Hosted Services for Advanced V&V Technologies: An Approach to Achieving Adoption without the Woes of Usage[1]

Position Paper for ACSE 2003

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

Owen O'Malley
QSS Group, Inc.
owen@email.arc.nasa.gov

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

William A. Brew

Abstract

Attempts to achieve widespread use of software verification tools have been notably unsuccessful. Even "straightforward", classic, and potentially effective verification tools such as lint-like tools face limits on their acceptance. These limits are imposed by the expertise required for applying the tools and interpreting the results, the high false positive rate of many verification tools, and the need to integrate the tools into development environments. The barriers are even greater for more complex advanced technologies such as model checking.

Web-hosted services for advanced verification technologies may mitigate these problems by centralizing tool expertise.

The possible benefits of this approach include eliminating the need for software developer expertise in tool application and results filtering, and improving integration with other development tools.

1. Introduction

Software engineering tool developers face numerous obstacles in getting their tools adopted. Some of these obstacles are listed in the Call for Papers for this Workshop. Others include the cost of the infrastructure for maintaining and applying the tools, and the difficulty of interpreting and filtering the results. Integration with other tools pose additional barriers. The approaches suggested in the CFP represent ways to address these problems.

Our focus in this paper is on adoption of verification tools, specifically, program analysis and simulation tools such as static analyzers and model checkers. These tools are likely to require significant expertise in their use, for reasons that we discuss in the next section. In addition, these tools generally require a greater effort to integrate than "front end" tools such as design tools and compilers. Therefore, for verification tools, a more radical approach may be needed to ensure adoption.

[1] The research on model checking 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. The experience reported regarding other technologies and tools is based on the authors' professional experience developing and applying them in a variety of organizations.

Hosted application service providers may provide an effective way, in appropriate markets, to dramatically lower these acceptance barriers.

2. The problem

The Intelligent Software Engineering Tools team at NASA Ames Research Center (ARC) develops advanced verification tools based on source code model checking technology[1], an ongoing research area in the Automated Software Engineering group at ARC. Our target languages are Java, C and C++. This position paper is based in part on our current work and in part on our previous experience at NASA and elsewhere building or applying a variety of commercial verification tools. These tools include PolySpace[2], Flexelint[3], and Y2K defect detection/ remediation tools as well as research prototypes such as Java PathFinder[1] and ESC/Java[4]. Effective use of many of these tools faces similar problems.

In our experience, specialized expertise is required for effective use and adoption of verification tools. Integration issues also impede adoption.

2.1 Kinds of knowledge required for effective use of verification tools

Effective use of verification tools includes detection of real defects; a low false-positive rate; the ability to triage reported defects; and a high confidence level in the results. Effective use of model checking tools requires a mental model of their operation to interpret the output, tune the model checker's operation, and identify the root cause of defects that it reports. This knowledge is largely application-independent.

In addition to a mental model of tool operation, effective use for our target languages, C, C++ and Java, may require expert knowledge of the semantics of language operators in order to evaluate a defect report. In our experience, C/C++ programmers may not have the level of understanding of language semantics necessary to interpret tool output. As is the case with tool expertise, this knowledge is application-independent.

Effective use of verification tools may also require application-specific knowledge. Static analyzers may be unable to conclude that an operation may produce an exception, because the range of possible values of variables cannot be derived from the source code. This knowledge is often provided in the form of formal specifications or design information.

All of these considerations are prior to root cause analysis, which imposes further demands on the user, if the defect reports are to be actionable. Once the defect is well-understood, confirmed as real, with high confidence, then application-specific knowledge *may* be important in the remediation task.

Our experience with defect detection tools based on static analysis suggests that some of these problem exist even for lint-like tools, which have been available for 20+ years. These tools have such a high false positive rate that programmers are reluctant to apply them: they cannot filter the output efficiently, nor are they motivated to spend time on what they view as "busy work".

2.2 Integration of verification tools

Effective use of verification tools also requires that the tools be well-integrated into the development environment. Static analysis tools can, in principle, be run at compile or build time. Thus they hold forth the promise of early defect detection if they are well integrated into the development process.

Verification tools generally require a greater effort to integrate than "front end" tools such as design tools and compilers, since the verification tools must report defects in a way that supports in-the-loop evaluation and remediation or other action.

Tools that have a high false positive rate, if they are to be integrated at all, require a well-defined filtering process —some combination of automated post-processing and human filtering. Our experience is that developing an efficient filtering process requires extensive experience with the tool and its use in a particular development environment; the distillation of this experience must be retained as enterprise knowledge in a training system because of the high turnover rate of reviewer personnel.

Advanced verification tools themselves are likely to be "niche market" tools, since their range of applicability is limited, and hence the resources available for integration and maintenance will be limited compared to, for example, the resources available for integrating and maintaining a new compiler.

3. Hosted Verification Services

We have argued that effective use of verification tools requires three kinds of knowledge: a mental model of the tool's operation; knowledge of the semantics of the target language; and application-specific knowledge. To the degree that the first two kinds of knowledge dominate, it makes sense to centralize that expertise and even to hide it from users. One way to do this is to provide web-hosted verification services.

Usage scenario. In one scenario for hosted verification services, a developer checks her successfully-compiled source files into the host server's configuration management system (CMS). The nightly build is run on the development team's network, which accesses the files

from the host's CMS. The build transcript is written to the CMS server. Following the build, a configuration analysis tool, which is resident on a server at the hosting service, analyzes the build transcript to determine what files need to be analyzed and how (what compiler and compilation options were used, etc.) Some of the options specific to the verifier have been preset by the service provider's tool experts based on prior customer input about the application or about the current build. These options may include decisions about what defects are of interest to the customer, how much explanation of the defects is to be provided, and the highest priority modules for verification.

The verification tools use the configuration analysis data to initialize the verification options. Then they proceed to analyze the application. Tool operation is monitored by the service provider and human interventions are made as necessary—for example, to make tradeoffs between runtime and completeness of the analysis, or to focus on specific execution paths. The verification tools may need to be run repeatedly, with different settings, to obtain the desired results. Verification output is then filtered by an automated filtering system and may be presented to human reviewers for final filtering. The human-filtered output is directed to other facilities that are provided at the hosting site, such as an issue tracking system and test case generator. Ideally the data are available to the user when she logs on in the morning.

Commercial models. The model for this scenario is not far from existing commercial application service providers (ASPs) of software development tools and services.

For example, DevX and Merant provide hosted issue management tools and services. VA Software and Collabnet provide tools and services for hosted configuration management and collaborative development. SoftGear focuses on testing. Other organizations provide source code inspection services. The degree of user access to the "tools", and what capabilities are provided by ASPs, varies. In some cases the user directly accesses a tool (such as a configuration management system) using a browser interface, and the services include maintenance of the tools and the platform. In other cases the user may simply make submissions (for example, an application to be tested) and later accesses a database for the results—the ASP provides a service based on a combination of tools and human expertise.

Lessons from Y2K verification. Our experience with the Y2K problem suggests the effectiveness of a service-based approach to verification. Solving the Y2K problem for Cobol required extensive program analysis, as well as the ability to understand specific Y2K errors and remediate them systematically. This was beyond the capability of most Cobol programmers, particularly given the time constraints and the massive volume of source code to be examined. Advanced program analysis tools based on alias analysis and program slicing largely automated the analysis, but required a good mental model on the part of the user in order to tune their operation and interpret the results. Our experience was that training Cobol programmers in the required concepts, such as parsing, alias analysis, reaching definitions, evidence and confidence levels, built-in heuristics, and remediation strategies was broadly ineffective. Intensive process and tool training in a "factory" enabled a high throughput whether the goal was independent verification or error detection and remediation.

Application-specific knowledge was largely unnecessary[2]: the most important consideration was obtaining a complete, consistent set of source and copy books (include files).

4. Success and Risk Factors

We are not advocating hosted verification as a panacea. We have already indicated the basic precondition for hosted verification services—dominance of the importance of tool knowledge over application knowledge, or the superior ability of the tool to acquire application knowledge. Specific additional factors within NASA enhance the prospects for hosted verification services there. There are risks also, both within NASA and in a broader context.

Intellectual property & security issues. Commercially, intellectual property issues coupled with security concerns produce a reluctance in some markets to allow source code offsite. This consideration is largely absent within NASA; there are security issues (such as ITAR) but these are addressable through existing procedures. And, as we discuss below, NASA already has an internal software verification facility.

Need for verification. In addition, there is a recognition within NASA that software complexity, particularly for autonomous vehicles, is increasing rapidly, and that advanced software V&V are enabling technologies for autonomous space exploration. NASA ARC has for years been conducting research in software V&V as well as other approaches for reducing defects in autonomous applications. ARC does research and some tool development; it does not provide verification services.

However, NASA does have an organization dedicated to V&V, the Independent Verification and Validation

[2] More accurately, Y2K defect analysis and remediation required *extensive* application knowledge, but this *never* favored the application expert: the toolset *became* the application expert in the course of analyzing the application.

Facility. Its charter includes "identifying system and software risks to improve software quality and safety". We view the IV&V Facility as is a possible natural entry point for hosting verification services within NASA.

Support for software development process improvement. There is also an increasing recognition within NASA that detailed data and metrics should be collected on high-assurance software engineering projects. A hosted development environment, with specialized V&V tools, provides a platform for obtaining such data and evaluating the effectiveness of the development process, including the individual V&V tools. It offers the possibility of obtaining fine-grained enough data on individual V&V tools that NASA can model the return on investment of each tool when used at various points in the development lifecycle and feed this data into risk assessment tools such as [6].

Integration with other tools. The same environment that hosts V&V services should also host other development tools—at the very least, CMS and issue tracking tools. This does not completely address the integration issues mentioned earlier, since builds and testing, for example, are usually not conducted in the hosting environment, and there is a wide range of development environments. However, hosting CMS and issue tracking may provide a synergistic environment—the CMS can provide a complete, consistent configuration for verification; and verification output can be directed to the issue tracking system and possibly the test environment. An integration risk is that there is a large range of development environments, and it may be difficult to integrate closely with the build process.

Turn-around time. Integration into the development environment also suggests that the verification results are available quickly enough to be actionable before the next build. Human intervention in the verification process, which we have argued is necessary to apply verification tools and filter the results, may preclude a rapid enough response—for example, when builds are done on a daily basis. One strategy for mitigating this risk is to provide several levels of verification, where the "deeper" (and more time-consuming) verification levels are reserved for high-assurance applications that are more tolerant of turn-around time. Another strategy is to optimize the verification insertion points in the customer's development cycle—for example, with respect to milestones such as start of integration testing, certification, alpha and beta release, etc. Incremental verification should also improve turnaround time.

One of our research goals is to determine the right lifecycle insertion points for verification tools, especially when several verification and validation tools are used together.

Incremental verification. A related risk is inability to support "incremental" verification on successive builds. The human effort required for verification should be approximately proportional to the amount of "new" code. Certainly this is the customer's perception. A combination of engineering and research may be needed to address this for large applications with frequent builds.

5. Conclusion

We need to understand how the verification tools we are developing at ARC can overcome barriers to deployment within NASA, and how they are best integrated into the development process.

Web-hosted verification services may provide an opportunity for verification technology to gain acceptance in NASA and elsewhere. One of the barriers to the use of verification tools is the expertise required to apply them effectively, which dominates the application expertise required on the part of the user.

Benefits of web-hosted verification services may also include better integration into the development lifecycle; better integration with other software development tools; and the ability to obtain fine-grained performance data for evaluating the effectiveness of particular tools in various contexts.

Risks include difficulty of providing application-specific knowledge to assist the tool; inability of the hosting site to model the application development site; inadequate turn-around time; and the inability to support incremental verification of large applications. In the commercial environment, intellectual property and security concerns may limit acceptance.

References

[1] W. Visser, K. Havelund, G. Brat, S. Park. "Model Checking Programs", *Proceedings of the 15th International Conference on Automated Software Engineering (ASE)*, Grenoble, France, September 2000.

[2] PolySpace is a trademark of PolySpace, Inc. http://www.polyspace.com

[3] Flexelint is a trademark of Gimpel Software, Inc. http://www.gimpel.com/

[4] http://research.compaq.com/SRC/esc/

[5] CodeWizard is a trademark of Parasoft, Inc. http://www.parasoft.com

[6] Raffo, D., and Kellner, M. I., "Predicting the Impact of Potential Process Changes: A Quantitative Approach to Process Modeling," *Elements of Software Process Assessment and Improvement*, IEEE Computer Society Press, 1999

[7] Cousot, P. "Abstract Interpretation: Achievements and Perspectives." http://www.polyspace.com/docs/Abstract_Interpretation_P_Cousot.pdf