivity Consortium. Rtca do-178b, 2002.
15. Dan Craigen and Susan Gerhart. Formal methods reality check: Industrial usage. *IEEE Transactions on Software Engineering*, 21(2):90–98, 1995.
16. Dan Craigen, Susan Gerhart, and T. Ralston. An international survey of industrial applications of formal methods. Technical Report NIST GCR 93/626, US Dept of Commerce, 1993.
17. J. Dick and E. Woods. Lessons learned from rigorous system software development. *Information and Software Technology*, 39(8):551–60, 1997.
18. S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*, 24(1):4–14, 1998.
19. K. Finney and N. Fenton. Evaluating the effectiveness of z: the claims made about cics and where we go from here. *Journal of Systems and Software*, 35(3):209–16, 1996.
20. P. Garbett, J.P. kes, M. Shackleton, and S Anderson. Secure synthesis of code: a process improvement experiment. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - World Congress on Formal Methods in the Development of Computing Systems*, volume II, pages 1816–1835, Toulouse, France, 1999. Springer-Verlag.
21. Anthony Hall. Using formal methods to develop an atc information system. *IEEE Software*, 13(2):66–76, 1996.
22. Anthony Hall. What does industry need from formal specification techniques? In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques*, pages 2–7, Los Alamitos, Cal, 1999. IEEE.
23. Anthony Hall and Shari Lawrence Pleeger. Some metrics from a formal development. In *IEE Colloquium on 'Practical Application of Formal Methods'*, volume Digest No.1995/109, pages 6/1–4, London, UK, 1995. IEE.
24. C. Heitmeyer. On the need for practical formal methods. In A.P. Ravn and H. Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'98*, pages 18–26, Lygby, Denmark, 1998. Springer-Verlag.
25. C. Michael Holloway and Ricky Butler. Impediments to the industrial use of formal methods. *IEEE Computer*, 29(4):25–26, 1996.
26. Manuel J. Fernandez Iglesias, Fransisco J. Gonzalez-Castano, Jose M. Pousada Carballo, Martin Llamas Nistal, and Alberto Romero Feijoo. From complex specifications to a working prototype. a protocol engineering study. In J.N. Oliveira and P. Zave, editors, *FME 2001*, volume LNCS 2021, pages 436–448. Springer-Verlag, 2001.
27. Michael Jackson. Formal methods and traditional engineering. *Journal of Systems and Software*, 40:191–194, 1998.
28. L.J. Jagadeesan, P. Godefroid, J. Kelly, S. Miller, and Frank Weil. Transferring formal methods technology to industry. In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques*, pages 128–131. IEEE, 1998.
29. Sara Jones, David Till, and Ann M. Wrightson. Formal methods and requirements engineering: Challenges and synergies. *Journal of Systems and Software*, 40(3):263–73, 1998.

30. Shmuel Katz. Faithful translations among models and specifications. In J.N. Oliveira and P. Zave, editors, *FME 2001*, volume LNCS 2021, pages 419–434. Verlag-Springer, 2001.
31. Richard Kemmerer. Integrating formal methods into the development process. *IEEE Software*, 7(5):37–50, 1990.
32. Steve King, Jonathan Hammond, Robd Chapman, and Andy Pryor. Is proof more cost-effective than testing? *IEEE Transactions on Software Engineering*, 26(8):675–685, 2000.
33. John Knight, Colleen DeJong, Matthew Gibble, and Luis Nakan. Why are formal methods not used more widely? In *Langley Formal Methods Workshop*. NASA, 1997.
34. F. Koob, M. Ullmann, and S. Wittmann. Industrial usage of formal development methods-the vse-tool applied in pilot projects. In *COMPASS '96. Proceedings of the Eleventh Annual Conference on Computer Assurance Systems Integrity. Software Safety. Process Security*, pages 56–64, Gaithersburg, MD, 1996. IEEE.
35. Peter Gorm Larsen. Applying formal specification in industry. *IEEE Software*, 13(3):48–56, 1996.
36. Baudouin Le Charlier and Pierre Flener. Specifications are necessarily informal or: Some more myths of formal methods. *Journal of Systems and Software*, (40):275–296, 1998.
37. J. McDermid, A. Galloway, S. Burton, J. Clark, I. Toyn, N. Tracey, and S. Valentine. Towards industrially applicable formal methods: Three small steps, and one giant leap. In J. Staples, M.G. Hinchey, and S. Liu, editors, *Conference on Formal Engineering Methods*, pages 76–88, Brisbane, Qld., Australia, 1998. IEEE Comput. Soc.
38. David Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*, pages 279–287, Sorrento, Italy, 1994. IEEE Press.
39. M. C. Paulk, B. Curtis, E. Averill, J. Bamberger, T. Kasse, M. Konrad, J. Perdue, C. Weber, and J. Withey. Capability maturity model for software. Technical Report CMU/SEI-91-TR-24 ADA240603, Software Engineering Institute, 1991.
40. J.S. Pedersen. Introduction to formal methods and experiences from the lacos and orsted projects. In *IEE Colloquium on Industrial Use of Formal Methods*, volume Digest No: 1977/171, pages 2/1–2/3, London, UK, 1997.
41. Jakob Lyng Petersen. Automatic verification of railway interlocking systems: a case study. In *FMSP 98*, pages 1–6, Clearwater, Fl, USA, 1998. ACM.
42. A. Puccetti. Improving the software evolution process using mixed specification techniques. Technical Report 27492 ESSI - ISEPUMS, Esprit, 2000.
43. G. Romanski. The challenges of software certification. *CrossTalk*, Sep 2001 2001. Available at Software Technology Support Center website:.
44. C. Snook and R. Harrison. Practitioners' views on the use of formal methods: an industrial survey by structured interview. *Information and Software Technology*, 43(4):275–83, 2001.
45. Allen M Stavely. Integrating Z and cleanroom. In *Proceedings of the Fifth Annual Langley Formal Methods Workshop*, Langley, Va, 2000. NASA.
46. Frank Weil. Wift '98 working group report: Incorporating formal methods onto industrial process. In *Workshop on Industrial Strength Formal Methods '98*, pages 134–6, Boca Raton, Fla, USA, 1998. IEEE Computer Society.
47. R. Wieringa and E Dubois. Integrating semi-formal and formal software specification techniques. *Information Systems*, 23(3-4):159–78, 1998.

48. A. Wong and M. Chechik. Formal modeling in a commercial setting: a case study. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - World Congress on Formal Methods in the Development of Computing Systems.*, volume I of *Lecture Notes in Computer Science*, pages 590–607, Toulouse, France, 1999. Springer-Verlag.