Technology Transfer Challenges for High-Assurance Software Engineering Tools[1]

Position Paper for *Software Engineering for High Assurance Systems: Synergies between Process, Product, and Profiling (SEHAS 2003)*

John Penix
NASA Ames Research Center
John.J.Penix@nasa.gov

Lawrence Z. Markosian
QSS Group, Inc.
lzmarkosian@email.arc.nasa.go

Abstract

In this paper, we describe our experience with the challenges that we are currently facing in our effort to develop advanced software verification and validation tools. We categorize these challenges into several areas: cost benefits modeling, tool usability, customer application domain, and organizational issues. We provide examples of challenges in each area and identify open research issues in areas which limit our ability to transfer high-assurance software engineering tools into practice.

1. Introduction

Over the past decade, NASA has been moving away from the attitude of "get it right at any cost" which was perhaps epitomized in the Space Shuttle and Space Station programs. The same might be said of high-assurance software engineering: gone are the days of critical software "kernels" which can be completely verified "at any cost". Software is now pervasive in critical systems and this critical software is pervasive in our lives. At NASA this means that software concerns and software costs are primary factors in almost every part of a mission lifecycle.

Due to the limited resources and risk margins for NASA missions, the adoption of new software engineering technologies requires a strong business case that clearly demonstrates the cost and benefits of the new technology. However, in performing this type of cost benefits analysis, we face a variety of challenges. For example, conducting realistic technology validation studies for new software engineering techniques is extremely expensive. In addition, due to the number of variables controlling who uses the tools and exactly how they are applied, the results of such studies are usually statistically invalid and, at best, unconvincing when generalized.

In this paper, we describe our experience in developing and evaluating advanced software verification tools for high-assurance applications. We then describe specific challenges we face in the areas of cost benefits modeling, tool usability, customer application domains and organizational issues. Along the way, we identify open issues and opportunities which we think will require coordinated research efforts within the community to address.

2. Experience

Our specific experience has been in the context of developing source code analysis tools based on model checking and static analysis technologies. Model checking is an automated verification technique for finite-state systems that has had significant main-stream success in the areas of microprocessor and protocol verification. Model checking algorithms can be used to verify that a finite-state model of a system is free from subtle concurrency errors such as deadlocks, and that specified unsafe states or sequences of states are unreachable.

The Automated Software Engineering Group at NASA Ames has been extending model checking technology to support program verification and validation. Our research has provided evidence that model checking can increase the level of assurance for autonomous control architectures and real-time operating systems. For example, in 1997, we constructed a model of the resource management architecture from the Remote Agent Executive for Deep Space 1. We were able to use model checking to automatically identify several concurrency bugs in the software that had not been discovered during

[1] The research described in this report was performed at NASA Ames Research Center's Automated Software Engineering group and is funded by NASA's Engineering for Complex Systems program.

testing [4][5]. A nearly identical bug arose (in a different part of the software) during flight, causing a control system deadlock. A follow-up study indicated that automated translation and abstraction tools would have reduced the cost of applying model checking and allowed this bug to have been detected prior to flight [3].

We also have an ongoing collaboration with the Honeywell Technology Center to apply model checking to the C++ implementation of a multi-threaded operating system for integrated modular avionics. This case study has been successful in guiding the development of abstraction techniques, enabling exhaustive verification of the core scheduling algorithm [6]. There have also been a number of investigations within NASA showing that model checking can be effective in early phases of the lifecycle, during the generation of test cases from specifications, as well as during Independent Verification and Validation activities [1][2][7].

3. Challenges

While model checking technology and tools have been progressing, there are still significant challenges to the transfer of the technology into NASA missions. Perhaps the most daunting challenge (and the one most pertinent to this workshop) is the development of cost/benefits or return on investment models for our tools.

In addition, we continue to encounter challenges in the areas of tool usability, characteristics of the customer's application domain, and various organizational issues. In the following sections, we provide examples of challenges in each of these areas. We also identify what we believe to be the critical open research issues that must be addressed to move tools into practice more effectively.

3.1 Models and metrics

Tools must be integrated into effective processes for facilitate technology adoption. We believe that metrics and models should be used to guide when, where and how technology should be inserted. However, the state of the art in modeling and measuring software engineering tool impact is very immature. Understanding the relative cost and benefits for applying software engineering tools is a critical issue for NASA because missions have very constrained resources and are not willing to bear the cost and risk of new technology adoption. In addition, the software engineering components of NASA's research programs do not have the resources required to sustain numerous tool development and transfer activities, increasing the risk of individual transfer efforts. Therefore, mission and research program resources available for technology insertion must be managed effectively, especially if the long-term relationship required for successful technology transfer is to be maintained.

With regard to model checking, because the technology has many potential insertion points in the lifecycle, interfacing with missions is particularly challenging. For example, the cost/benefits analysis for model checking depends on: the point of insertion into process (requirements, design, coding or testing); the availability and quality of requirements specifications; the existence of early lifecycle models; the characteristics of the application (real-time constraints, programming language); and, the education and expertise of the tool user. Therefore, it is difficult to: A) develop the cost/benefit analysis required to show a quantified improvement in assurance and B) perform the comprehensive cost analysis needed to get management buy-in. This effectively limits our research activities to opportunistic case studies and high-risk tool development activities.

While we can argue that our technology is a risk reduction/mitigation technology, this is a difficult argument to win without a strong cost/benefits model. In addition, the area of software risk itself is not very well understood, therefore quantifying (or even understanding) the benefits of mitigating specific software risks for specific missions is not possible. Therefore, better methods for categorizing and assess software related risk within a project is critical for supporting technology transfer.

Any description of the cost and impact of a tool must be described in terms of some metrics. However, the various stakeholders are focused on a wide range of metrics. At the tool level, the focus in on understanding how tool performance metrics impact development metrics such as defect density or rework costs. However, in many cases, metrics efforts are insufficient to support good analysis. For example, data such as defect detection are often not tracked until late in the testing phases of the project. We believe there should be an effort to determine what metrics are important to understand tool impact and to provide standard classification schemes for these metrics (such as early life-cycle defects) to support analysis and comparison of tool benefits.

3.2 Users and usability

Tool usability has not always been a high-priority issue for high-assurance domains. However, as tools are applied to larger systems and as the number of application domains which require high-assurance techniques

broadens, the cost of using the tools and user productivity become serious issues.

One usability issue that always arises for tools based on formal methods is the issue of education. There is some disagreement over what can we reasonably expect a user to know, and who is responsible for providing this education. We don't believe there is a single answer to this question from a tool perspective. Instead, we should focus on capturing the relative cost and benefits associated with different classes of users. However, we are not aware of any efforts to assess tools in this fashion. This type of analysis should also take into account the trade-offs between application domain knowledge and tool knowledge, and incorporate models of human learning to account for the effects of learning curves and variation among users.

Another important issue is that many tools are developed with single users in mind. However, software engineering is, now more than ever, a collaborative experience. While there have been many recent advances in information technology to support collaborative work, innovative applications of this technology to support collaborative software engineering tasks are just beginning. We believe that the research methods used in work practice studies as well as the collaborative technology itself can be used to improve the usability of high-assurance methods and tools. This may be especially true in areas such as requirements elicitation and design reviews which are team oriented, human centered activities. We are currently working on developing research partnerships with collaborative technology researchers within NASA.

3.3 Target application domain issues

Our most significant challenge in terms of engaging mission engineers is being able to handle the complexity of real software with our tools. In the areas of embedded systems, the most common concerns are maintaining fidelity with respect to real-time constraints and interrupt handling. These types of issues have recently been expanding due to the increasing number of real-time frameworks and runtime support libraries which are being used by applications. The brings a new challenge to tools: they must be flexible in terms of runtime elements such as thread models, synchronization mechanisms and support for various runtime "semantics".

If it weren't enough that there are a lot of these technologies, they are also constantly evolving. Java, for example, is continuing to evolve as it becomes more pervasive and moves into the real-time domain. This means that tools must also be flexible in their ability to support language and library evolution. We believe that state of the art software engineering principles and implementation techniques can be used to provide this flexibility. However, because of the cost of developing robust, flexible tool architectures, these efforts should be open, coordinated and shared. We are currently working on open sourcing several of our more mature tool architectures as a way to help address these issues in the community.

3.4 Organizational issues

Our most significant organizational challenge had been in identifying and developing working relationships with potential early technology adopters. Developing this type of relationship requires the equivalent of a "planetary alignment": research goals, technology benefits, tool capabilities, application domain characteristics and customer goals must all line up for an extended period of time.

This is an area where it may be beneficial for multiple projects to pool resources. For example, NASA has recently begun the development of several High Dependability Computing testbed projects which will allow multiple research projects to evaluate software engineering technology on specific NASA project applications.

Conclusions

We have identified several significant challenges for high assurance software engineering tools in the areas of cost-benefits analysis, tool usability, application domain characteristics and organizational issues. In some cases, we have made suggestions on possible research directions which may help address these challenges. However, there are many open issues in this area which makes up the gap between research and practice. We look forward to learning about other open issues and potential solutions to these problems as we develop collaborations with other researchers working on these issues.

References:

[1] Callahan, J. R., and Schneider, F., "Specification-based Testing using Model Checking," 1996 SPIN Workshop, Rutgers University.

[2] Easterbrook and Callahan, "Formal Methods for Verification and Validation of partial specifications: A Case Study," Journal of Systems and Software, 40(3), 1998.

[3] Klaus Havelund, Michael Lowry, SeungJoon Park, Charles Pecheur, John Penix, Willem Visser, and Jon L. White, "Formal Analysis of the Remote Agent Before and After Flight", In *Proceedings of*

the 5th NASA Langley Formal Methods Workshop, Williamsburg, VA, June 2000.

[4] Klaus Havelund, Michael Lowry, and John Penix, "Formal Analysis of a Space Craft Controller using SPIN", *IEEE Transactions on Software Engineering*, June 2001.

[5] Lowry, M., Havelund, K. and Penix, J., "Verification and Validation of AI Systems that Control Deep-Space Spacecraft", Tenth International Symposium on Methodologies for Intelligent Systems, Charlotte, North Carolina, Oct. 15-18, 1997. Springer-Verlag Lecture Notes in Artificial Intelligence, Vol. 1325.

[6] Penix, J., Visser, W., Engstrom, E., Larson, A., and Weininger, N., "Verification of Time Partitioning in the DEOS Scheduling Kernel", 22nd International Conference on Software Engineering, Limerick, Ireland, June 2000.

[7] Schneider, F., Easterbrook, S. M., Callahan, J. R., and Holzmann, G. J., "Validating Requirements for Fault Tolerant Systems using Model Checking," *Third IEEE Conference on Requirements Engineering*, Colorado Springs, CO, April 1998.