Supplemental Proceedings of the 13th International Symposium on Software Reliability Engineering, IEEE Computer Society, 2002, pp. 7-11.

Everyday Dependability for Everyday Needs

Mary Shaw
Institute for Software Research, International
School of Computer Science
Carnegie Mellon University
+1-412-268-2589
http://www.cs.cmu.edu/~shaw/
mary.shaw@cs.cmu.edu

ABSTRACT

Everyday software must be sufficiently dependable for the needs of everyday people. Everyday people can usually intervene when software misbehaves, and problems with their software are usually irritating but not catastrophic. Everyday software must thus provide cost-effective service with reasonable amounts of human attention. Dependability for these everyday needs arises from matching dependability levels to actual needs, achieving reasonably low failure rates at reasonable cost, providing understandable mechanisms to recognize and deal with failure, and enabling creation of individually-tailored systems and configurations from available resources. This leads to different challenges from mission-critical systems that operate autonomously and risk catastrophic failure.

Much everyday software depends on inexpensive or free information resources available dynamically over the Internet or through retail channels. Much of this software runs on mobile devices with limited power. Increasingly, it is composed by its users rather than by professionals, and the resulting software uses information resources in ways that the resources' creators could not anticipate. In the near future user-managed configurations will have to interact acceptably with configurations managed by other users. Software development in this setting requires methods that tolerate incomplete knowledge, pursue value rather than simply capability, and base reasoning on aggregate rather than fully-detailed information. We will identify research challenges that arise from the need for everyday dependability and give examples of current Carnegie Mellon research that addresses these challenges.

Keywords

Dependability, everyday dependability, user-centered requirements, sufficient correctness, anomaly detection, software homeostasis, software utility theory, software insurance, value-based software engineering.

1. THE COMPUTING GAME HAS CHANGED

The past five years or so have brought a sea change in the character and use of software systems. The Internet has become the delivery mechanism of choice for a wide variety of information resources, and personal information appliances are emerging as significant information platforms. These changes strain the traditional models for software development, configuration, and quality control.

The Internet provides a wealth of information resources: not just software components, but information sources, data feeds, computing services, and communication mechanisms. These are

- *Autonomous:* they are independently created and managed, and they may change without notice
- *Heterogeneous:* they are packaged differently and have different conditions of use
- *Non-procedural:* they are often independent systems with different affordances from software components, and their incidental effects may be useful.

The usual closed-shop software development methods assume that the project manager controls the components being used and that the components are reasonably well specified. Neither assumption holds for Internet resources. Nevertheless, everyday people will inevitably use these resources for ordinary, everyday purposes, and they will combine multiple resources for purposes of their own. The dependability challenge is finding ways to form coalitions -- "systems" would be too strong a term -- of these resources that work well enough for everyday purposes.

Mobile personal computing devices pose another set of challenges for software developers, especially as these devices become internet-capable. These devices have limited capability, their access to external resources changes as they move about, and they must regularly reconfigure their application loads in response.

In both these cases, different users have different tolerances for system error and failure and different interests in the affordances of resources. Further, their tolerances and interests differ from time to time. The criteria for acceptable behavior should reflect these differences. As a consequence, we can no longer associate requirements and specifications solely with components; they must reflect the dynamic needs of individual users.

So, while traditional software systems are localized, independent, and centrally administered, the resource coalitions in these new settings is distributed, interdependent, and user-managed. This raises new issues for dependability. Section 2 describes these issues, and Section 3 illustrates the kinds of research that can address them.

2. PRACTICAL SYSTEMS ARE MORE THAN MERE PROGRAMS

Internet-based coalitions and mobile personal appliances present both familiar and novel dependability challenges. The familiar challenges arise from the differences between software systems and simple programs, as suggested by Table 1.

Table 1. Programs vs systems

Programs: *Complete Knowledge*	*Systems:* *Approximate Knowledge*
Goal: correctness	Goal: adequacy, fitness
Failure prevention	Problem remediation
Good component specs	Components poorly understood
Monolithic design	Cohesion/coupling issues
Open loop operation	Closed loop operation
Requirements tied to components	Requirements sensitive to needs of users

In addition to these familiar differences, the structures of internet-based coalitions and mobile systems are dynamic: both the individual components and the selection of components must change over time. In addition, the criteria for satisfactory performance must be user-centered rather than component-centered.

The traditional model of correctness no longer serves us adequately. In this model the gold standard is functional correctness, or correspondence between computational results and specifications. For systems, this standard extends to include extra-functional properties such as performance, usability, and reliability .

In practice, however, most software in everyday use has bugs, even serious bugs, yet we manage to get useful work done. In practice, it isn't practical to obtain complete specifications:

- There are too many properties that people can depend on, especially when system configuration is user-managed.
- The information that we do have is of variable quality.
- It's too expensive to collect specification information for the properties that are most commonly at issue.
- The specifications should reflect individual users' requirements, especially for extra-functional properties.

As a result, we need to be able to reason from our current partial knowledge -- the *credentials* at hand -- and to update our conclusions as new knowledge becomes available [11]. I conclude that, at least for software that supports everyday activities, it would be more productive to pursue the goal of "good enough for intended use" rather than "correct".

Determining whether software is *sufficiently correct* requires understanding both the level of dependability required for a particular application and the level of dependability provided by the software. Further, the level of dependability is multidimensional: for example, some applications depend on low latency but can tolerate low precision; in other applications precision is critical but latency is not. It follows that software that is acceptable in one situation may be deficient in another.

This view of dependability is rooted in the value proposition of engineering. Engineering seeks timely, cost-effective solutions to practical problems, preferentially solutions based on results that are well grounded in mathematics and science. This entails reconciling conflicting constraints and making design decisions with limited time, knowledge, and resources. This sets the objective of software engineering as creation of overall value, not simply the creation of functional and extra-functional capability.

A growing community of software engineering researchers is explicitly considering ways to infuse value considerations into technical decision-making in software product and process design [4]. This *value-based* approach to software engineering explicitly includes cost-benefit tradeoffs in design. To this end, it identifies useful techniques from economics, social science, and other disciplines and adapts them for use in software design.

3. THE DEPENDABILITY GAME SHOULD ALSO CHANGE

To illustrate the ways that value-based software engineering research addresses the dependability challenges posed by distributed interdependent collections of user-managed resource coalitions, I review some current research at Carnegie Mellon.

Generally speaking, there are two strategies for dealing with undesirable software system behavior: prevention and repair. Prevention involves a priori validation of correctness (or at least sufficient correctness). Traditionally, this has entailed careful development and analysis of the software artifact itself. Recognizing that the criterion for acceptable behavior must reflect the needs of the user implies that this validation must be user-centered. The repair alternative arises from recognizing that it is sometimes more cost-effective to deal with problems when they occur than it is to absolutely prevent them. Traditionally, software repair has concentrated on technical support for fault tolerance. Another option, economic compensation, has seen little attention in the software domain.

Naturally, no one of these alternatives stands alone -- an appropriate strategy should strike a balance between preventing as many problems as is cost-effective, then repairing any others that occur.

3.1 User-centered design and validation

User-centered design must consider the needs and preferences of users. This blurs the line between correctness and quality, because different users have different needs, their needs change over time, and they may be inarticulate about the differences between preferences that reflect dissatisfaction and preferences that reflect actual failure.

Butler has developed a technique for selecting an appropriate suite of security technologies for a particular computer installation [1][2][3]. Different installations must protect different resources; they have different budgets and different concerns about security threats. Selection of security technology for an installation should begin with a quantitative analysis of that installation's risks, but the staff is usually not able to quantify either the frequency of possible attacks or the consequences of a successful attack. Butler adapted the techniques of multi-attribute decision theory to create a risk analysis technique that elicits subjective comparisons that the installation staff is able to make and convert these comparisons to quantitative figures of merit that can be used for

subsequent analysis. Given this analysis, Butler considers the countermeasures available in the marketplace. Adapting the military doctrine of defense in depth, she considers the contributions of potential countermeasures to prevention, detection, and remediation of attacks. Combining threat analysis with countermeasure information, she identifies candidate technologies, performs sensitivity analysis, and iterates the analysis with the security staff of the installation. The process leads both to recommendations for security technology acquisition and to deeper understanding of security issues on the part of the installation security staff.

Poladian is developing techniques for dynamically configuring mobile computing appliances [6]. These appliances have limited resources: processor power, communication bandwidth, battery life, storage capacity, etc. In addition to the hardware limitations, user attention is a scarce, and valuable, resource. Some resources, such as communication bandwidth and externally-available information resources, change as the user moves from place to place. Further, users require different capabilities at different times; this may require changes ranging from adapting the quality of service of a communication link to reconfiguration the applications that are executing. Poladian is modeling the applications' use of resources with linear and integer programming models and casting their preferences as utility functions. This allows him to adapt linear and integer programming models to find preferred configurations. He is particularly interested in the problem of dynamic reconfiguration as needs change. Extending the model to consider future needs in the current reconfiguration decision may incorporate elements of real option theory.

These examples suggest a further research opportunity in the area of systematically identifying and reconciling conflicting priorities and preferences from many sources, including multiple users.

3.2 Value-based fault tolerance

Traditional software fault tolerance is reactive: it makes explicit distinctions between normal states and broken or degraded states, then creates mechanisms that add transitions from bad states back to good states.

These feedback models are of good service when the criteria for distinguishing good states from bad states are available. Unfortunately, software component specifications are often incomplete, and the specifications for internet-available information resources are usually much sketchier. With the added complication that individual users may depend on various affordances of a resource that the proprietor of that resource does not recognize, the specifications supplied with a resource are often unlikely to be adequate to support state-based fault tolerance.

Raz is applying machine learning and statistical techniques for detecting anomalies in data feed resources. She does this by inferring predicates describing "normal" behavior of a data feed from the past performance of that data feed, with guidance from the user [7][8][9][10]. Unlike software components, which have APIs, data feeds provide sequences of data records with fields that may (or may not) be correlated to each other and exhibit patterns of behavior over time. Raz has a toolkit of inference techniques that can propose predicates describing historical behavior of a data feed. For a given user and data feed, Raz' process has a setup step that elicits the user's expectations about the data feed by presenting candidate predicates for the user to accept, reject, or modify. This leads to a set of inferred predicates that can serve as proxies for specifications in the operational stage, when the values delivered by the data feed are checked for anomalous behavior. Like Butler, Raz has found that a benefit of the process is improved understanding on the part of the user.

An alternative to the traditional stateful view of fault tolerance is that the explicit distinctions between states are often not crisp enough for reactive repair, and that defining these states requires specification effort that might better be invested elsewhere [12][13]. In this view, there are gradual transitions between desirable and undesirable conditions, and the important thing to specify is the gradient between them. This captures two aspects of practical systems that are missing from state-based fault tolerance. First, there are gradients of desirability within the zones of normal and broken behavior. Second, the same mechanisms that move the system from "broken" back to "normal" can also serve to improve performance within the normal region (or to make things less broken in the broken region). This view handles adaptation to changing needs and changing resources in the same way as it handles repair. This observation argues for research in fault tolerance techniques based on homeostasis rather than state-based recognition. This research would generalize from specific examples such as Internet packet routing.

3.3 Economic compensation

For everyday software, compensation for losses may be a reasonable alternative to technical remediation. This is especially true for time-dependent results and when the consequences of failure are large enough to matter but not so large as to be catastrophic. This is commonly achieved in other areas through product insurance and warranties.

Li is investigating ways to provide the actuarial data required for insurance and warranty models. He has examined the synergy between insurance and probabilistic certification, which offers the prospect that the same data that supports probabilistic certification (in this case, of the timeliness of results) can provide the actuarial basis for insurance [5]. He is also investigating ways to predict future failure rates from historical data, and doing so early enough in the product history to set premiums. To this end, he is developing a model for predicting failure rates in software releases on the basis of development information and defect reports for previous releases. Actually establishing a system of software component insurance requires additional work in defining insurable risks, identifying failures, assessing damages, and the integration of insurance models in software production and use. However, the ability to predict failure rates is one of the core technical problems. Note that Li's results could potentially be combined with Raz' anomaly detection techniques to automatically detect failures and file claims.

4. MAJOR POINTS

I argued that the internet and mobile computing have fundamentally changed the character of software systems and that software development methods, especially dependability methods, must change in response. I identified "sufficient correctness" as a standard of dependability that reflects both individual user needs and the pragmatic limitations on specifications. I offered several current research results in value-based software engineering as examples of promising approaches to the new dependability problems.

5. ACKNOWLEDGMENTS

The Economics-Driven Software Engineering Research community has provided an invaluable sounding board for the research reported here. This research is supported by the National Science Foundation under Grant ITR-0086003, by the Sloan Software Industry Center at Carnegie Mellon, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

6. REFERENCES

[1] Shawn A. Butler. Security Attribute Evaluation Method. A Cost-Benefit Approach. *Proc ICSE 2002 - Int'l Conf on Software Engineering*, 2002

[2] Shawn A. Butler and Paul Fishbeck. Multi-Attribute Risk Assessment. *Symposium on Requirements Engineering for Information Security*, 2002.

[3] Shawn A. Butler and Mary Shaw. Incorporating Nontechnical Attributes in Multi-Attribute Analysis for Security. *Proc EDSER-4: Workshop on Economics-Driven Software Engineering Research*, 2002.

[4] Proceedings of the Workshops on Economics-Driven Software Engineering Research, EDSER-1, -2, -3, and -4. Workshops held in conjunction with the 21st through 24th ICSE's: International Conference on Software Engineering, 1999 to 2002.

[5] Paul Luo Li, Mary Shaw, Kevin Stolarick, and Kurt Wallnau. The Potential for Synergy Between Certification and Insurance. *Proc Int'l Workshop on Reuse Economics*, 2002.

[6] Vahe Poladian, David Garlan, and Mary Shaw. Software Selection and Configuration in Mobile Environments: A Utility-Based Approach. *Proc EDSER-4 - Workshop on Economics-Driven Software Engineering Research, 2002*.

[7] Orna Raz, Philip Koopman, and Mary Shaw. Semantic Anomaly Detection in Online Data Sources. *Proc ICSE 2002 - Int'l Conf on Software Engineering*, 2002.

[8] Orna Raz, Philip Koopman, and Mary Shaw. Benchmarking Semantic Availability of Dynamic Data Feeds. *Workshop on Dependability Benchmarking*, 2002.

[9] Orna Raz, Philip Koopman, and Mary Shaw. Enabling Automatic Adaptation in Systems with Under-Specified Elements. *Proc WOSS'02 - Workshop on Self-Healing Systems,* 2002.

[10] Orna Raz and Mary Shaw. An Approach to Preserving Sufficient Correctness in Open Resource Coalitions. *Proc IWSSD-10 - Int'l Workshop on Software Specification and Design*, 2000.

[11] Mary Shaw. Truth vs Knowledge: The Difference Between What a Component Does and What We **Know** It Does. *Proc IWSSD-8 - Int'l Workshop on Software Specification and Design*, 1996.

[12] Mary Shaw. Sufficient Correctness and Homeostasis in Open Resource Coalitions. *Proc. ISAW-4 - Int'l Software Architecture Workshop*, June 2000.

[13] Mary Shaw. "Self-Healing:" Softening Precision to Avoid Brittleness. *Proc WOSS'02 - Workshop on Self-Healing Systems,* 2002.