Empirical-based Estimation of the Effect on Software Dependability of a Technique for Architecture Conformance Verification

Sima Asgari[1], Victor Basili[1,2], Patricia Costa[2], Paolo Donzelli[1], Lorin Hochstein[1], Mikael Lindvall[2], Ioana Rus[2], Forrest Shull[2], Roseanne Tvedt[2], Marvin Zelkowitz[1,2]

{basili|sima|donzelli|lorin}@cs.umd.edu;{pcosta|mlindvall|irus|fshull|rtesoriero|mvz}@fc-md.umd.edu

1. University of Maryland 2.Fraunhofer Center for Experimental Software Engineering, Maryland

Abstract

The High Dependability Computing Program (HDCP) project is a NASA initiative for increasing dependability of software-based systems. It researches achieving high dependability by introducing new technologies. We focus on the evaluation of the effectiveness of technologies with respect to dependability. We employ empirical evaluation methods along with evaluation testbeds. In this paper, our technology evaluation approach is described. A testbed representative of air traffic control applications is used. An experiment to evaluate a technology to for identification of architectural violations is presented.

1. Introduction

Developing high dependability software requires a) specifying the dependability requirements of a system, b) using development techniques and methods (what we will call “technologies”) to build-in high-dependability as the product is developed, and c) using technologies to verify that the required level of dependability has been achieved.

The work presented in this paper addresses these three aspects. The work is part of the High Dependability Computing Program (HDCP) project, a NASA initiative for increasing dependability of software based systems. (See also [16]). The HDCP project proposes and investigates achieving high dependability by introducing new technologies that are developed by researchers at several participating universities and research centers.

Our (University of Maryland and Fraunhofer Center Maryland) particular research focuses on the evaluation of the effectiveness of the proposed technologies in different contexts with respect to dependability. Our evaluation takes into account these techniques’ direct and indirect effects on dependability, as well as the cost of applying them. For this purpose, we are employing empirical evaluation methods along with the concept of multilevel evaluation testbeds (described in section 3). In this paper we illustrate our technology evaluation approach by describing a testbed that is representative of air traffic control applications. The dependability model developed with the help of our dependability framework and corresponding tool is used for fault seeding. The technology that is evaluated offers a method for identification of architectural violations.

2. Modelling Dependability

Software dependability has been a research topic for the computing community for several decades; however, a common definition of dependability is still missing.

The IFIP WG-10.4 [9] defines dependability as “the trustworthiness of a computing system that allows reliance to be justifiably placed on the services it delivers.” Since “reliance” is contextually subjective and depends on a particular stakeholder’s needs in different circumstances, our focus is on different properties of services such as availability, performance, real-time response, ability to avoid catastrophic failures, capability of resisting adverse conditions and prevention of deliberate intrusions.

Dependability can be considered an umbrella for other system attributes such as reliability, availability, safety, survivability, maintainability, and security. Achieving systems dependability is a major challenge and has spawned many efforts, both at national and international level. Some examples are the European Dependability Initiative [6], the US Government strategy “Trust in cyberspace” [18] and the Critical Infrastructures improvement and protection initiatives adopted by various countries [13,19].

Various definitions of dependability are suggested in the literature [3,8,10,14]. Rather than stating another definition of dependability, we identify a common framework for modeling the dependability of a system. The framework will allow a dependability modeler to capture each individual stakeholder’s specific dependability needs in a uniform way.

2.1. A framework for modeling dependability

There are many different attributes of dependability, and each attribute is defined in several different ways in the literature. To start our analysis, let us take into consideration the following examples of dependability attributes:

- Reliability is an index of how often the system or part of it fails
- Accuracy is the ability of the system to provide data within the desired range and with the required precision
- Availability is the degree to which a system or component is operational and accessible when required for use
- Security is the capability of the system to resist abuse and disaster
- Safety is the absence of catastrophic consequences on the user(s) and the environment caused by the system or a component
- Maintainability is the ability to undergo repairs and modifications.

Based on the above definitions, we observe that dependability can be viewed as an index of the issues that the system can cause to the users. In other terms, given two similar systems, the one that causes fewer and less severe issues is the one considered more dependable by its users. In particular, based on the above definitions, we recognize two kinds of issues:

- Failure is any departure of the system behavior from the one expected by users (see, for example, definitions of reliability and accuracy).
- Hazard is a state of the system that can lead to catastrophic consequences for the user(s) and the environment (see definition of security).

From the above definitions (see, for example, definitions of reliability and availability), we can also observe that the issues caused to the users by a system could result from the misbehavior of the whole system or of part of it, for example, a service or component. Thus, we can characterize an issue in terms of the part of the system that it affects (scope), for example, by defining the scope as:

- The (whole) system, or
- A service (a functionality delivered by the system).

Then, we recognize that some failures are the results of some external events (see, for example, the definition of security). Thus, the concept of **external event** emerges as another common and independent item across the different definition of dependability attributes. Each dependability attribute can in fact be defined in terms of some kind of issues (failure or hazard) affecting the whole system or part of it (the scope), due or not due to some external events.

So far, we have introduced three main building blocks of dependability: the ISSUE (failure and or hazard) that is a threat to dependability, the possible EVENT that may cause the issue and the part of the system (SCOPE) affected by the issue. These three parameters provide us with a generic framework for building qualitative definitions of dependability, or, in other terms, to specify the failures and the hazards that we do not want to occur.

Although useful this is only partly valuable, given that failures and hazards will always be likely to happen. For this reason, it is important to introduce the possibility of expressing a measure of dependability, or, in other terms, to specify the tolerable manifestations for the identified failures and the hazards. This allows the stakeholders not only to identify the undesired failures and hazards for the system or a specific service, but also to quantify what they assume to be tolerable corresponding manifestations. The framework can easily address such a need. It is in fact possible to introduce the concept of measure as another basic element of the framework, an item whose value defines the manifestation of the issue. The measure can be expressed, for example, as the probability of occurrence of the failure over the next time unit (hours, minutes, or seconds) or over the next transaction.

Up to now, we have used the framework to specify "negative" non-functional requirements, i.e. to specify undesired system behavior, as a whole, or while delivering specific services. The framework can be extended to enable the stakeholder to provide ideas about how or by what means dependability can be improved. In other words, while expressing his/her view of dependability in terms of acceptable manifestations of failures and hazards, the stakeholder may want to specify also how the system should react to the issue in order for it to be more dependable from his/her point of view. The concept of reaction is thus introduced as another basic item, through which the stakeholder can describe the desired system behavior in case of occurrence of the issue.

Our final dependability framework consists of the five main parameters EVENT, ISSUE, SCOPE, MEASURE and REACTION. Figure 1 shows the relation between these parameters.

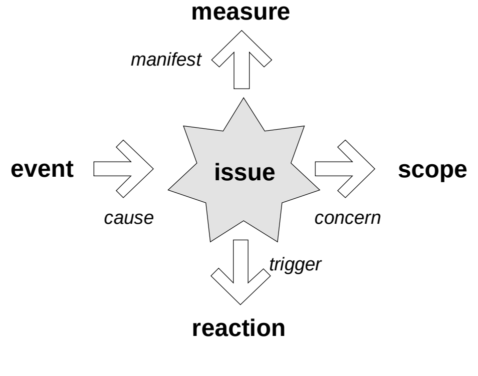
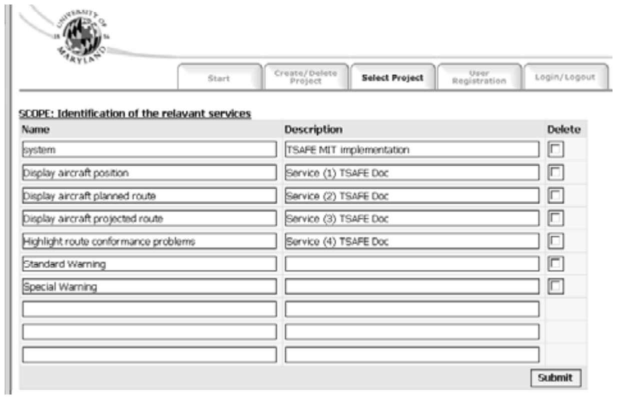


Figure 1 – The dependability modeling framework

2.2. An implementation tool for the framework

A Web-based tool that implements the framework has been developed. Details about the tool are provided in [1]. In the following we provide a brief overview of the tool's two main table frames, which collect data from the user.

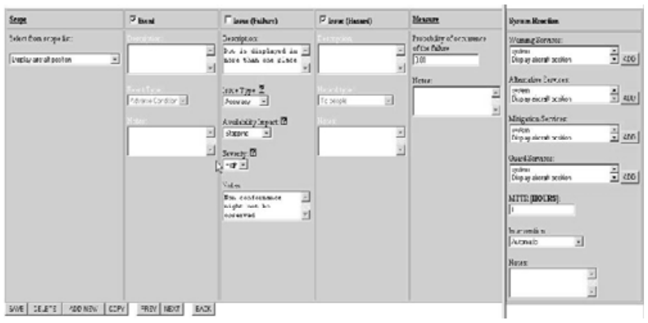
The **Table "Scope"** allows the stakeholder to identify all the services of the system for which dependability could be of concern. For the system and each identified service, the stakeholder has to provide an identifier (left column), and a brief description (right column).



SCOPE: Identification of the relavant services

Name	Description	Delete
system	TSAFE MIT implementation	☐
Display aircraft position	Service (1) TSAFE Doc	☐
Display aircraft planned route	Service (2) TSAFE Doc	☐
Display aircraft projected route	Service (3) TSAFE Doc	☐
Highlight route conformance problems	Service (4) TSAFE Doc	☐
Standard Warning		☐
Special Warning		☐

The **Table Frame "Issue"** allows the users to specify their dependability needs by identifying potential issues (failures and/or hazards), their tolerable manifestations, the possible causing events, and the desired system reactions.



Select from scope list: Display aircraft position

Description: Pos is displayed in more than one place

Availability Impact: stopping

Probability of occurrence of the failure: 0.01

Warning Services: system Display aircraft position

Alternative Services: system Display aircraft position

Mitigation Services: system Display aircraft position

Guard Services: system Display aircraft position

MTTR (HOURS): 1

Intervention: Automatic

3. Technology evaluation testbeds

We view each new technology that contributes to increasing dependability as maturing and passing through a series of evaluation milestones, each demonstrating its context of effectiveness [15]. These milestones are:

1. Internal set: Typically, internally developed set of examples on which the technology researcher has applied the technology.
2. Milestone 2. Basic common set: A basic set of common examples (testbeds) on which to stress and analyze technologies, both individually and in groups.
3. HDCP domain-specific off-line set: Domain-specific set of examples, chosen from past NASA or NASA-contractor projects or from HDCP company projects.
4. 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.

The evaluation milestones apply to all experiments in HDCP. The experiment described in this paper brings technologies from milestone 1 to milestone 2 using the Tactical Separation Assisted Flight Environment (TSAFE) testbed. TSAFE is a new system designed to aid air traffic controllers in detecting and resolving short-term conflicts between aircrafts. Conceived at NASA Ames Research Center, TSAFE was proposed as a component of a larger Automated Airspace Computing System, whose goal is to shift the burden of maintaining aircraft separation from controllers to computers. The TSAFE prototype that we are using as a testbed was developed at MIT [5]. It consists of 20,000 lines of Java code and performs two primary functions: conformance monitoring and trajectory synthesis.

4. Architecture evaluation technique

Many of the technologies in the HDCP project deal with the system's architecture, which is one of the major reasons we decided to start experimenting with this class of technologies. Software architecture deals with the structure and interactions of a software system. The most basic building blocks of the structure of software architecture are components and the interrelationships among them. In addition to structure, behavior is part of software architecture. Constraints and rules describe how the architectural components communicate with one another.

A system's software architecture may be viewed at different levels for different purposes. When viewed at the highest levels, a system's architecture is referred to as the macro-architecture of the software system. At lower levels of abstraction, it is referred to as micro-architecture. Architectural styles and design patterns are similar to what Bhansali [2] describes as generic forms of software architecture. Often architectural styles guide the structure and interactions of the system when describing the system's software architecture at the macro-architectural level. When describing the structure and/or interactions of a system at a micro-architectural level, design patterns can be used.

Software architectural evaluations are investigations of a system's structure and behavior, with the purpose of suggesting areas for improvement or understanding various aspects of a system (e.g., maintainability, reliability, or security). In many cases, a software architectural evaluation is performed before a system has been implemented. Often, this type of evaluation is performed to compare design alternatives or to determine whether or not the architecture is complete or appropriate for the application. In other cases, a software architectural evaluation is performed after the system has been implemented. This type of architectural evaluation typically is performed to assure that the actual implementation of a system matches the planned architectural design [17]. Some of the technologies in HDCP evaluate architectures before implementation, some after. The technology discussed in this paper is of the latter kind and is called *Implementation-oriented software architectural evaluation*. Since this type of software architectural evaluation is performed after a version of the system exists, it can utilize data measured from the actual source code and associated documentation. For example, the source code and associated documentation can be used to reconstruct the actual software architecture in order to compare it to the planned or conceptual software architecture. Recovering the actual architecture of an implemented system is used for risk assessment and maintenance cost prediction as well. The analysis of the actual software architecture can be used to evaluate whether the implemented software architecture fulfills the planned software architecture and associated goals, rules and guidelines.

Violations of architectural guidelines affect the maintainability of the system, which analysis of Mozilla showed. The conclusion was "that either its architecture has decayed significantly in its relatively short lifetime, or it was not carefully architected in the first place" [7].

This is not a new problem. In 1969, Lehman found that "the main problem of large systems is unintentional interaction between components, that require changes to the components for their elimination" [11]. The observations were formed into a collection of "laws" stating, for example, that: "As a program is evolved, its complexity increases unless work is done to maintain or reduce it" [12]. Brooks added: "all repairs tend to destroy the structure, to increase the entropy and disorder of the system. Less and less effort is spent on fixing original design flaws; more and more is spent on fixing flaws introduced by earlier fixes. As time passes, the system becomes less and less well-ordered" [4]. Brooks drew the conclusion that all systems will eventually require a complete redesign as a consequence of this degeneration [4]. Maintainability is an important dependability attribute because it affects the mean time to repair (MTTR) related to the time needed to fix a defect once it has been discovered. If the maintainability of a system is low then MTTR will be high and the dependability of the system will decrease.

A technology able to detect architectural violations increases maintainability of the system and thus increases the dependability of the system.

5. Experiment

We are instrumenting the MIT version of TSAFE and using it as a platform for designing and executing the following dependability-related experimental activities:

- Define what dependability means for MIT TSAFE, by applying the UMD model described in section 2
- According to this definition of dependability, identify potential failures and corresponding faults in the code that could cause these failures
- Identify inputs for executing the software such that these faults are exercised and cause failures
- Seed the code with the identified faults
- The technology developers formulate their hypotheses in terms of faults that can be detected by their technology (and possibly the impact on dependability) and estimate the costs associated with applying the technology
- Apply the technology and record the detected faults as well as the cost of its application.
- Execute the system (containing these seeded faults) and record the occurring failures (as well as their frequency of occurrence)
- Analyze these failures in the context of dependability defined in the first step and validate or refute the technology hypotheses
- Estimate the effect on dependability and the cost of this technology for the MIT TSAFE testbed

The outcome of these activities will be:

- A method for defining dependability
- A method for designing and performing experiments for estimating the effect of a technology with respect to dependability and cost, for a given system.
- An example for the application of this method
- An instrumented testbed for future experiments

The first experiment will consist of applying and evaluating the architecture evaluation technique. The goal of the experiment is to determine how the technique contributes to increased dependability by detecting faults. The first step of the experiment is the characterization of the technique, which involves defining classes of faults that the technique will detect and formulating the techniques' hypotheses regarding other benefits gained when applied. We will seed the testbed with faults for this purpose. The need for fault seeding is due to the relatively good shape of the test bed and the need to know how many faults reside in the system. The current version of the testbed is a baseline to which we compare faulty versions of the system. We identified faults to be seeded based on two strategies: 1) based on failures, 2) based on architecture-oriented faults. We conducted a hazard analysis in which we identified a set of likely system failures derived from the dependability model described earlier. For each of the failures, we identified a set of plausible faults, making the link between failures and faults explicit. We defined a set of fault candidates that are related to the architecture of the system. For each fault, we identified the impact on the system's behavior and how failure would be detected. These faults will be seeded into the system. We are also in the process of instrumenting the testbed so that it becomes a testbed in such a way that it will be possible to determine the internal operational behavior based on traces.

7. Future work

The future work is to complete the testbed, run the experiment, and analyze and interpret the results. If necessary, we will then improve the experiment, the testbed, and possibly even the technique. The process will then be repeated using the same technique for SCRover, a testbed developed at University of Southern California. We will also apply other techniques such as code inspection and determine their ability to detect defects in different situations on different testbeds. Based on these results and the documented links from these faults to failures, we will be able to reason about the impact of these technologies on dependability.

Acknowledgements

The authors acknowledge support from the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298. We also want to thank our HDCP team members at University of Southern California lead by Dr. Barry Boehm, Winsor Brown, and Alex Lam. We thank Jen Dix for proof reading the paper.

References

[1] Basili, V., Donzelli, P., Asgari, S. Modelling Dependability – The Unified Model of Dependability, University of Maryland Technical Report, April 2004 (forthcoming).

[2] Bhansali S., Software Design by Reusing Architectures. Knowledge-Based Software Engineering Conference, McLean, Virginia September 1992.

[3] Boehm, B., Huang, L., Jain A., Madachy, R., The Nature of Information System Dependability – A Stakeholder/Value Approach - USC Technical Report November 2003.

[4] Brooks, F. P., The Mythical Man-Month, Addison Wesley, 1995.

[5] Gregory, D., TSAFE: Building a Trusted Computing Base for Air Traffic Control Software, Masters Thesis, MIT, 2003.

[6] European Initiative on Dependability: towards dependable Information Infrastructures http://www.cordis.lu/ist/ - http://deppy.jrc.it.

[7] Godfrey, M. W. and Lee, E. H. S., Secrets from the Monster: Extracting Mozilla's Software Architecture, Symposium of Constructing Software Engineering Tools (CoSET00), 2000.

[8] Huynh, D., Zelkowitz, M., Basili, V., and Rus, I., Modeling dependability for a diverse set of stakeholders, Distributed Systems and Networks Workshop 2003, San Francisco, CA, June 2003.

[9] International Federation for Information Processing (IFIP WG-10.4), www.dependability.org.

[10] Laprie, J., Dependability: Basic Concepts and Terminology, Dependable Computing and Fault Tolerance, Vienna, Austria, Springer-Verlag, 1992.

[11] Lehman, M. M. and Belady, L. A., Program Evolution: Processes of Software Change, London: Harcourt Brace Jovanovich, 1985.

[12] Lehman, M. M., Laws of Software Evolution Revisited, European Workshop Software Process Technology, 1996.

[13] Moteff, J., Copeland, C., Fisher, J., Critical Infrastructures: What Makes an Infrastructure Critical?, Report for Congress RL31556, The Library of Congress, 21 January 2003.

[14] Randel B., Dependability, a unifying concept, Computer Security, Dependability and Assurance: from needs to solutions Workshop, York, UK, July 1998.

[15] Rus, I., Basili, V., Zelkowitz, M., and Boehm, B., Empirical evaluation techniques and methods used for achieving and assessing high dependability, Workshop on dependability benchmarking, International Conference on Dependable Systems, Washington, DC, June, 2002.

[16] Scherlis, W.L., Dependability and Architecture: An HDCP Perspective, ICSE'02 Workshop on Architecting Dependable Systems, Orlando, FL, May 2002.

[17] Tesoriero Tvedt, R., Costa, P., and Lindvall, M., Does the Code Match the Design? A Process for Architecture Evaluation, Proceedings of the International Conference on Software Maintenance, 2002.

[18] U.S. The National Strategy to Secure Cyberspace, February 2003; http://www.whitehouse.gov/pcipb.

[19] Wenger, A., Metzger, J., Dunn, M., (editors) International CIIP Handbook, ETH, Swiss Federal Institute of Technology Zurich, 2004.