NPS-AM-14-C11P07R03-038

PROCEEDINGS
OF THE
ELEVENTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

WEDNESDAY SESSIONS
VOLUME I

Combining Risk Analysis and Slicing for Test Reduction in Open Architecture

Valdis Berzins
Naval Postgraduate School

Published April 30, 2014

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

The research presented in this report was supported by the Acquisition Research Program of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact any of the staff listed on the Acquisition Research Program website (www.acquisitionresearch.net).

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Panel 7. Challenges of Software Development in an Open Architecture Environment

Wednesday, May 14, 2014	
1:45 p.m. – 3:15 p.m.	**Chair: TBD** ***Achieving Better Buying Power Through Cost-Sensitive Acquisition of Open Architecture Software Systems*** Walt Scacchi, University of California–Irvine Thomas Alspaugh, University of California–Irvine ***Analyzing Quality Attributes as a Means to Improve Acquisition Strategies*** Lisa Brownsword, Carnegie Mellon University Cecilia Albert, Carnegie Mellon University Patrick Place, Carnegie Mellon University David Carney, Carnegie Mellon University ***Combining Risk Analysis and Slicing for Test Reduction in Open Architecture*** Valdis Berzins, Naval Postgraduate School

Combining Risk Analysis and Slicing for Test Reduction in Open Architecture[1]

Valdis Berzins—is a professor of computer science at the Naval Postgraduate School. His research interests include software engineering, software architecture, reliability, computer-aided design, and software evolution. His work includes software testing, reuse, automatic software generation, architecture, requirements, prototyping, re-engineering, specification languages, and engineering databases. Berzins received BS, MS, EE, and PhD degrees from MIT and has been on the faculty at the University of Texas and the University of Minnesota. He has developed several specification languages, software tools for computer-aided software design, and fundamental theory of software merging. [berzins@nps.edu]

Abstract

The Navy's open architecture framework is intended to promote reuse and reduce costs. Relevant cost components include both planned test and evaluation effort and possible future failures of deployed software. Pairing system risk analysis using fault trees with NPS research on dependency analysis should enable these benefits and keep resource requirements at feasible levels.

This paper describes methods that evaluate the risk impact of software components. The approach combines system risk analysis, dependency analysis applied to source code, and automated testing applied to executable component implementations. Determining the level of reliability required for each software component to achieve acceptably low system risk exposure is a key concern in this context, since the number of test cases required can be derived from this information.

The paper applies open architecture principles to reduce testing effort and costs by finding the minimum testing effort required to reduce exposure to identified risks. When combined with our previous work on safe test reduction in cases where the requirements and code for a subsystem have not been changed, but the component will be used together with new or modified components, or may be reused in a different context with different operational loads and different system risk exposures, the results will enable further safe reductions in testing costs. These situations are common in the Navy due to technology advancement upgrades and policies that emphasize component reuse across systems.

Introduction

Open architecture seeks to reduce costs by enhancing competition between vendors and reducing duplication of effort. The first is done by breaking large systems into networks of simpler subsystems defined by open standards and interfaces, thus creating a market for "plug-compatible" components, which can be either software or hardware. The second is done by reusing such subsystems across multiple platforms. In both cases, quality assurance is one of the bottlenecks. Our research has been aimed at alleviating this bottleneck, focusing on improving methods for regression testing because this area has the greatest potential for reducing costs due to duplication of test and evaluation effort. The key is to reduce costs safely, that is, without increasing operational risk due to fielding of new or upgraded systems.

[1] This research was supported in part by the Acquisition Research Program under project F13-015 and by the N1 studies program under task P14-0626.

According to current Navy procedures, each new release of a system must be completely retested before being released to the field. These full regression testing procedures are used to reduce the risk of operational failures due to system faults that could be introduced by the engineering changes comprising a technology upgrade. These upgrades typically happen every two or four years for each ship, depending on the type of platform, and regression testing accounts for a major part of the cost of each upgrade. Such tests include full-system testing, whose purpose is to check that all the parts work harmoniously together. This part of the effort cannot be reduced unless it is replaced by new and more effective means to ensure the absence of such faults. Even with current procedures for full-system testing, the incidence of expensive "system integration problems" is higher than program managers and the public would like.

Our previous work has focused on two cases:

1. Neither specification nor the code of a software module has changed from the previous release. This case is of interest because under certain conditions that can be checked by software it is safe not to repeat functional testing for such modules.
2. The specifications have not changed but the implementation depends on code that has changed. In this special case the regression testing can be completely automated relatively easily and can affordably achieve test sample sizes large enough for high levels of statistical confidence that the behavior of the module has not changed from what it was in the previous release if the component has been updated correctly, and for detecting faults if it has not.

A road map for the overall approach is in [1] and details can be found in [2] [3] [4] [5].

The work reported in this paper addresses situations where the specification of the component is changed in the new release, which includes cases where new functionality has been added as well as cases where the architecture has been extended to include components that were not present in the previous release, or the component is part of a completely new system that has no previous release. In this case automated testing takes additional set up effort because it is necessary to implement software that checks whether the new requirements are satisfied for given input values and the outputs actually produced by the software under test when run on those inputs. After this fixed set up procedure has been completed, the incremental cost of additional test cases will be very small, and test sets large enough to provide high levels of statistical confidence will be affordable.

Regression testing provides the simplest context in which we can transform testing procedures from a test-to-fit approach that must recheck the collaboration of the parts of a stove-piped system all over again for each new configuration, even if none of the parts may be new, into a new test-to-standards approach that supports software construction with interchangeable parts that do not have to be completely recertified for each use.

The motivation for such a transformation is to reduce duplication of effort. Potential benefits of such a transformation can be substantial, particularly for reconfigurable systems. The set up cost of such a transformation includes the costs of corresponding changes in development methods, introducing new representations of architectures, and new quality assurance processes for the architectures and the standards associated with them. Methods for designing systems to tolerances are more difficult and labor intensive than those for designing stove-piped systems, and so are the kinds of analysis needed to show that such a design will work for any combination of components that meet their specified tolerances. The corresponding advantage is reduction in regression testing. A potentially even greater

future advantage is for systems with replaceable modular components that are intended to be swappable in the field.

This kind of modular design is getting increasingly popular in new Navy systems. For example, the Littoral Combat Ship (LCS) has replaceable mission modules. Under current procedures, each possible configuration must undergo a full regression test before it can be fielded. This is not too onerous in the case of the LCS, because there is only one optional module on the platform, which is currently planned to have three different variants. That would result in three different configurations to test, which should make the test cost proportional to the number of optional module variants, which may be acceptable. However, as confirmed by several Navy students, no two ships are configured exactly alike. Thus for reliable detection of system integration problems, regression testing would have to be repeated for each individual platform that is upgraded, rather than once per upgrade to an entire platform class. However, each LCS is supposed to be capable of doing a module swap at any port, possibly in mid mission [6]. That concept of operations does not appear to provide for a full regression test after each module swap.

The greatest regression testing difficulty arises when there are several different optional module slots on the same platform that can be independently varied. In this case, the number of configurations to test is the product of the number of options for each of the optional module slots, and the number of configurations to test increases much more rapidly with the number of options for each slot. In general, the number of configurations increases exponentially with the number of optional slots. In particular, if we have 10 optional slots and there are only two possible options for each slot, the number of system configurations to be tested would be 1024, which is very unattractive, and if each slot can be filled by a choice of 10 different optional modules, the number of system configurations to test is 10 billion, certainly cost prohibitive.

The test-to-standards approach should greatly reduce the certification effort required for reconfiguring a system via module swapping, but it may not eliminate it entirely. Testing to standards should be sufficient to ensure the module will meet its functional requirements in all possible configurations if the standards themselves can be certified to be strong enough to support this, but other issues related to timing constraints and other resource constraints (e.g., network bandwidth, memory space, absence of deadlocks, etc.) may need to be checked in separate admissibility checks for each proposed new configuration. A simple example of an admissibility check may illustrate this concern. When replacing the speakers of a stereo system, we have to check that the new speakers have a power rating no less than that of the amplifier. If the speaker has been certified with respect to stereo system standards, the updated system will meet its functional requirements (playing music) even if this check fails, but it may be subject to failures under stress (blowing out the speakers if the music has loud sections). Nevertheless, admissibility checks typically require much less effort and cost than full system recertification, and are much more likely to be practical to perform in the field. Previous work on admissibility checks related to timing constraints can be found in [7].

The work reported in this paper takes a step towards supporting the transformation described above. Specifically, we have investigated the question of how much testing is needed for adequately reducing operational risks due to possible software faults. This involves a system wide risk analysis, the effects of which are propagated to individual units under test via a software dependency analysis. This process tests the parts of the system with greater risk exposure more intensively than the parts with lesser risk exposure. The number of test cases for each part is set at a level that achieves a sufficiently high statistical confidence level. The required confidence level is set so that the probability of mistakenly

passing an unreliable software service by pure chance (due to sampling error) is no more than the maximum acceptable failure rate derived from the accepted residual risk exposure for the service. The process described and illustrated in this paper is needed to support the part of the test-to-standards approach that certifies a newly developed variant of a module against the standards and requirements of the architecture slot that the module is intended to fill. Other parts of that process [1] [3] include quality assurance for the architecture and the standards, admissibility checks to guard against possible interference between the new components and the others present in the configuration, and runtime monitoring to check that the code of the component has not been modified from the version that was certified, and to recover from such faults by restoring the desired version from read-only media that are not subject to corruption if modifications are detected. Such undesired code modifications can occur due to natural processes such as hard radiation flipping bits in the instruction memory, software faults that can corrupt memory contents such as writes through dangling pointers, or cyber-attacks that deliberately modify code for hostile purposes.

The intended principles of operation for the proposed risk based testing approach can be summarized as follows (from [8]):

1. Perform a conventional whole-system operational risk analysis, using approaches adapted from safety procedures certification such as MIL STD 882-E [9]. The result of this step is a list of potential mishap types, associated with and ranked by their risk levels. We propose to extend the definition of mishaps in this context to include various aspects of mission failures that could be induced by system failures, in addition to the types of mishaps traditionally considered in a safety certification.
2. Perform a system level dependency trace to identify which hardware subsystems affect each type of mishap listed in step 1, and which software services affect each of those subsystems.
3. Perform a software dependency analysis using software slicing, to identify which software modules affect each of the software services identified in step 2.
4. Using mishap list from step 1 and the dependency relations from steps 2–3, identify the set of potential mishaps that can be affected by each software module.
5. Associate the maximum risk level of the set of mishaps identified in step 4 with the corresponding software modules.
6. Use the risk level derived in step 5 to determine the level of testing and possibly levels of additional risk mitigations to be associated with each software module.

Software Dependency Analysis

Program slicing [10] is a type of dependency analysis that eliminates program statements irrelevant to a given slicing criterion. Slicing algorithms detect and follow dependencies of the kinds described above. A survey of known slicing algorithms can be found in [11]. Typical slicing criteria limit attention to the software behavior visible from a particular observation point, such as given output from the system or the result produced by a given software service. A slice includes all of the code that can affect the behavior relevant to the behavior selected by the slicing criterion. In the context of risk-based testing, the services that can affect a given system hazard can be identified by checking their

membership in particular slices. Details are described in [8] [12]. Evaluation of commercially available tools with respect to this context can be found in [8] [13].

Risk Analysis

MIL-STD-882E is the standard for the "management of environmental, safety, and occupational health mishap risks encountered in the development, test, production, use, and disposal of Department of Defense (DoD) systems, subsystems, equipment, and facilities" [14]. A key word in the description above is "management." The standard does not provide low-level details on the analysis methods used to determine the risks encountered, only the requirements for analysis and the way the risks are to be managed and focuses on the system as a whole. Risk is comprised of two components, mishap severity and probability. This standard treats risks related to software separately from those related to the other parts of the system, due to the perception that probability of software failure is a metric that is impractical, if not impossible to measure. One of the difficulties that leads to this perception is that for systems with a high degree of reliability, it may not be affordable to collect and assess test samples large enough to demonstrate any failures at all, let alone a large enough number to enable accurate estimates of the failure probability.

We utilize a method for bounding such software failure probabilities to high levels of statistical confidence via automated testing. Our method gets around the difficulty noted above by focusing on obtaining a statistical upper bound on the failure rate, without seeking to determine its exact value. Such a bound can be obtained from a sufficiently large number of observations that are completely failure-free. This method opens up the possibility of systematic design to meet large scale reliability requirements by means of failure budgets that can be allocated to different subsystems [15], much in the same way as chip area budgets are allocated to various subsystems in the design of VLSI chips. Similar hardware design methods are used by NASA to control risks for their spacecraft. Corresponding methods for software design are relevant to achieving reliability properties of software architectures, which should be of interest to the Naval Open Architecture community.

The failure rate bounding approach described in the current paper and in [2] bounds the failure rate of a software component with respect to an "operational profile," that is, a probability distribution that characterizes the frequency of its possible inputs in a given execution environment. This is necessary because software reliability depends strongly on the operating environment as well as on the properties of the software—for any software component that operates correctly sometimes but is not 100% perfect (all practical systems), there exist operating environments in which the reliability of the component matches any value arbitrarily chosen from the range 0% to 100% [2]. Thus reliability assessment must be done with respect to operating environments expected to occur in practice. Methods for determining such probability distributions based on historical data recorded from system use are described in [4]. To use this result for risk analysis, we need to bridge the failure rate bounded in [2] , which is relative to the number of executions of the component, to the failure rate usually required in risk analysis, which is the failure rate per unit of time. The conversion factor between the two kinds of failure rates can be determined by measuring the average number of executions of the software component per time unit. This can be done by monitoring the number of component executions in the system for a sufficiently long measured time interval during training exercises or actual operation.

Specific risk analysis methods can be found in ARP4761 [15] , which is an Aerospace Recommended Practice from the Society of Automotive Engineers International. Despite being a recommended practice, it is commonly considered a de-facto standard. This standard was primarily created as a means to comply with the Federal Aviation Regulation

(FAR) 25.1309 [16], which is a federal airworthiness requirement for the safe and expected operation of equipment, systems and installations. ARP4761 provides a systematic means of performing safety assessments across the whole aircraft and its sub-components. The standard describes a number of safety analysis techniques (Functional Hazard Analysis, Fault Tree Analysis, etc.) that can be utilized. Some of these techniques are similar, e.g., Fault Tree Analysis (FTA) can be replaced by Dependence Diagrams (DD) or Markov Analysis (MA).

Due to time limitations, the case study described next focused only on the system level Functional Hazard Analysis (FHA) and corresponding FTAs for the design to illustrate the proposed risk analysis method, rather than producing a complete and thorough system safety report.

Drone Risk Case Study

A case study was carried out [12] to provide an example of the proposed risk based testing methodology described above. Only a Preliminary Hazard Analysis was conducted, but this should be sufficient to demonstrate the proposed methodology. The preliminary hazard analysis was limited to product risks only (risks to the drone). Risks to the surrounding air and space, such as collision with commercial aircraft, ground vehicles or people were not considered. Those could be added to the analysis using the same methods, although that was not done in the current case study due to schedule and resource limits.

The system that was examined is the Parrot AR.Drone 2.0, a commercially available quad-rotor toy that can be controlled via WiFi using Apple, Android devices or a PC. An image of the item is shown in Figure 1 [17]. Despite being a toy, it features many advanced features such as image recognition for augmented reality games. This quad-rotor was chosen due to the Software Development Kit (SDK) being open source and not subject to restrictions.

Figure 1. AR Drone 2.0
(from [17])

The hazards that were identified for analysis are as follows:

- Loss of communication
- Loss of propulsion
- Environmental Damage
- Loss of battery power
- Loss of situational awareness

The hazards were analyzed for their assessed risk levels, relative to the risk matrices adapted from examples in MIL-STD-882E for the case study and reproduced in Tables 1–3 below. FTA for each hazard was also carried out. The severity matrix for the case study was chosen based on cost, legal ramifications and potential damages incurred, and scaled according to the assumed context.

Table 1. Case Study Severity Matrix

Catastrophic	Could result in irretrievable loss of aircraft, human life, damages exceeding $1,000, or irreversible severe environmental damage that violates law or regulation.
Critical	Permanent damage to retrievable aircraft requiring complete replacement, permanent partial disability, loss exceeding $1,000 or reversible environmental damage causing a violation of law or regulation or indirectly causes mission failure.
Marginal	Major reparable damage to aircraft requiring replacement of expensive parts with total repair cost exceeding $150, non-permanent injury or mitigate-able environmental damage causing a violation of law or regulation.
Negligible	Minor reparable damage to aircraft requiring replacement of cheap parts with total repair not exceeding $150, or minimal environmental damage not violating law or regulation.

The probability matrix was chosen based on expected failure rates for an aircraft. Since no historical data was available as commercial organizations are reluctant to reveal failure data, only a subjective analysis could be reasonably performed. There are therefore no specific reliability ranges are used as criteria.

Table 2. Case Study Probability Matrix

Frequent	Expected to occur multiple times in the operation of the aircraft.
Probable	Will be expected to occur at least once in the operation of the aircraft.
Occasional	Will be expected to occur at least once after several operations of the aircraft.
Remote	Unlikely, but can be expected to occur at least once in the life of the aircraft.
Improbable	Highly unlikely to occur, but still possible to occur at least once in the life of the aircraft.

The risk matrix was adapted by combining Table 3 and Table 4 from MIL-STD-882E:

Table 3. Case Study Mishap Categories

	Catastrophic	Critical	Marginal	Negligible
Frequent	High	High	Serious	Medium
Probable	High	High	Serious	Low
Occasional	High	Serious	Medium	Low
Remote	Serious	Medium	Medium	Low
Improbable	Medium	Medium	Low	Low
Designed out	Low	Low	Low	Low

Loss of Communication

Drones and Ground Control Stations (GCS) are connected via an 802.11b/g wireless network, with a range normally of up to approximately 100 meters. This network includes connections between ground station and each drone and also among the drones via an ad-

hoc setup (the default setting). Infrastructure mode can be forced by uploading a script to the drone via telnet once an ad-hoc connection is established.

Loss of Communication may arise from a problem associated with either the ground station, drone or inter-drone communication, or the external communications channel.

Ground Control System Communication Problems

Communication problems associated with the GCS may arise from the following events:

- *Improper Protocol*
  Connections with the drone are established using UDP and TCP with specific configurations. Deviating from the configurations (e.g., using a different message block size) will be classified as improper protocol.
- *DHCP Client/Server*
  The drone is designed to provide IP addresses to its controllers automatically using a DHCP Server. Any update on either DHCP client or server may result in unexpected behavior. The allocation of IP addresses can be an issue if drones or other GCSs are added at a later time, as some drones may be configured with static IPs that may previously have been assigned.
- *Wireless Network Interface Card (NIC)*
  There may be a physical problem in the hardware responsible for converting electrical signals to data flow. The NIC is susceptible to physical damage and has a reliability value like any other hardware.
- *Antenna*
  A physical failure of the antenna connected to the NIC will yield poor or no connection.
  The severity of the loss of communication due to GCS events previously described is considered *catastrophic* as the operator will no longer have control of the aircraft. The aircraft is programmed to slowly descend for landing when it loses communications with the GCS. While this in principle should prevent damage to the aircraft, damages can still be incurred based on the environment that the drone lands in (e.g., a busy highway). Since problems with the GCS should be picked up on take-off, the likelihood of failure is considered *improbable* thus giving it a risk of *medium*.

Drone Communication Problems

Communication problems associated with the drone may arise from the following events:

- *Out of Range*
  The drone is assumed as out of range when it no longer has effective communications over the WiFi connection due to signal attenuation. The effective range of a WiFi connection is affected by multiple variables, such as humidity, encryption algorithm, other radio signals and obstacles.
- *Improper Event Handling*
  The drone Operating System (OS) might give a trivial and blocking process higher priority that would prevent the drone's ability to respond to ground station in a proper and timely manner.

- *DHCP Client/Server*
  As discussed in section titled Ground Control System Communication Problems.
- *Antenna*
  The drone's antenna is susceptible to physical failures due to weather conditions and vibration experienced in flight.
  The severity of the loss of communication due to drone events is considered *catastrophic* due to the same reasons as the loss of communication due to the GCS. However, the likelihood of the aircraft going out of range is not rare and is considered an *occasional* occurrence. This gives the failure mode a *high* risk.

External Communication Problems

- *Radio Interference*
  This is an external source that can result from intentional jamming or being in a densely populated area. The WiFi frequency range (bandwidth) is limited but because it does not require a license to operate in these frequencies, many other products (such as cordless phones and wireless mice) utilize it. Radio interference can therefore also occur from saturation of the area due to an excessive number of devices using the bandwidth.

  The severity of the loss of communication due to radio interference is considered *catastrophic* for the same reasons as previously mentioned. The frequency saturation of the airspace can be assessed on takeoff based on the signal strength. However, the operator has no control on other devices coming and leaving the affected area and therefore the likelihood of this event occurring is considered *remote* thus giving a risk of *serious*. Combining these failure modes as per the FTA of Figure 2, the overall the risk for the loss of communication is assessed as *high*.

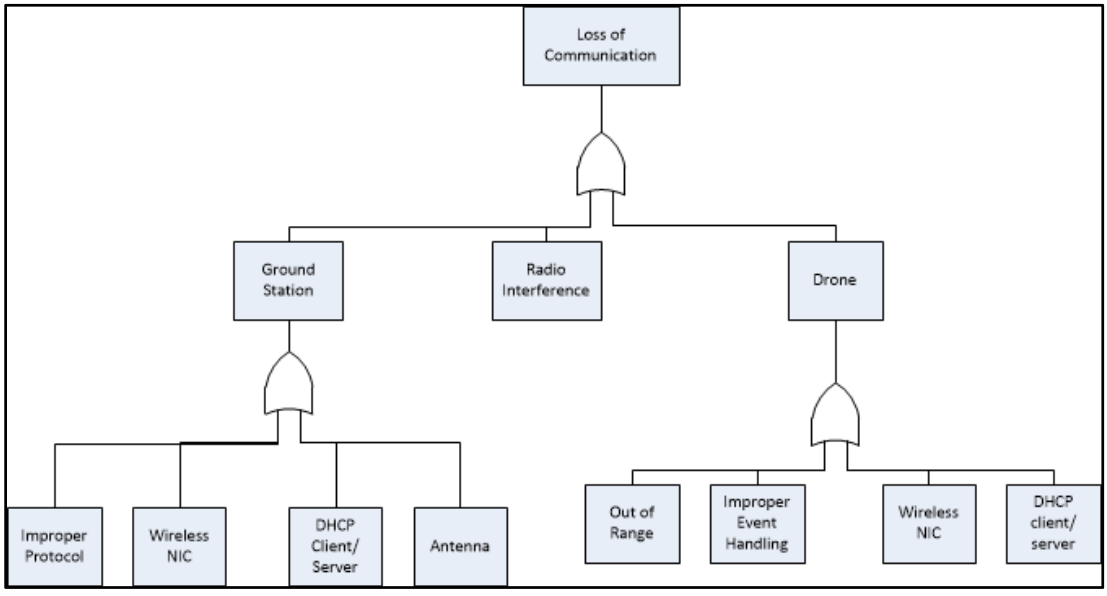

Figure 2. FTA for Loss of Communication

Risk Analysis Summary

Details of the analysis of the other risk factors can be found in [12]. The results of the analysis are summarized in Table 4.

Table 4. System Risk Summary

Hazard	System Risk
Loss of Communication	High
Loss of Situational Awareness	High
Loss of Propulsion	Serious
Loss of Battery Power	Medium
Environmental Damage	Medium

As shown in the above table highest risks are the loss of communication and situational awareness. The main contributor to these risks is the event where the aircraft goes out of range. According to potential influence on this event, the software modules of utmost importance are those that affect the wireless connectivity. These would be identified by an analysis of the design to determine which top-level software services can affect wireless connectivity. Software slicing with respect to slicing criteria corresponding to the results of those services would then be used to identify the lower level modules that can affect those services. An initial attempt at demonstrating such an analysis can be found in [12]. This demonstration is incomplete due to gaps in software tool support.

In practice, additional risk mitigation measures, such as design changes that prevent occurrence of certain states or events, would be added after a preliminary risk analysis like the one done in the case study, to reduce exposure to risks above a risk exposure threshold set by the program manager, and the analysis repeated to assess the severity of residual risks—those remaining after the mitigations have been applied. After that, maximum acceptable failure rates would be set for the hazards of greatest severity, as needed to reduce residual risk exposure to levels acceptable to the risk acceptance authorities defined by MIL-STD-882E. These maximum acceptable failure rates would then be used to determine the minimum number of test cases needed for an acceptance test. In such a context, all test cases must pass without the observation of any failures in order for the system to be certified as having acceptable risk.

Conclusion

This paper demonstrates risk-based processes for achieving safe and affordable software systems in the context of open architectures. These provide a systematic way to answer to the question “how much testing is enough.” Instead of testing until schedule and budget are used up, it determines how many test cases are needed based on a risk analysis that identifies the worst-case impact of a failure of each software component. Impact is determined by severity of potential consequences in terms of damage to people and property as well as mission failure. The results of the analysis are likely different for different components, providing a quantitative process that realizes the common sense guideline to test the mission- and safety-critical modules more thoroughly than those whose results have less impact.

We are also conducting ongoing research on affordably reducing system integration problems for systems with open architectures. This is a continuing and very expensive problem that has almost as much public attention as chronic cost and schedule overruns in software development projects. One of the preliminary results of our study is the conclusion that most system integration problems are symptoms of faults in the architecture. Some of

the system integration issues can be illustrated by the following simple examples from the domain of architectures for houses:

1. The kitchen plan calls out a Miele microwave oven and an electric outlet. The electrical contractor installs a 110 volt outlet. When the oven is delivered, the accompanying installation guide requires a 220 volt power supply. The oven will not work with the existing interface. The problem is that the architecture did not specify all the constraints needed to ensure the subsystems will work together harmoniously (in this case power supply voltage).
2. The laundry plan calls out an electrical outlet, water supply, and drain for the washer and an electrical outlet, gas supply, and air vent for the drier, and a big window on top of both machines. The plumber installs the water supply, drain and gas pipes in the limited space below the structural members supporting the window. When the electrical contractor arrives, he finds that all space that could be used for the electrical outlets has been obstructed by the pipes. The problem here is that the architecture did not provide enough detail to deconflict resource requirements for the subsystems (in this case volumes of physical space).

Our recommendations based on these preliminary findings are that architecture descriptions and associated standards should be a required deliverable in all contracts for systems with open architectures, and that these should be required to pass quality assurance reviews specifically targeted at preventing system integration problems. Such reviews should at a bare minimum check if the architectures and standards are strong enough to strong to ensure harmonious operation of all subsystems that conform to the subsystem specifications associated with the architecture slots (not just harmonious operation of the specific subsystem versions expected in the first release), and that they are strong enough to deconflict resource constraints (such as power, weight, network bandwidth, memory size, processor speed, etc.).

Our ongoing research is aimed at developing more systematic and effective methods for reducing system integration problems. Results will be reported in future publications.

References

[1] V. Berzins, M. Rodriguez and M. Wessman, Putting Teeth into Open Architectures: Infrastructure for Reducing the Need for Retesting, *In Proceedings of the Fourth Annual Research Symposium* (pp. 285–312), 16–18 May 2007.

[2] V. Berzins, Which Unchanged Components to Retest after a Technology Upgrade, *Proceedings of the Fourth Annual Research Symposium – Acquisition Research: Creating Synergy for Informed Change*, pp. 142–153, 14–15 May 2008.

[3] V. Berzins and P. Dailey, How to Check If It Is Safe Not to Retest a Component, *In Proceedings of the Sixth Annual Research Symposium – Acquisition Research: Defense Acquisition in Transition*, pp. 189–200, 12–14 May 2009.

[4] V. Berzins and P. Dailey, Improved Software Testing for Open Architecture, *Proceedings of the Seventh Annual Research Symposium – Acquisition Research : Creating Synergy for Informed Change*, pp. 385–398, 11–13 May 2010.

[5] V. Berzins, P. Lim and M. B. Kahia, Test Reduction in Open Architecture via Dependency Analysis, *Proc. of Eighth Annual Acquisition Research Symposium*, pp. 333–344, 11–12 May 2011.

[6] LCS Innovation Workshop, Naval Postgraduate School, Monterey CA, 25–26 March 2014.

[7] Y. Qiao and Luqi, Admission Control for Dynamic Software Reconfiguration in Systems of Embedded Systems, *Proceedings of the 2004 International Conference on Embedded Systems and Applications*, pp. 136–142, 21–24 June 2004.

[8] V. Berzins, Certifying Tools for Test Reduction in Open Architecture, in *Proc. of Ninth Annual Acquisition Research Symposium*, Monterey, CA, 2012.

[9] DoD, MIL STD 882D—Standard Practice for System Safety, 10 Feb 2000 [Online]. Retrieved from http://www.everyspec.com

[10] M. Weiser, Program Slicing, *IEEE Transactions on Software Engineering*, , Vols. SE-10, no. 4, pp. 352–357, 1984.

[11] P. Lim and M. Ben Kahia, Suitability of Commercial Slicing Tools for Safe Reduction of the Testing Effort, in *Naval Postgraduate School, MS Thesis*, June 2011.

[12] R. Pudadera, Risk Management Approach to Software Testing through Software Slicing, in *MS Thesis, NPS*, March 2013.

[13] A. Zghidi, Evaluation of Existing Slicing Tools and their Usefulness to Safely Reduce Regression Testing, in *MS Thesis, NPS*, March 2013.

[14] DoD. System Safety, 2012. [Online]. Retrieved from http://www.system-safety.org/Documents/MIL-STD-882E.pdf

[15] Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers International, 2007.

[16] Federal Aviation Administration, Equipment, Systems, and Installations, December 2007.

[17] Parrot, AR.Drone Developer Guide, SDK2.0, Southfield MI, 2012.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CA 93943

www.acquisitionresearch.net