Formal Methods Adoption: What's working, What's not!

Dan Craigen

ORA Canada
1208 ONE Nicholas
Ottawa, Ontario, K1N 7B7
CANADA
dan@ora.on.ca
WWW home page: http://www.ora.on.ca

Abstract. Drawing from the author's twenty years of experience in formal methods research and development, and, particularly, with the EVES-based systems, this paper provides personal impressions on *what is* and *what is not* working with regards to the adoption and application of formal methods.
As both the community's understanding of technology transfer issues and formal methods technology improve, one is optimistic that formal methods will play an increasingly important role in industry. However, significant impediments continue to exist with, perhaps, the increasing complexity of systems being both a blessing and a curse.

1 Introduction

Drawing upon my twenty years of experience in formal methods R&D, I will discuss personal impressions on *what is* and *what is not* working with regards to the adoption and application of formal methods.[1] I will structure this paper primarily around a narrative of the history of my group's work on formal methods.[2] Some conclusions are drawn in 20-20 hindsight. I will generally eschew technical discussions of our own technology. The citations provide appropriate pointers and many of our technical reports are available through our web site.

2 Genesis: m-EVES

Our work on formal methods grew out of a mid- to late-70s DARPA motivated effort in developing a verifiable programming language, ultimately named Euclid.

[1] I am writing this paper as we introduce our newborn daughter (Cailin) into our family (which also includes Liz and Ailsa). Sleep deprivation and various distractions have taken their toll on what I had hoped to include herein.

[2] Currently, the formal methods group at ORA Canada consists of myself, Sentot Kromodimoeljo, Irwin Meisels and Mark Saaltink. Bill Pase (automated deduction) and Karen Summerskill (technical editor) are instrumental past members of the group.

Euclid was a Pascal derivative which was provided a Hoare-style axiomatization, extended Pascal in various ways to support larger-scale system programming, and even had a compiler written for much of the language. Initially, our R&D plans were to develop a Euclid Verification and Evaluation System (hence, the acronym, EVES[3]). However, circumstances would dictate otherwise.

In the early 80s, Bill Pase and the author were sponsored to review and assess U.S. efforts in developing program verification systems (Affirm, the Gypsy Verification Environment, the Stanford Pascal Verifier, and HDM [the Hierarchical Design Methodology]).[4] This was a great opportunity to come to terms with the then cutting-edge of technology in various disciplines, including automated deduction, software engineering and language design. One of our observations resulting from the survey and subsequent experimentation, was that the mathematical underpinnings of the program verification systems were suspect (actually, generally unsound). To be fair, however, it must be noted that the prime goals of the researchers were to show that such verification systems could be developed and applied. Consequently, it was prudent to ask whether these systems could or should be used to reason about security- or safety-critical applications. We felt that appropriate mathematical foundations were necessary.

As the survey came to conclusion, sponsorship for developing EVES was obtained. The initial intent was to use results from the U.S. program verification efforts (including the possible use of system components) and to enhance Euclid (resulting in a language called Ottawa Euclid) with a formal specification framework. To cut a long story short, we ended up rejecting Euclid (and Ottawa Euclid) because of language and logic pathologies and the inimical nature of the pathologies towards our goal of reasoning about security-critical applications. Furthermore, we were advised by our colleagues in the automated deduction community that it would be best to develop our own automated deduction facility (ultimately called Never). With these decisions, we launched into language design and prover development. A somewhat risk-adverse approach was taken in that we were to develop a proof-of-concept system for a reasonably simple programming language. Consequently, we developed the m-EVES system [1, 2] with its programming (somewhat akin to a cleaned up Pascal with libraries) and specification (essentially, first order type theory) language (m-Verdi) and prover (m-Never). As part of the language design, we developed the underlying semantics and proof theory. m-Never was a nifty integration and enhancement of ideas drawn from the Stanford Pascal Verifier, Affirm, Gypsy, and the Boyer-Moore prover). Subsequently, we demonstrated the use of m-EVES on (somewhat large) toy examples.

In retrospect, there were a number of observations and lessons that came out of the m-EVES exercise:

[3] Though we continue to use the EVES acronym, (for example, in Z/EVES) we have long ago jettisoned the underlying interpretation.

[4] From a technology transfer perspective, it is interesting to note that Bill and I became conduits for information flow between the research groups which, otherwise, had been working independently of each other.

Technology driven: The project was technology driven. We focused on engineering and R&D issues as it pertained to a proof-of-concept comprehensive and sound verification environment. Except for the consideration that our technology would be used for critical systems, we gave no real consideration of how clients might apply the technology. Be customer-centric. However, be wary, for sometimes the customer does not have it right!

Puritanical: As the overall project goals were to support the development of security- (later safety-) critical systems, we were serious about the mathematical underpinnings for the system. This has both positive and negative consequences, which were accentuated with EVES and are discussed in the next section. We were religious and inflexible in our certitude.

Controlled distribution: In the mid-80s, many North American systems were tightly controlled as to accessibility. This had the deleterious effect of constraining the development of user communities and of independent feedback.

Technology transfer: We gave only limited consideration of actual transfer of the technology. While we embodied what we viewed as good software development principles, we gave no real consideration to commercialization of m-EVES nor of how the technology might be adopted by other organizations. A few other organizations did successfully use the technology, but these generally were early adopters following technical interests. Be prepared to take the *whole product* perspective. A substantial amount of boring stuff needs to be developed to really provide a useful product.

The completion of m-EVES was a significant technical milestone and it led to the almost immediate effort in developing EVES. In retrospect, we should have delayed moving onwards with EVES until we had given m-EVES more substantial trials and considered its linkage into our sponsors' organizations.

3 Let's do it again!

In the mid-80s we got to do it all over again, both with respect to surveying technology and moving forwards with the EVES development.

3.1 Verification Assessment Study

Firstly, of some note, was the Verification Assessment Study [3] led by Dick Kemmerer (University of California at Santa Barbara). This study brought together representatives of Enhanced HDM (Karl Levitt, then at SRI International), the Gypsy Verification Environment (Don Good, then at UT-Austin), Affirm (Dave Musser, then at GE Schenectady) and FDM (Debbie Cooper, then at SDC, Santa Monica), along with myself as an independent participant. The primary idea for the Study was to allow for the sharing of technical details between the participants and help form the basis for future development. This experience allowed me to update my knowledge of these systems and provided some input into the ongoing EVES development.

Of some note, is the then view of what a mid-80s state-of-the-art verification system would look like, if only to consider how we have progressed since then. The VERkshop III proceedings [4] identified such a system as consisting of:

- a specification language based on first-order, typed predicate calculus.
- an imperative language derived from the Pascal/Algol 60 family of programming languages.
- a formal semantic characterization of the specification and implementation languages.
- a verification condition generator (itself a major research challenge of the mid-70s).
- a mechanical proof checker with some automated proof generation capabilities.
- a (small) supporting library of (reusable) theorems.
- a glass (as opposed to hard copy) user interface, possibly using bit mapped displays.
- a single system dedicated to one user at a time.
- the embedding of these components in a modest programming environment.

As it turned out, our work with EVES was aiming to achieve at least these requirements.

Technology transfer was discussed during the study and efforts were made by the sponsors to make the verification systems available for use on a Multics system through the ARPAnet. However, there was still contention between those who felt a more open distribution of technology was appropriate and those who did not. As noted above, the controlled distribution of many systems impeded adoption.

3.2 EVES

With the m-EVES proof of a concept deemed a success (at least in our minds and those of our sponsors), we moved forward (in the mid-80s) with the *production* EVES system [5, 6]. With EVES, we would not be making some of the simplifying assumptions of m-EVES. However, a key continuing tenet to our work was that EVES would be used for the specification and development of critical applications and, consequently, sound mathematical foundations were crucial.

Our first important decision was which mathematical framework to use. The expressiveness we wanted could be found in either untyped set theory or (our initial bias) higher-order logic. We chose set theory since, in part, we felt it would be an easier adoption channel and had better automated deduction support.

EVES was a technical success. We generally achieved all the goals that we set out for. We developed a language (Verdi) that supported the expression of general mathematical theories, specification of programs (pre/post form), sequential imperative constructs, and a reasonable library mechanism. The prover brought together ideas from across the automated deduction domain (Boyer-Moore style heuristics in a richer logic, decision procedures, good mix of automation and

user direction). A solid mathematical basis in which the language semantics were described and the proof theory shown to be sound. The development of an independent proof checker mechanism that validated the proofs found by the prover.[5] The validation of putative proofs is an important requirement for critical applications. And a rigorously developed compiler, in which key passes of the compiler were rigorously specified and traceability from the compiler source to the compiler specification demonstrated. EVES has been successfully used (by both ourselves and external groups) on a number of applications including fielded safety- and security-critical applications.[6]

However, being able to demonstrate a successful project from a technical perspective is different from actually having the technology transition into general use. It must be admitted that EVES has not been successfully transfered. I believe there are a number of reasons; many of which generalize to other efforts in formal methods.

Language: Not many people (especially within industrial contexts) wish to learn yet another specification or programming language. Verdi was spare with respect to its notation (basically akin to Lisp s-expressions). Also, the programming component was restrictive. One interesting lesson was that our design was not usually restricted by formal semantics concerns, but by logic constraints. For example, we excluded global state because of the difficulties resulting in the Verdi inference rules. The typeless basis also seemed to be an impediment. The equivalent of type reasoning is usually a part of the proof effort. This meant that feedback on simple specification infelicities came a bit later than necessary. Finally, I generally concluded that language design was more trouble than it was worth. No matter what decisions one made, there were always critics!

Industry will consider adopting new languages if there are significant commercial reasons for doing so. The attributes of formal semantics and (support for mechanized) proof are very low on the priority list and languages designed with these latter attributes as their main characteristic just will not transfer.

Ambitious: We were highly ambitious in our language design goals and for the overall system. *I know of no system or research effort that has cast as wide a net as the EVES project.* By choosing such ambitious goals, we, in effect, delayed our "coming out party." It also meant that we were subject to derogation. By the 90s, if not earlier, the concept of "program verification" had engendered a pejorative tone. Though EVES supported the expression

[5] Hence, we separated issues pertaining to proof discovery (using the prover) and to proof validity. For proof discovery, we were free to use any mechanisms (including extra-logical mechanism) to help find proofs. The use of integer linear programming techniques in the decisions procedures is one example. The (comparatively) smaller proof checker was an embodiment of the EVES logic proof rules, was carefully scrutinized, and is reasonably reviewable by independent parties.

[6] Numerous technical reports on all these aspects of EVES are available at our web site.

and analysis of general mathematical concepts, competitors would sometimes deride our efforts as purely "program verification."

For ambitious projects, attempt to structure them in such a manner that useful products and results can be released throughout the project time frame.

Complexity: EVES is a complex system, as are most serious verification systems. The complexities of formal specification, heuristic proof, etc., make it a difficult system to come to terms with.

Industry does not need to add complex development and analytical technologies atop of already complex processes and products. While education will, in the long term, help the insertion of formal methods technology, crafty integration and hiding of complex technologies is likely to be more successful.

Isolated: As with most verification systems, EVES was a stand-alone system. This isolation (both physical and conceptually) makes it difficult to integrate into industrial processes.

Integrate, integrate, integrate! We have to bite the bullet and accept the realities of current hardware and software development processes. Aim to improve and evolve current processes. It is highly unlikely that we will succeed with discontinuous innovations, even though this is a hallmark trait of high-technology.

Publication: We did not actively publish our work. A substantial amount of quality R&D was performed and published in technical reports. However, for various reasons, we did not actively publish (publicize!) our efforts. Consequently, the broader community was not necessarily aware of our achievements.

Not open: EVES was a closed system. We gave no allowance for users to add functionality (except through the addition of reusable theories). This limited certain R&D opportunities.

Where possible, we need to open our systems; perhaps to the point of distributing open source software. At the very least, we need to define system APIs so as to support the integration process.

Distribution impediments: Though substantially eased by the early- to mid-90s, there were still some distribution impediments early on.

And we still see it today, where governmental security concerns are being replaced by corporate priority and market advantage issues.

4 Technology Transfer

4.1 Survey of Industrial Applications

In the early 90s we started to come to terms with technology transfer. In 1992/3, Susan Gerhart (then at Applied Formal Methods), Ted Ralston (then at Ralston Research Associates), and I embarked on what became a fairly influential survey of industrial applications of formal methods [7–9]. In this survey we looked at

twelve formal methods projects in North America and Europe.[7] The objectives of the study were threefold:

- To better inform deliberations within industry and government on standards and regulations.
- To provide an authoritative record on the practical experience of formal methods to date.
- To suggest areas where future research and technology development are needed.

I will not rehash the conclusions of the survey, but one of the consequences of the effort was to apply an "innovation diffusion" model to the survey data [9]. As I believe the analytic framework is useful for understanding formal methods adoption, I will take a few moments to discuss the framework. Formal methods researchers would do worse than to consider their adoption trajectories in terms of these criteria (and of Chasm Model, discussed later). As of the mid-90s, we concluded that formal methods advocates were facing significant, though not unique, impediments in diffusing their technology to industry. The innovation diffusion model provides a framework for understanding the forces advancing and/or retarding the diffusion of formal methods. The framework is based on work by Rogers [10] and others.

As discussed in [9], we identified the following criteria:

Relative advantage: An analysis of the technical and business superiority of the innovation over technology it might replace. Until recently, formal methods did not compellingly pass the economic profitability test. However, especially with the use of model checking in Hardware verification, the balance is shifting. Achieving reliability levels through testing solely is running into problems with complexity and size.

The lack of a compelling economic argument often relegated formal methods to (a perceived ghetto of) security- and safety-critical systems, where the cost of system failure is inordinate. But, even here, adoption was minimal.

Compatibility: An analysis of how well the innovation meshes with existing practice. Typically, formal methods systems have not meshed well with existing practice. However, improved recognition of this issue is resulting in the insertion of formal methods-based technology into tools that are actually used by developers.

It is important to note that compatibility is not only at the tool level, but at a social level, in that the proposed innovations must fit well with *community values*. Regulatory environments can be particularly difficult to penetrate if only for the conservative view of change.

Complexity: An analysis of how easy the innovation is to use and understand. For large scale adoption, we have to hide some of the complexity of formal methods. Complexity can be mitigated by aggressive educational programs, push-button tools, and a formal methods taxonomy.

[7] A clear oversight of the survey was its lack of projects using model checking.

Trialability: An analysis of the type, scope and duration of feasibility experiments and pilot projects.

Observability: An analysis of how easily and widely the results and benefits of the innovation are communicated.

Transferability: An analysis of the economic, psychological and sociological factors influencing adoption and achieving critical mass. Principal factors are:

Prior technology drag. The presence of large and mature installed bases for prior technologies.

Irreversible investments. The cost of investing in the new technology.

Sponsorship. The presence of an individual or organization that promotes the new technology. Promotion includes subsidization of early adopters and the setting of standards.

Expectations. The benefits accruing from expectations that the technology will be extensively adopted.

Adoption of a superior innovation may still be negated by concerns, on the part of early adopters, arising from two early adoption risks: "transient incompatibility" and "risks of stranding." Transient incompatibility refers to risks accruing from delays in achieving a sufficiently large community interested in the new innovation. Stranding refers to the risks arising due to the failure to achieve critical mass.

We applied the above criteria to our survey data and concluded at that time, correctly I believe, that formal methods have a low adoption trajectory. See [9] for a detailed discussion of applying this framework to formal methods.

It is important to note that a low adoption trajectory is not necessarily inherent to formal methods. Changing perspectives on how to use the technology effectively, the development of new capabilities and the creation of new markets may all result in enhanced adoption for key technical facets.

4.2 Adopting Z

In addition to the survey, we were also looking for means to enhance the adoption rate of our EVES technology. This search was manifested by our efforts, for example, to remedy the syntax (resulting in s-Verdi) and to port EVES to Windows. However, the main manifestation was in our decision to link our EVES technology with Z, resulting in Z/EVES [11–13]. *This meshing of technologies has been particularly effective with Z/EVES now distributed to over* forty *countries.*

When we viewed the Z world we found a world in which there were an extensive number of consumers, a reasonable pedagogical literature, an incomplete language semantic definition and, generally, lousy tool support. If one negates

the previous four conjuncts, one had a reasonable description of the EVES world. A union seemed to make sense! Furthermore, sponsorship was available to pursue the work. Though we were certain that technical difficulties would arise in linking Z with EVES, we were also certain of the benefits. We also noted that many of our North American colleagues derided Z as a language for which mechanical reasoning was impossible and showed absolutely no interest for tapping into a broader community. It is true that Z presents some challenges to mechanized proof support, but these are offset by Z's facilities for the concise and clear specifications. *Z/EVES shows that powerful proof support can, indeed, be provided for Z.* One does not necessarily have to design new languages that, arguably, have been designed for mechanized proof. Such efforts, while useful R&D, will not succeed in transitioning to a broad user community.

Z/EVES integrates a leading specification notation with a leading automated deduction capability. Z/EVES supports the entire Z notation. The Z/EVES prover provides powerful automated support (*e.g.*, conditional rewriting, heuristics, decision procedures) with user commands for directing the prover (*e.g.*, instantiate a specific variable, introduce a lemma, use a function definition). We have automated much of the Z Mathematical Toolkit and include this extended version with the Z/EVES release.

Z/EVES supports the analysis of Z specifications in several ways:

- syntax and type checking,
- schema expansion,
- precondition calculation,
- domain checking (*i.e.*, Are functions applied only on their domains?),
- refinement proofs, and
- general theorem proving.

The range of analysis supports an incremental adoption of Z/EVES capabilities. For example, very little knowledge of the theorem prover is required for syntax and type checking, and schema expansion. Even with domain checking, many of the proof obligations are easily proven; and for those that are not, often the generation of the proof obligation is a substantial aid in determining whether a meaningful specification has been written. It has been our experience that almost all Z specifications are materially improved through syntax and type checking conjoined with domain checking (even if only performed informally). Consequently, for very little effort on the part of the Z/EVES user, material returns accrue from analyses that fall short of the full use of the analytical capabilities of Z/EVES.

Z/EVES accepts its input in the markup format used by the LaTeX system and by Spivey's "fuzz" typechecker. Additionally, the Z/EVES interface is basically Emacs (or a Windows-related clone). While the use of LaTeX and Emacs are fine within the research community, both are impediments to a broader use by industry and by a next generation brought up in a PC-centric, Microsoft dominated world. John Knight's group at the University of Virginia have been working on a Framemaker-based tool for developing specifications in Z called

Zeus [14, 15]. Z specifications are written using all the publishing facilities of Framemaker, uses the Z character set, and interacts with Z/EVES via a graphical user interface. We are also focusing on GUI and word processor integration issues [16, 17]. A portable GUI for Z/EVES is due to be completed by the Fall of 1999. As part of this work, we have chosen XML as our markup language for Z and as the main means of communication between tools. We have opened up Z/EVES by formally specifying (in Z) the Z/EVES API [16]. This will allow others to integrate with Z/EVES in a seamless manner. We are also considering further how Z/EVES can be linked with Word [17]. A necessary evil!

Both our work on a Z/EVES GUI and the University of Virginia work on Zeus were discussed at the Z/EVES session of the FM'99 Z Users' Group meeting. We expect that these evolutionary changes to Z/EVES will further enhance its adoption trajectory. A current significant use of Z/EVES is in undergraduate and graduate courses. We expect that Z/EVES will play a significant role in formal methods training.

5 Applications

Over the years we have had a number of opportunities to use formal methods technology (primarily as embodied in EVES or Z/EVES). Some of these efforts have been clear technical successes. Others, a bit more marginal.

Recently, we have been analyzing authentication protocols and open source public key infrastructures (PKI). These efforts have been great learning experiences and helped to reinforce certain biases. While our work is generally hidden behind a veil of proprietary issues, some general observations can be reported.

Modeling and Analysis: Mathematically modeling key attributes of complex systems and analyzing such attributes can be highly cost effective. By focusing our analysis on certain design and implementation aspects of the PKI, we found significant infelicities. It's worth noting that modeling and analysis can be used in both a forward and reverse engineering manner. Either way, significant benefits can accrue. Obviously, one would prefer to engineer formal modeling and analysis into the development of systems; however, an industrial reality is that there is a huge base of legacy software that is becoming increasing difficult to manage and predict.

Complexity: The complexity of today's software and hardware artifacts is profound and, often, is a result of ill-discipline and reaction to market forces. (The mindset appears to be one along the lines of *best to get a product out with capability X, even though the implementation and design of X is suspect. There's always time to fix it later!*) My general view is that while formal methods has made outstanding progress over the last decade (or so), we are falling further behind industrial practice because of the rapid advances of hardware and software system requirements. At best, we must carefully slice out key aspects of such systems for modeling and analysis.

Open Source Software: Arguments have been made about the potential for achieving high reliability levels with open source software. Our review of

important attributes of the PKI indicated important infelicities at both the conceptual and implementation levels. While there are definite benefits to the open source software movement, one must be very careful in attributing high reliability to such software. Many individuals involved with using and testing such software will have only fleeting commitments. I'm unsure that open source is a valid path to high reliability.

Limitations of Model Checking: In our work with PKI and authentication protocols we have used a number of tools in addition to EVES and Z/EVES. For example, we experimented with Murϕ, SMV, FDR2 and Dymna (a nice tool which identifies potential masquerade attacks in authentication protocols). The enhanced automation of the model checking tools certainly helped, but, in many cases, still required substantial setup. In fact, especially with Murϕ and SMV, I felt that I was in the midst of a programming exercise; in effect, writing abstract executable specifications. The latter statement is not necessarily a pejorative comment; such a view may aid adoption. But, what surprised us, in addition to the substantial setup, was that there were extreme difficulties in controlling state space explosion. In fact, we knew of one significant infelicity in the PKI that we tried to find using Murϕ. No matter how much we progressively abstracted our specification and pointed Murϕ to the problem area, we couldn't get the counterexample shown. There's obviously substantial R&D required in understanding how to abstract systems effectively, how to partition and/or control state space explosions, and on how to integrate (either through loose or tight coupling) the various modeling and analysis engines.

Benefits of Automation: Automated analysis of models has the significant potential benefits of generating accurate proofs (or calculations), extending our capabilities to complex artifacts that otherwise would be unmanageable, and doing so rapidly. Model checking has shown great promise in these regards. Theorem provers, much less so (because of the richer properties normally tackled).

Legacy Code: Formal methods can be highly effective in understanding legacy code. (I guess IBM learned this quite early with their use of Z to understand components of CICS.) One must be careful in selecting the important aspects of such systems, but there is a huge potential market here.

We are now much less puritanical in our use of formal methods; we are now making much better use of engineering judgment in choosing where and how to apply formal methods. In fact, in many respects, the perspective has changed from one in which one aims for total correctness and, instead, aims at exposing infelicities. It's almost become "design and formally debug," rather than "design and verify."

For years, our project resources were directed at evolving our technology base, *not* on applying the technology on artifacts of interest to the broader community within which our sponsorship resided. While such an approach has its benefits, it also has significant costs in that we were not able to demonstrate the utility of the technology to mission-oriented criteria, nor were we able to

validate the applicability of the technology. We did not have the opportunity to evolve our work towards the realities of the sponsor's marketplace. There are substantial benefits that accrue from applying new technology on true industrial scale problems.

6 The Chasm

Geoffrey Moore's book *Crossing the Chasm* [18] focuses on high technology marketing. However, much of what he has to say relates to technology transfer and sets a useful model for formal methods adoption. While I cannot do full justice to his treatise, I feel it is worthwhile noting a few points, for they provide some indications on how formal methods must adapt to become truly successful.

Moore's view of the Technology Adoption Life Cycle is primarily based on the identification of five groups of potential adopters: Innovators, Early Adopters, Early Majority, Late Majority and Laggards. Understanding the differences and mindsets of these adopters is crucial for successful technology transfer.

Innovators: These folks pursue new technology products aggressively. Technology is the central purpose of their lives. One must win this group over as their endorsement of a new technology reassures other potential adopters.

Early Adopters: These folks are not techies, but individuals who find it easy to appreciate the benefits of a new technology and to relate the benefits to their concerns. They do not rely on well-established references in making buying decisions, preferring their intuition and vision. Early adopters are key to opening any high-tech market segment.

Early Majority: The early majority share the early adopter's ability to relate to technology, but are driven by a strong sense of practicality. They want to see well-established references before investing substantially. It is with the early majority the the opportunities for significant profits start to appear!

Late Majority: The late majority has the same concerns as the early majority plus one major additional fact: they are uncomfortable with technology products. They want a well-established standard, lots of support, and buy from large and well-established companies. If one succeeds in winning this group, profits can be substantial as most R&D costs have been amortized.

Laggards: Laggards don't want anything to do with new technology. It has to be buried deep inside another product.

Moore goes on to claim that the groups of adopters form a bell curve for technology adoption, except that there are gaps: between innovators and early adopters; between the early majority and the late majority; and, most significantly, between the early adopters and the early majority. In Moore's view, the last gap is, in fact, a chasm and therein lies the most significant danger for technology adoption. Early adopters are looking for a change agent. The early majority wants a productivity improvement for existing operations.

To cross the chasm, one must target a market segment defined around a *must-have* value proposition. There are three sources of a must-have condition:

- It enables a previously unavailable strategic capability that provides a dramatic competitive advantage in an area of prime operational focus.
- It radically improves productivity on an already well-understood critical success factor.
- It visibly, verifiably, and significantly reduces current total overall operating costs.

Basically, one must be able to argue that the new technology will radically improve productivity on an already well-understood critical success factor specific to the organization being targeted. Furthermore, there is no existing means by which comparable results can be achieved. Exercise for the reader: How does your technology meet these criteria?

From this perspective, one concludes that we chose some rather unfortunate value propositions for EVES. Perhaps the most unfortunate value proposition was that of mathematical soundness. Soundness (to a certain extent) is not a marketable proposition. Our efforts on code verification were also likely misplaced.

7 Discussion

Though our experiences and those of the general formal methods community has been mixed, I continue to believe that the technology will play a crucial role in the future.

One reason for my optimism is that we are increasing understanding key technology transfer issues. Models such as those of Rogers and Moore provide important insights. Moore's model seems to suggest that, on the most part, formal methods is still at the innovator or early adopter stage. Though, one can point to apparent successes at, for example, Siemens and Intel, suggesting incremental movement towards the early majority.[8] In both cases, it appears that the limitations of traditional validation processes (mainly simulation) were not up to the task for achieving the requisite reliability requirements. Reducing Q&A effort, wherein lies much of the development time, is the critical business requirement. Model checking and equivalence checking appear to have suitable business cases. Theorem proving does not (at least according to one senior Siemens individual)!

There have been failures at the innovator and early adopter stage as well, especially in the security-critical area. These failures are due to complex reasons, but amongst the reasons are (i) the lack of clear mission-oriented goals (rather than technical driven projects), (ii) intransigence and inflexibility on the part of formal methods researchers who refused to react positively to the real needs of their sponsors, (iii) the lack of research focus and clear evaluation criteria, and

[8] The perspective taken here is that the market for adoption is within Siemens, Intel or the like; not broad industrial acceptance. At Siemens and Intel, it appears that general development groups are using formal methods (as embodied in model checking).

(iv) the lack of a clear winning observable "product." In some organizations, a harsh "formal methods winter" has set in as current management view the lack of return on significant investments.

Positive results can come from surprising corners. One of the main achievements of NQTHM is the use of executable specifications to model hardware chips, sometimes at a speed that is faster than traditional hardware simulation tools.

We are also learning how to integrate our technology in a manner that potentially enhances existing engineering practices. For example, Prover Technology's integration of NP-Tools with STERNOL allowed for the automatic verification of safety properties of computerized railway interlocking systems. The actual analysis engine was completely hidden from the engineers, they had only to continue to use their own language and push the analysis buttons at the right time. According to Prover Technology, system verification was reduced by 90%. Numerous other current and planned efforts are looking at this type of integration (say, with more widely disseminated languages such as VHDL and UML).

The prime formal methods business case, at this point in time, is to enhance Q&A; to complement current testing resources. For those areas in which mechanized formal analysis can be effectively used, much of the limited testing resources can be reallocated to other parts of the assurance process. Formal modeling has obvious benefits, especially with regards to requirements elicitation and clarification. But, we are up against major challenges with the increasing complexity of systems and the changes of paradigm. Java is quite different from Algol 60. Heterogeneous distributed systems quite different from single processor systems. Out-of-order and speculative execution. Dynamic compilers. And so on.

8 Acknowledgements

My thanks to Mark Saaltink for his review of an earlier draft of this paper.

References

1. Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Andy Nielson, Bill Pase and Mark Saaltink. m-EVES: A Tool for Verifying Software. In *Proceedings of the 11th International Conference on Software Engineering (ICSE'11)*, Singapore, April 1988.
2. Dan Craigen. An Application of the m-EVES Verification System. In *Proceedings of the 2nd Workshop on Software Testing, Verification and Analysis*, Banff, Alberta, July 1988.
3. Richard Kemmerer. Verification Assessment Study: Final Report. Volumes I-V, National Computer Security Center, 1986.
4. Proceedings of VERkshop III: A Formal Verification Workshop, Pajaro Dunes Conference Center, Watsonville, California, February 1985. Software Engineering Notes, Volume 10, Number 4, August 1985.
5. Dan Craigen, Sentot Kromodimoeljo, Irwin Meisels, Bill Pase and Mark Saaltink. EVES: An Overview. In *Proceedings of VDM'91: Formal Software Development Methods*. Volume 551, Lecture Notes in Computer Science, Springer-Verlag 1991.

6. Dan Craigen and Mark Saaltink. Simple Type Theory in EVES. In *Proceedings of the Fourth Banff Higher Order Workshop*. Graham Birtwistle (editor). Springer-Verlag, 1990.
7. Dan Craigen, Susan Gerhart and Ted Ralston. An International Survey of Industrial Applications of Formal Methods. NIST GCR 93/626 (Volumes 1 and 2), U.S. National Institute of Standards and Technology, March 1993. Also published by the U.S. Naval Research Laboratory (Formal Report 5546-93-9582, September 1993), and the Atomic Energy Control Board of Canada, January 1995.
8. Susan Gerhart, Dan Craigen and Ted Ralston. Observations on Industrial Practice Using Formal Methods. In *Proceedings of the 15th International Conference on Software Engineering (ICSE'15)*. Baltimore, Maryland, May 1993.
9. Dan Craigen, Susan Gerhart and Ted Ralston. Formal Methods Technology Transfer: Impediments and Innovation. In *Applications of Formal Methods*, M.G. Hinchey and J.P. Bowen (editors). Prentice-Hall International Series in Computer Science, September 1995.
10. Everett Rogers. Diffusion of Innovations. Free Press, New York 1983.
11. Mark Saaltink. The Z/EVES System. In *ZUM'97: The Z Formal Specification Notation (10th International Conference of Z Users*. Bowen, Hinchey, Till (editors). Lecture Notes in Computer Science, Volume 1212, Springer-Verlag.
12. Mark Saaltink and Irwin Meisels. The Z/EVES Reference Manual. ORA Canada Technical Report TR-97-5493-03d, December 1995.
13. Mark Saaltink. Domain Checking Z Specifications. Presented at the 4th NASA LaRC Formal Methods Workshop, Langley, Virginia, September 1997. ORA Canada conference paper CP-97-6018-65.
14. John Knight, Thomas Fletcher and Brian Hicks. Tools Support for Production Use of Formal Methods. Discussion paper in Proceedings of *FM'99: World Congress on Formal Methods.*
15. University of Virginia Software Engineering Group. Zeus: A Comprehensive Tool for Developing Formal Specifications. Version 1.3, March 1999.
16. Mark Saaltink, Sentot Kromodimoeljo and Irwin Meisels. The Z/EVES GUI Design and API. To appear.
17. Mark Saaltink. Integration of Z/EVES with Word Processors. To appear.
18. Geoffrey A. Moore. Crossing the Chasm. Harper Business, 1991.