Innovations Syst Softw Eng (2005) 1: 3–11
DOI 10.1007/s11334-005-0007-z

ORIGINAL PAPER

Mikael Lindvall · Ioana Rus · Forrest Shull
Marvin Zelkowitz · Paolo Donzelli
Atif Memon · Victor Basili · Patricia Costa
Roseanne Tvedt · Lorin Hochstein
Sima Asgari · Chris Ackermann · Dan Pech

An evolutionary testbed for software technology evaluation

Received: 1 September 2004 / Accepted: 28 December 2004 / Published online: 11 March 2005
© Springer-Verlag 2005

Abstract Empirical evidence and technology evaluation are needed to close the gap between the state of the art and the state of the practice in software engineering. However, there are difficulties associated with evaluating technologies based on empirical evidence: insufficient specification of context variables, cost of experimentation, and risks associated with trying out new technologies. In this paper, we propose the idea of an evolutionary testbed for addressing these problems. We demonstrate the utility of the testbed in empirical studies involving two different research technologies applied to the testbed, as well as the results of these studies. The work is part of NASA's High Dependability Computing Project (HDCP), in which we are evaluating a wide range of new technologies for improving the dependability of NASA mission-critical systems.

1 Introduction

A large gap exists between the state of the art and the state of the practice in software engineering. Research into software engineering practices, methods, and tools is producing a large number of exciting new technologies that promise to reduce the cost or improve the quality of software. By technology we refer to algorithms, techniques, methods, methodologies, processes, and software tools that can be used to develop software-intensive systems. Examples of technologies are: agents, agile methods, architecture languages, aspects, assertions, and automated programming – to name just a few. However, while research efforts are continuing apace, only a small percentage of these technologies is seeing any significant use on real development projects.

It is becoming more commonly accepted among software engineering researchers that empirical evidence is needed to close this gap. That is, we can know whether a technology is feasible, widely usable, and capable of meeting the claims specified for it only if the technology has been applied by others than the technology developer and empirically studied. However, there are some key obstacles that make this difficult.

Insufficient specification of context variables It is clear that there are many types of users and many contexts in which a technology can be applied. These context variables clearly bound and limit the effectiveness of a technology. The fact that a given technology was evaluated positively under certain conditions does not a priori imply that it must thus be good under all other project conditions. Determining which conditions do or do not make results transferable between project environments is an open and challenging question.

Cost Researchers have two possible strategies for exercising their new technologies: (1) to create realistic development artifacts from scratch on which they can be applied or to (2) work with development teams, learn their context and terminology and familiarize themselves with appropriate artifacts, and apply the technology to these artifacts. Either strategy imposes severe costs on researchers. Moreover, conducting studies with representative subjects tends to increase the costs of experimentation (the most relevant subjects for most studies come from the population of professional software developers, whose time is almost always overbooked and highly expensive) while making it more difficult to achieve statistical significance.

Risk Trying out new technologies (or even technologies new to a given environment) under conditions approaching a

M. Zelkowitz · P. Donzelli · A. Memon · V. Basili
L. Hochstein · S. Asgari
University of Maryland, MD, USA
E-mail: {mvz,basili}@fc-md.umd.edu
{donzelli, lorin Sima}@cs.umd.edu

M. Lindvall (✉) · I. Rus · F. Shull · M. Zelkowitz · V. Basili
P. Costa · R. Tvedt · C. Ackermann · D. Pech
Fraunhofer Center for Experimental Software Engineering,
MD, USA
E-mail: {mlindvall, irus, fshull, mvz, basili, pcosta, rtvedt, cackermann, dpech}@fc-md.umd.edu

realistic project environment carries significant risk. Not only can unsuccessful attempts not help, but they may also actively hinder a project by requiring unbudgeted effort, missing important project problems, etc. If the project fails because of the new technology, the project manager will be faulted for taking an "unnecessary" risk.

To address these problems, we propose the idea of packaged "testbed" applications that can facilitate empirical studies of new technologies in a way that assists the eventual technology transfer of mature results into live project environments. Such testbeds can reduce costs (researchers can make use of software artifacts, at a useful level of representativeness, without investing the effort to create them from scratch, even if the use of testbeds does not explicitly address the problem of involving representative subjects); reduce risks (the testbeds are designed to support an iterative cycle of experimentation, predicated on the idea that technologies can mature over time based on empirical results, that "buys down" risk before taking the technology to live project environments); and help contextualize the results (as the context of each testbed can be described in detail for the research community to use). In addition, testbeds can be used as a vehicle for collaboration between researchers because it allows them to work on common artifacts provided by an independent source.

This paper describes the HDCP testbed concept by providing a detailed description of a technology testbed that we constructed for NASA and of the process by which technology researchers can use the testbed to generate empirical data about their own technologies. We show that the testbed adequately addresses the above problems by describing the application of empirical studies involving two different research technologies to the testbed and showing the detailed results of these studies. Lessons learned and future work conclude the paper.

1.1 Related work

We have some confidence that publicly available packaged testbeds can help in this area because of our prior experience in technology transfer in the NASA Software Engineering Laboratory (SEL). The SEL work involved studies of technologies at different levels of maturity, under conditions ranging from small, sample, offline documents to live projects at NASA (in which we performed smaller experiments and monitored the use of the technology quite closely) [2]. These experiences helped inform our process for how unacceptable risks to projects can be retired with offline studies.

We have been experimenting on understanding what constitutes a useful "testbed" of software artifacts that can be reused in multiple empirical studies. For example, we have run multiple experiments on reusable software artifacts, from multiple domains, that were seeded with defects in order to evaluate technologies. Some of those artifacts had a long history of being reused by other researchers. In some cases, this was done to replicate or explore our results in more detail. For example, one study of inspection techniques [1] that drew conclusions about team performance was replicated by independent researchers who analyzed in more detail the performance of different team members [9]. In other cases, it was done to allow the evaluation of new technologies against a well-understood baseline. (For example, a recent study of testing technologies [10] reused documents on which we had previously evaluated different variants of software inspections [12]). We have analyzed the strengths and weaknesses of such artifacts for transferring explicit and implicit knowledge from the original creators to other research users [11]. Weaknesses we uncovered were that these "testbed" artifacts were static and limited with respect to the set of technologies for which they were relevant, thus restricting the technology studies that could be performed on them.

In NASA's High Dependability Computing Project (HDCP), we are faced with the problem of evaluating a wide range of proposed new technologies for improving the dependability of NASA mission-critical systems (http://www.hdcp.org/). This means that we are not part of any single development environment, as was the case in our work in the NASA SEL, nor could we deal with small testbeds focused on a small set of technologies. We needed an evolvable testbed that could be easily extended to accommodate new technologies and further experimentation. The testbed concept described in this paper extends the experiences with reusable artifacts to meet HDCP needs.

2 HDCP and technology transfer

HDCP is a NASA initiative for increasing dependability of software-based systems. The HDCP project proposes and investigates the achievement of high dependability by introducing new technologies that are developed at participating universities and research centers.

HDCP views high-dependability technology evaluation as passing through a series of milestone gates, each demonstrating its context of effectiveness. The milestones go from level 1 (internal set – least mature) to level 4 (live examples – most mature). We indicate in Table 1 the NASA technology readiness levels (TRLs) associated with each of the HDCP levels (TRL 1 representing a concept and TRL 9 corresponding to an operational technology).

A significant gap between level 1 and level 3 needed to be addressed in the HDCP project. Level 1 represents a situation in which the inventor of a new technology applies it to a software system that she knows well. This form of study is relatively uncomplicated since it is likely that most inventors have access to or can develop a software system to which they can apply their technology. Level 1 studies are necessary for initial experimentation with a technology; however, the value of the results is limited. The experimenter knows not only the system and its correct behavior well, but also the technology and what is expected from it. Because of this knowledge and bias (the inventor wants the technology to show promising results), it is difficult to generalize

Table 1 Technology levels

HDCP level	NASA TRL	Milestone description
4	7 or 8	Live examples: this milestone definition is specific to a part or all of a system currently under development at NASA, a NASA contractor, or an HDCP company by mutual agreement of all concerned.
3	6	HDCP domain-specific offline set: domain-specific set of examples chosen from past NASA or NASA-contractor projects or from HDCP company projects.
2	5	Scaled-down versions of NASA-related software-intensive systems with high-dependability requirements.
1	3	Internal set: typically internally developed set of examples on which the technology researcher has applied the technology.

Table 2 HDCP Milestones

Milestones	Subject's level of independence from		Objectivity
	Testbed	Technology	and credibility
Level 2c	High	High	High
Level 2b	High	Low	↑
Level 2a	Low	High	
Level 1	Low	Low	Low

the results from such initial studies. In order to show that the technology is feasible and also meets its claims in other contexts, it is important to experiment on higher levels. Level 2 represents a scenario in which the inventor of a technology applies it to a software system that is a scaled-down version of NASA-related software. Several NASA projects were interested in participating in this activity, for example center-TRACON automation system (CTAS) and Mars Science Laboratory (MSL), but access restrictions made it infeasible to scale down use for experimentation. Increased security measures made it difficult for non-US citizens to investigate these and other similar systems. In order to bridge the gap to level 3, level 2 testbeds were created that could serve as replicas of scaled-down NASA applications. Thus, CTAS was represented by the tactical separation assisted flight environment (TSAFE) testbed, an air traffic system identifying the position of aircraft on a map, initially developed at MIT [5] and turned into a testbed at FC-MD. SCRover, developed at the University of Southern California, is representative of MSL [4].

Focusing on level 2 unveils several sublevels in terms of the experimenter's degree of independence from technology and testbed. Levels 1 and 2 can be characterized as presented in Table 2. In level 1, the inventor of the technology experiments on a familiar testbed. Within level 2, the level of independence, and therefore the objectivity and credibility of the experiment, increases from level 2a to level 2c. In level 2a, the experimenter experiments with a technology invented by someone else on a testbed she is familiar with. In level 2b, the experimenter experiments with technology developed by herself on an unfamiliar testbed. In level 2c, the experimenter is an external assessor, familiar with neither the testbed nor the technology.

Table 1 shows how testbeds can be used to design experiments with increasing objectivity and credibility, thereby reducing the risks of using new technologies by prequalifying technologies in a related context, enabling cost-effective HDCP technology integration, and accelerating the pace of HDCP technology maturity and transition.

In order to develop an efficient testbed that would meet the needs of experimentation in level 2, a HDCP testbed must be under version and configuration control and must fulfill the following requirements:

- Representative of NASA and NASA-related missions.
- Easy to distribute and use.
- Open to non-US citizens and permanent residents (not subject to ITAR and other similar restrictions).
- Easy to evolve and tailor to different technologies.
- Cost-effective on people and computing resources.[1]

3 Testbed and its usage in experimentation

We define a testbed as a set of artifacts associated with a software (or a software-intensive) system, together with the infrastructure needed for running experiments on that system. If the system under study includes hardware, then it is part of the testbed as well.

By experiments we mean verifying hypotheses associated with software development technologies. In other words, evaluating the effects of using technologies in the development of software with respect to well-defined goals (e.g., feasibility, effectiveness, or cost).

A testbed is not static; it evolves as existing artifacts and infrastructure features are improved and added. Thus, over time there will be many different *versions* of these items. For each experiment on the testbed, a set of artifacts and infrastructure features is selected based on the design of the experiment and the characteristics of the technology. This set of items constitutes a *configuration* of the testbed.

Testbed artifacts are either product- or process-oriented. Product-oriented artifacts include source code, executables, and documentation such as requirements, design, test plans, test cases, and user manuals. Process-oriented artifacts include project plans, descriptions of methods and techniques applied during software development, and their results (e.g., results of inspections, testing, and certification). Different

[1] Based on a presentation by Barry Boehm at the HDCP workshop in June 2004

technologies are designed to detect different classes of faults. An important testbed artifact is therefore the set of seeded faults that can be used to create a configuration of the testbed.

The infrastructure consists of instrumentation that must be added to the system in order to experiment with it. For example, the code needs to be instrumented in order to monitor and record its execution, and test case generation and execution must be automated.

The testbed has an associated experience base in which experimental designs, results from previous experiments, lessons learned, and history of evolution are kept. This is analogous to experimental protocols frequently used in medicine [7].

Stakeholders in the experimentation process include the *technology developer*, the *testbed developer*, and the *experimenter*. A person can have more than one role. Each of these individuals has a certain familiarity with the subjects and objects of the experiment. For example, the technology developer is familiar with the technology while the testbed developer is familiar with the testbed. Some bias (intentional or not) in designing and running an experiment might result from this familiarity or from the interest of a stakeholder. For example, the technology developer might want to demonstrate that her technology satisfies its claims (rather than to evaluate whether or not that is true, i.e., the well-known psychological principle of confirmation bias where the believers of a phenomenon are more likely to observe it). Therefore, the value of an experiment in terms of objectivity and credibility increases with decreased dependency of the experimenter in relation to the technology and the testbed, as illustrated in Table 1.

4 Building, using, and evolving the TSAFE testbed

The TSAFE concept was defined at NASA Ames Research Center [6] and was proposed as a principal component of a larger Automated Airspace Computing System. The goal is to shift the burden of ensuring aircraft separation from human controllers to computers. The TSAFE software, upon which we based the testbed, is a 20 KLOC Java program that performs two primary functions: conformance monitoring and trajectory synthesis. In addition to the source code and an overall description, MIT provided an installation guide and a user manual as well as recorded flight data for test input.

As part of our effort to turn TSAFE into a testbed, we added, based on the existing artifacts, a requirements specification and an architecture document. We synthesized faults that were seeded into the source code. Test cases are under development. We added a testbed infrastructure to facilitate experimentation, e.g., support to seed the code with synthesized faults, to monitor seeded faults as they get executed, and to generate system output and traces that can be captured and used to determine the status of the TSAFE system under execution. A test case input generator is under development. Some initial studies were conducted and documented to serve as examples for other experimenters interested in using the

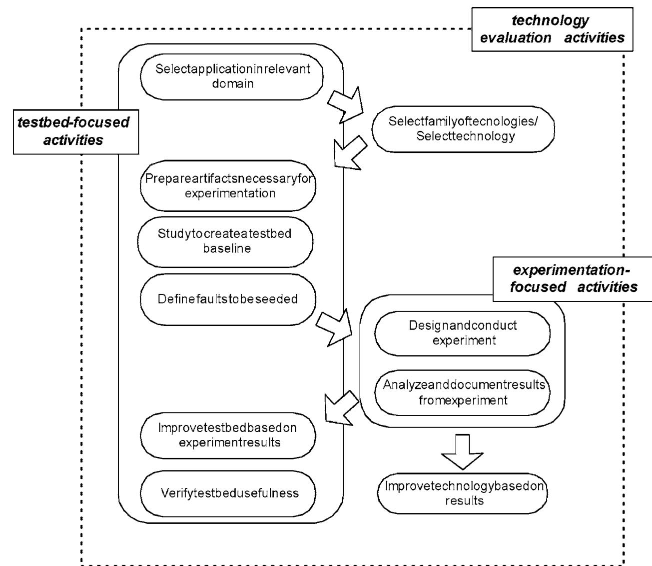

Fig. 1 Testbed-related activities

testbed. The experience from these technology experiments as well as feedback and lessons learned have been collected and will be provided together with the other artifacts as part of the testbed experience base in order to maximize the usefulness of the testbed and minimize the cost and effort of experimentation using the testbed. Artifacts such as documents are managed in hyperwave using its version and configuration management system. Artifacts such as source code are managed in CVS.

4.1 Testbed-related activities

For building, using, and evolving the testbed we performed the following activities (Fig. 1):

1. Select application in relevant domain.
2. Select family of technologies.
3. Select one technology within the family.
4. Prepare artifacts necessary for experimentation.
5. Conduct study to create a testbed baseline.
6. Define faults to be seeded.
7. Design and conduct experiment.
8. Analyze and document results from experiment.
9. Improve testbed based on analysis.
10. Improve technology based on analysis.
11. Verify testbed usefulness.

How many of these steps will be repeated in future experiments on the testbed depends on whether or not the new technology is part of a technology family that has already been applied to the testbed.

4.1.1 Select application in relevant domain

The Tactical Separation Assisted Flight Environment (TSAFE) was the selected application, and the relevant domain was NASA software for air traffic control. TSAFE is a

good choice for a software testbed due to its relationship with the CTAS system and its related dependability requirements. TSAFE will automatically and independently make decisions without relying on human controllers. TSAFE software is reasonably complex without being too difficult to study.

4.1.2 Select family of technologies

We performed a brief characterization of the technologies in HDCP and identified several families of technologies. Members of a family possess some commonalities, such as the input artifacts on which they operate and the development activities they support, their purpose, and outcome. For each of these families, different artifacts, infrastructure, and seeded faults are needed. Since many of the technologies in the HDCP project are related to software architecture analysis, we concentrated on such technologies.

Software architecture deals with the structure, interactions, and behavior of a software system. The building blocks of software architecture are its components and their interrelationships. Constraints and rules describe how the architectural components communicate with one another.

We focused on software architectural evaluations, which are investigations into how a system is structured and behaves, with the purpose of suggesting areas for improvement or for understanding various aspects of a system (e.g., maintainability, reliability, or security). In many cases, a software architectural evaluation is performed before a system is designed or implemented. In other cases, a software architectural evaluation is performed after the system has been implemented. This type of architectural evaluation is typically performed to assure that the actual implementation of a system matches the planned architectural design [13].

We chose a set of technologies that support *implementation-oriented software architectural evaluation*. These evaluations typically reconstruct the actual software architecture from the source code in order to compare it with the planned architecture.

4.1.3 Select one technology within the family

The particular technology chosen for initial experimentation is called software architecture evaluation (SAE) [13]. It identifies deviations between the planned architecture and the actual (implemented) architecture of a software system. This analysis is used to evaluate whether the implemented software architecture follows the planned software architecture and associated goals, rules, and guidelines.

The quality issues that SAE claims to find are discrepancies between the planned and the actual architecture. The planned architecture is represented by a set of components and their interconnections, as well as some information regarding classes that constitute the component interface. SAE thus claims to find missing as well as extra interconnections between components and to identify when classes other than the specified interface classes are used to access components.

4.1.4 Prepare artifacts necessary for experimentation

The necessary artifacts were determined by analyzing the inputs to the selected technology: planned and actual architecture. The planned architecture was based on architectural information in the original TSAFE documentation. It describes the components and their interactions, the design rationale behind this particular architectural design, and the mapping from components to source code constructs. The actual architecture is extracted from the source code using a tool that is part of the SAE technology, and hence the related preparation was to install the testbed source code in the correct folders.

4.1.5 Conduct study to create a testbed baseline

The first study involved applying SAE on the TSAFE testbed in order to detect architectural violations. The analysis revealed three such violations. Further analysis of the detected violations determined that they were all minor and not sufficient for experimentation; therefore, faults needed to be synthesized.

The baseline study resulted in a better understanding of the testbed architecture and its source code and allowed us to improve the testbed artifacts. For example, a set of architectural rules was added to the architecture document, making it easier to understand. These rules explicitly specified the components of the testbed and their interconnections. They named the classes of each component that were considered interface classes and through which all communication with the component should occur. The rules also explicitly described each occurrence of design patterns and the expected implementation.

4.1.6 Define faults to be seeded

Since there are many different families of technologies that find faults of different types, the fault space covered by all technologies is quite large. Instead of attempting to cover the entire fault space, we decided to start with faults related to the selected technology. Thus, we defined (synthesized) a set of faults based on the claims of the SAE, which is related to the architecture of TSAFE. For each of these faults we documented the architectural rules that they violated.

We then divided the faults into three different fault sets. There were several reasons for creating several such sets. First, we deemed it unrealistic, based on our experience from previous studies of this technology [13], that a source code of the size of this testbed (in terms of number of components and their interrelations) would contain more than ten architectural violations. Second, it was impractical to seed all architectural faults into one version since faults interfere with each other, making it difficult to analyze the results for such faults. Third, we wanted to run repeated experiments based on the same technology and therefore needed several testbed configurations based on different sets of faults. The results were four different testbed configurations: three fault-seeded

versions of the testbed source code in addition to the original version of TSAFE, which is considered a baseline to which we compare faulty versions of the testbed.

4.1.7 Design and conduct experiment

We designed an experiment in which the technology would be applied to the testbed, with the goal of investigating the feasibility of SAE for project use, concentrating especially on the cost (how much effort is required to be spent on its use) in comparison with benefits (how many architectural issues are found that would realistically impact system quality, specifically flexibility and maintainability). Since this was an early study of the application of SAE, we were most interested in whether there was enough evidence of the usefulness of the technology to support further work. For this reason, the study design concentrated on answering the following specific research questions:

- Can SAE, when applied by a user knowledgeable in SAE, really find the types of quality issues it was designed to uncover?
- Does applying SAE result in the identification of other quality issues outside of the original target set?

To allow for experimentation of SAE on the testbed, the experimental team prepared architectural documents including a high-level diagram of the testbed software and the set of rules describing features the architecture should possess. Of the 42 rules, 29 describe how components are supposed to interact and were considered to be relevant to SAE. Thus, SAE was expected to detect violations of these 29 rules. The other 13 were related to lower-level details of the software design such as information on how classes inside components interact. Thus, SAE was not expected to find violations of these 13 rules.

Next, as stated above, three versions of the source code were prepared, each seeded with different faults. Version 1, for example, included a set of 13 rule violations (i.e., 13 instances where the actual architecture did not conform to the planned architecture). Of these, SAE was expected to detect 7. These 7 expected rule violations were considered to map to 4 distinct faults in the code.

Each fault-seeded version was used in a separate experiment conducted by the same subject (as described below). This experimental design allowed us to replicate our own study twice. Because SAE is largely tool-based (a tool automatically retrieves the actual architecture from the source code, compares it with the planned architecture, and reports deviations), this was not seen as a major threat to the validity of the study (the tool does not learn from one application to the next). However, because SAE does require some interpretation of the tool output on the part of the subject, there was a slight risk that the human subject would notice differences in the system from one run to the next, which would have allowed her to find faults more effectively than would be expected on a real system.

In this study, the subject was a researcher who was a member of the team developing SAE. The subject could be considered highly experienced with the object of study (SAE) as well as highly experienced with software architecture issues, although a novice with TSAFE. This was an attractive choice for an initial study since the subject required no training in SAE and was less likely to produce faulty results based on a misapplication of the technology. To mitigate the risk of the associated bias, the subject was not included in any planning about the study, the selection of the testbed, or the seeding of faults. This series of experiments is thus an example of experimentation on level 2b: the independence of the subject from the technology is low, but the independence from the testbed is high.

To assess the feasibility of the SAE technology, this study measured the number of seeded rule violations and the number of seeded faults that were found by the human subject.

4.1.8 Analyze and document results from experiment

The results of the study were useful for further evolution of the SAE technology:

- The time spent on applying SAE was about 4 h, which was not considered prohibitive.
- Six out of the seven expected seeded rule violations were found. The one seeded fault that was missed helped identify a defect in the tool that has since been fixed.
- The six violations found by SAE were considered to map to three out of the four faults, meaning that the tool defect caused one of the four faults to be missed.
- An additional three out of the six unexpected rule violations were found, which indicates that SAE can be useful outside the set of narrowly targeted problems it was designed to catch.

The results were similar for all three experiments.

Our conclusion is that SAE, when applied by a user knowledgeable in SAE, can find most of the types of quality issues it was designed to uncover. We also conclude that applying SAE results in the identification of other quality issues outside of the original target set.

4.1.9 Improve technology based on analysis

Software architecture evaluation can be improved to detect all the faults that it claims to uncover, and the SAE process can be extended to cover, in a systematic way, issues outside the original target set. It was discovered that architectural violations could shadow each other. Therefore, identifying and removing the first detected violations may not be sufficient. Thus, it was determined that the SAE technology would benefit from additional instructions that would help the user identify such problems in a structured way, thereby ensuring the detection of all violations.

The fact that more violations than expected were found indicates that the technology can be improved inexpensively. This is due to the fact that the subject used already existing information in new ways that led to the discovery of these unexpected rule violations.

The results also indicated how SAE could be improved in other areas. Design patterns are considered important constructs of a software's architecture. Violations of design patterns are, for example, not systematically detected unless design patterns involve several components. SAE should be improved by adding instructions for how to detect such violations when design patterns are used inside components.

4.1.10 Improve testbed based on analysis

Based on the analysis of the experimental results, a number of testbed improvements were made. For example, the subject reported multiple instances where misstated rules caused some confusion as to what exactly should be checked in the code. As a result, the language describing the rules belonging to the testbed was restated and improved.

Measuring our effort revealed that it took 331 h to prepare the experiment (develop the artifacts: 100 h, synthesize the faults: 200 h, design the experiment: 20 h, develop the data collection form: 10 h, install the testbed source code: 30 min, and instruct the subject: 30 min), 4 h for the subject to complete the experiment, and 10 h for the experimenter to analyze the results of the experiment. Total time for preparation and analysis of the experiment was 341 h. The goal is to reduce this cost drastically, and the proposed solution is the reuse of testbeds.

4.1.11 Verify testbed usefulness

To study the usefulness of the testbed concept, we conducted a similar experiment with a related technology by applying reflexion models [8] to the fault-seeded TSAFE version 1. The RM technology is similar to the SAE. The RM technology detects absences and divergences between the planned and the actual architecture and is typically used to help develop a better understanding of a complex software system.

We used the same fault seeding and the same set of artifacts, including data collection forms and experimental instructions, as in the previous experiment. The subject was already familiar with the RM technology, which minimized the learning time.

The two variables that changed were the technology (The RM technology instead of the SAE technology) and the subject (from inventor to noninventor). This experiment is thus an example of experimentation on level 2c: the independence of the subject from the technology is high, as is the independence from the testbed.

Results from testbed usefulness experiment The results of the study were that:

- The time spent on applying RM was 4 h.
- Fourteen issues were reported, of which eight were related to suggested improvements of the rules rather than architectural faults.
- The violations found by the RM technology were considered to map to two out of the four faults. However, because of the way the RM technology detects absences and divergences, the two undetected faults were outside of its scope and thus not expected to be found.
- No additional unexpected rule violations were found, which indicates that the RM technology identifies the problems it was designed to uncover.

Comments on testbed usefulness The results from applying the RM technology shed light on the need for support for describing and navigating hierarchies involving components and subcomponents of a software system. This is especially important for larger systems composed of components on many different levels of abstractions.

Comparing the results from applying the technologies SAE and RM identified a significant overlap between the faults detected by the two technologies. It was also noticed that the two technologies provide different and complementary tool support and that they analyze, detect, and report differences between planned and actual architectures in a different manner. This and other insights into the differences between the two technologies resulted in the creation of a research project with the goal of combining them and creating a more powerful technology.

Most importantly, the study of the RM technology confirmed the value of the testbed. It was indeed cost-efficient to reuse the testbed for this study. It took 60 min for the experimenter to prepare the experiment (develop the artifacts: 0 h, synthesize the faults: 0 h, design the experiment: 0 h, develop the data collection form: 0 h, minutes, retrieve and prepare the data collection form from the experience base: 15 min, install the testbed source code: 15 min, and instruct the subject: 30 min), 4 h for the subject to complete the experiment, and 6 h for the experimenter to analyze the results of the experiment. Total preparation and analysis time was 7 h.

5 Testbed experience and lessons learned

The fact that the faults were seeded into three different versions of the testbed allowed us to replicate our own experiment several times. This helped us improve the architectural rules, faults, and experimental procedures as well as the data collection forms.

One of our goals was to evaluate SAE claims and identify limitations of this technology; thus we synthesized and seeded faults related to this goal. Other goals and other technologies may require different faults to be seeded. As additional technologies are applied to the testbed, leading to an increase in the number and type of uncovered faults, the coverage of the fault space will increase.

The testbed is currently configured to support experimentation in which software tools are used to find faults, but new artifacts will be added in order to support other kinds of technologies as well.

Another limitation of the testbed is that it supports the Java programming language, but we do not see this as a problem because most of the HDCP technologies use Java.

It is clear that testbed development requires investment up front, and we expect to recover this investment by being able make the testbed available for experimentation by many researchers.

6 Usage scenarios supported by the testbed

Several scenarios describe how different stakeholders, such as technology developers, practitioners, and educators, can use the testbed.

Comparing effects of technology versus a baseline For technology developers whose technology is similar to other technologies already applied to the testbed, the testbed has many benefits. The results from previous experiments can be used as baselines against which technologies can be compared. A technology developer can apply her technology to the testbed and compare the results to the data and to experiences recorded from experiments with other technologies. Such comparisons can help the technology developer determine strengths and weaknesses as compared to related technologies. The results may help the technology developer convince real projects to apply and use the technology (level 3). Another benefit is the ability to experiment with the technology on the testbed without incurring the costs of creating a testbed or experimental material as the testbed is already defined and experimental material relevant to the technology is already available.

Confirming effectiveness and identifying areas for improvement For technology developers whose type of technology has not yet been applied to the testbed, there are still many benefits. The product-oriented artifacts can be reused as is. The instrumentation and other infrastructure features can possibly be reused. The seeded faults and experimental designs can serve as examples of how to synthesize new faults and to design experiments for the new technology.

Choosing technologies to improve software quality Based on the results from experiments on the testbed, potential technology users can analyze the effects of the various technologies on the development process. Based on this analysis, potential technology users can make better decisions regarding technologies that will be beneficial to improving their software.

Adapt and use as class material Educators can reuse many of the artifacts included in the testbed for class projects. Educators wanting to provide hands-on experience in software engineering to students need projects that can be completed in a semester and yet are interesting and represent real-world projects. Coming up with such projects from scratch is a very expensive task. Educators can reuse the testbed artifacts in their class projects and feed back the results regarding the artifacts they used, so that the experience from using the artifacts is captured and shared.

Teaching up-to-date development practices and their effects Educators can use the results of the applications of technologies to the testbed to demonstrate examples of applications of software development practices. Educators can also use the repository of technologies applied to the testbed to identify new technologies that they want to teach to students.

Teaching empirical studies and replicating studies The collection of data we will provide from the different studies from the technology developers can be used to illustrate the various types of studies and their benefits. Educators can also replicate studies in order to teach a more hands-on class on empirical studies.

7 Future work

Our current work focuses on improvement and evolution of the TSAFE testbed, tailoring it for experiments targeted to specific software attributes such as dependability and maintainability, and integration with related HDCP testbed systems. To improve the testbed, we will have external researchers use the testbed for experiments conducted remotely and completely independent of us. We will gather their feedback regarding how complete and easy the testbed was to use when running their experiments. In this way we will also improve the packaging and sharing of testbed-related material for developing the empirical study experience base around the testbed.

So far we have developed the testbed for (and experimented with) technologies that apply to software architecture. The next step is to extend the testbed for experiments with a different kind of technology, e.g., code-related technology. A study is under way that characterizes tools that find malicious code that threatens the security of a system. The observations regarding reusability of the testbed are similar to those reported above.

For HDCP, since the focus is on evaluating technologies with respect to dependability, we have developed a method for defining software dependability for a specific system [3] and applied it to TSAFE. By linking the potential failures that impact on dependability to the faults that might cause them, and examining the effect of different technologies on preventing, detecting, or removing these faults, we are able to design experiments and augment TSAFE for assessing the effect of technologies on dependability. We also plan to do the same for maintainability.

We are coordinating and exchanging experience with USC for testbed development and usage. We will create a repository of testbeds and empirical studies that will integrate our TSAFE and SCRover-related contributions. We are also planning on integrating the TSAFE testbed with the HDCP Redwood framework.[2]

We are building a testbed experience base so that the community can use and continuously improve these testbeds as well as the collected design of experiments and empirical results. The experience base will include data regarding results from the application of the technologies to corresponding testbeds.

8 Summary and conclusion

This paper identified, on the one hand, the need for testbeds to bridge the gap between technology developers and their

[2] Presented at the HDCP workshop in June 2004

laboratory experiments and, on the other hand, potential users of new technologies and their needs to assess the readiness and benefits of a technology before applying it. The paper also addressed the need to perform experiments (in order to characterize and evaluate technologies) that are less expensive, allow for comparison of results, and are easier to replicate compared to current practices. We presented the development process for the TSAFE testbed and its usage for experimenting with the software architectural evaluation and reflexion model technologies. We exemplified the use of the testbed and discussed how we assessed the usefulness of the testbed. We showed how a testbed approach can augment the level of independence in experimentation, thereby increasing the objectivity and credibility of experimental results. We demonstrated that using a testbed addresses some of the issues with experimentation such as cost, replication, and comparison of technologies. For example: the time to prepare and analyze the second experiment was 7 h instead of 341 h. We showed that the approach of experimenting with publicly available testbeds and their experimental results is beneficial for technology developers as well as for project managers who need to make decisions about which technologies to use in their projects.

Acknowledgements The authors thank Gregory Dennis for licensing TSAFE under the GNU licensing policy and enabling us to turn it into a testbed. The authors acknowledge support from the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298. We thank our HDCP team members at the University of Southern California led by Dr. Barry Boehm, Winsor Brown, and Alex Lam, and Jen Dix for proofreading the paper.

References

1. Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sorumgaard S, Zelkowitz MV (1996) The empirical investigation of perspective-based reading. J Empir Softw Eng 1(2):133–164
2. Basili VR, McGarry FE, Pajerski R, Zelkowitz MV (2002) Lessons learned from 25 years of process improvement: the rise and fall of the NASA Software Engineering Laboratory. In: IEEE Computer Society and ACM international conference on software engineering (ICSE 2002), pp 69–79
3. Basili V, Donzelli P, Asgari S (2004) Modelling dependability – the unified model of dependability. University of Maryland, Technical Report CS-TR-46–01
4. Boehm B, Bhuta J, Garlan D, Gradman E, Huang L, Lam A, Madachy R, Medvidovic N, Meyer K, Meyers S, Perez G, Reinholtz K, Roshandel R, Rouquette N (2004) Using empirical testbeds to accelerate technology maturity and transition: the SCRover experience. In: 2004 international symposium on empirical software engineering (ISESE‘04), pp 117–126
5. Dennis G (2003) TSAFE: building a trusted computing base for air traffic control software. Masters Thesis, MIT, Cambridge, MA
6. Erzberger H, Paielli RA (2002) Concept for next generation air traffic control system. Air Traffic Control Q 10(4):355–378
7. Kitchenham BA, Dyba T, Jorgensen M (2004) Evidence-based software engineering. In: Proceedings of the 26th international conference on software engineering, pp 273–281
8. Murphy GC, Notkin D, Sullivan K (1995) Software reflexion models: bridging the gap between source and high-level models. In: Proceedings of the 3rd ACM SIGSOFT symposium on the foundations of software engineering. ACM Press, New York, pp 18–28
9. Regnell B, Runeson P, Thelin T (2000) Are the perspectives really different? – further experimentation on scenario-based reading of requirements. J Empir Softw Eng 5(4):331–356
10. Schroeder PJ, Bolaki P, Gopu V (2004) Comparing the fault detection effectiveness of N-way and random test suites. In: Proceedings of the 2004 international symposium on empirical software engineering, Redondo Beach, CA, pp 49–59
11. Shull F, Basili V, Carver J, Maldonado JC, Travassos GH, Mendonça M, Fabbri S (2002) Replicating software engineering experiments: addressing the tacit knowledge problem. In: International symposium on empirical software engineering (ISESE’02) Nara, Japan
12. Travassos GH, Shull F, Fredericks M, Basili VR (1999) Detecting defects in object-oriented designs: using reading techniques to increase software quality. In: Proceedings of the conference on object-oriented programming, systems, languages, and applications (OOPSLA)
13. Tesoriero Tvedt R, Costa P, Lindvall M (2002) Does the code match the design? A process for architecture evaluation. In: Proceedings of the international conference on software maintenance, pp 393–401