Proc of ACM SIGSOFT WOSS "02, South Carolina, 2002, pp. 111-114

"Self-Healing": Softening Precision to Avoid Brittleness

Position paper for WOSS '02: Workshop on Self-Healing Systems

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

Modern practical computing systems are much more complex than the simple programs on which we developed our models of dependability. These dependability models depend on precise specifications, but it is often impractical to obtain precise specifications of practical software-intensive systems. Furthermore, the criteria for acceptable behavior vary from time to time and from one user to another. When development methods are based on the classic models that assume precise specifications, the resulting systems are often brittle -- they are vulnerable to unexpected conditions and hard to tune to changing expectations. Practical systems would be better served by development models that recognize the variability and unpredictability of the environment in which the systems are used. Such development methods should pursue not the absolute criterion of correctness, but rather the goal of fitness for the intended task, or *sufficient correctness*. They should accommodate environmental unpredictability not only by reactive mechanisms, but also by design that produces resilience to environmental change, or *homeostasis*. In many cases, this resilience may be achievable by relaxing tolerances in the specifications, thereby enlarging the envelope of acceptable operation.

Keywords

Self-healing systems, sufficient correctness, software homeostasis, software utility theory.

1. PROGRAMS vs. PRACTICAL SYSTEMS

Much of software development methodology is rooted in classical computer science, which addresses the problem of developing correct, largely deterministic programs for well-specified problems. The setting assumed for this sort of software development includes well-specified components, causal domains [2] with predictable causal relationships among causal phenomena, and independence from external effects on the computation. These programs often compute a result and stop, and they usually take the objective of preventing failure.

Real systems, however, are much more complex. They are utilitarian, focusing on fitness for purpose even when the problem is not completely understood and the requirements change unpredictably over time. They are built with under-specified components for use in domains that are only biddable [2], lacking predictable causality and subject to uncontrolled external influences on the computation or the system in which it is embedded. These systems are often embedded in physical systems, run indefinitely, and must remediate problems instead of preventing them. Table 1 summarizes these distinctions.

Maintaining the health of practical systems is correspondingly more complex. First, preservation of health depends on knowing what health is. Since the designer's understanding of both the properties of the system and the users' requirements will be incomplete and dynamic, "health" itself will be imprecisely understood. Second, most of the reactive repair mechanisms depend on the ability to detect unacceptable system behavior. Since this will be defined imprecisely, the repair mechanisms should, in part, operate independently of such detection.

Table 1. Characteristics of practical systems that make them more complex than simple programs

Programs: *Complete Knowledge*	*Systems:* *Approximate Knowledge*
Goal of correctness or perfection	Goal of adequacy or fitness for purpose
Stable knowable configuration	Dynamic components and configuration
Components well specified	Components poorly understood
Not subject to external upset (causal domains, can run open-loop)	Vulnerable to changes in environment (biddable domains, should run closed-loop)
Often execute and stop	Run indefinitely
Prevent failures	Remediate problems

When development methods based on assumptions about simple programs are used on systems for which precise specifications and control of the operating environment are unachievable, the resulting systems are often brittle. This can be addressed to some extent by adding layers of error detection and repair, provided the "errors" can be precisely-enough defined. However, designing tolerance for the uncertainties into the systems should yield more manageable results. This position paper argues that we need to consider new approaches to healing and adaptation that deal

WOSS '02, Nov 18-19, 2002, Charleston, SC, USA.

directly with the uncertainties in requirements and in knowledge about the underlying system components and operating conditions. It suggests promising opportunities for research in this area.

2. SUFFICIENT CORRECTNESS

Maintaining system health requires knowing what "health" is and recognizing when the system needs to be healed. The first problem is establishing the criterion for health, which depends on the way the user is depending on the system. This criterion varies from one user to another and from one situation to another. The second problem is recognizing the difference between "healthy" and "unhealthy" conditions. To avoid the brittleness that results from depending on precise requirements and specifications, we consider less precise alternatives.

2.1 What is "health"? Eliciting requirements

Classical models require a full static requirement, but this is impractical for most systems because users are often unable to articulate their needs, because the distinction between "healthy" and "broken" is often indistinct, and because even soft explications of the requirement change over time. An alternative to attempting to elicit precise requirements is to elicit an envelope of acceptable behavior that reflects the concerns and priorities of the user. Butler has shown how to apply multi-attribute decision theory to this problem for security risk analysis [1].

Satisfying the requirement, however precisely it may be stated, depends on many properties of a system. In addition to correctness, these properties include performance, dependability, fidelity, and others. Our knowledge of these properties is inevitably incomplete, because the set of properties is open-ended and the cost of obtaining specifications for all possible properties is too high. 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 [5].

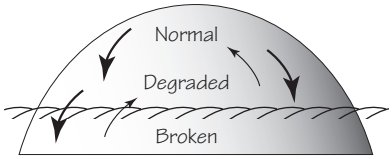
Further, the user's preference among these properties is often a complex function of the quality with respect to each property. Poladian is applying utility theory to striking reasonable balances among properties, within resource constraints, for mobile systems [3]. Mobile systems are resource-constrained, and different users have different preferences (utilities) for different capabilities at different times. The applications that can provide these applications differ in their relative consumption of the available resources, but it appears reasonable to model the resource use as linear in some measure of problem size. This leads to a linear programming formulation of the constraints and objective.

2.2 When does illness set in? Spotting trouble

Traditional models of self-healing or self-repair make explicit distinctions between normal states and broken or degraded states. However, given the difficulties in establishing a precise specification of "health" -- and indeed the gradual transition between perceived health and perceived unhealth -- we need a new approach to recognizing when a system is in need of healing.

An alternative view [4] is that explicit distinctions between normal and non-normal states are artificial and serve more to support a state-based model than to capture actual system behavior. In this view, there are gradual transitions between desirable and undesirable conditions, and the important thing to specify is the gradient between them. Figure 1 suggests this situation: there are regions in which the behavior is clearly acceptable or unacceptable, separated by a fuzzy zone of marginal behavior.

Figure 1. Degradation and failure in real systems



This view captures two aspects of practical systems that are missing from state-based models. 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). In addition, it saves the specification effort required to precisely delineate the difference between the states.

This view handles adaptation to changing needs and changing resources in the same way as it handles healing when too much capacity is lost.

2.3 Maintaining health -- being good enough

An appropriate view is that for many practical systems, health corresponds to reasonable assurance that the system will work well enough for its intended purpose, or

> **Sufficient correctness:** *The degree to which a system must be dependable in order to serve the purpose its user intends, and to do so well enough to satisfy the current needs and expectations of those users.*

That is, self-healing systems should pursue the objective of being good enough for the task at hand. Since systems may be used for different purposes, it is not reasonable to expect a single global standard of health. This definition recognizes the common case of imprecise requirements and the differences among utility functions for different uses of the system.

3. HOMEOSTASIS

Most approaches to self-healing require explicit specification of the conditions that trigger healing responses. They uses these specifications to establish the "setpoints" for feedback mechanisms that compare system behavior to the setpoints, then adjust system parameters to drive the system toward the setpoints. Making these adjustments requires the ability to predict the effect of the adjustment on the system.

In practice, however, the distinction between health and unhealth is often gradual. An alternative is to design for ongoing background activities that tend to drive the system up a gradient of good health, independent of whether or not the system is healthy.

3.1 Homeostasis vs. feedback

Homeostasis is the mechanism through which a system acts to maintain a stable internal environment despite external variations. Bernard first stated the concept in the 19th century, and Cannon coined the term itself in 1932. Homeostatic systems react to change, not to explicit transition into undesired states. So the pupil of your eye expands and contracts in response to light

through variations in normal light levels. At an ecological level, the predator/prey relation between, say, foxes and rabbits operates homeostatically to control the populations of those species.

Simon, discussing system design, distinguishes the internal environment (the one we're designing) from the external environment (the one our system exists in). He says [7], p.149

> Few of the adaptive systems that have been forged by evolution or shaped by man depend on prediction as their main means for coping with the future. Two complementary mechanisms for dealing with changes in the external environment are often far more effective than prediction: homeostatic mechanisms that make the system relatively insensitive to the environment and retrospective feedback adjustment to the environment's variation.
>
> Thus a stock of inventories permits a factory to operate without concern for very short-run fluctuations in product orders. Energy storage in the tissues of a predator enables it to cope with uncertainties in the availability of prey. A modest excess of capacity in electric generating plants avoids the need for precise estimation of peak loads. Homeostatic mechanisms are especially useful for handling short-range fluctuations in the environment, hence for making short-range prediction unnecessary.
>
> Feedback mechanisms, on the other hand, by continually responding to discrepancies between a system's actual and desired states, adapt it to long-range fluctuations in the environment without forecasting. In whatever directions the environment changes, the feedback adjustment tracks it, with of course some delay.

Here Simon emphasizes homeostatic mechanisms that allow fluctuations without explicit response. The broader definition of homeostasis includes mechanisms that react to change without reference to a distinction between "good" and "bad" states.

3.2 Homeostasis for software

Homeostasis for software systems arises from mechanisms that continually improve system health (revised from [6]).

> *Homeostasis is the propensity of a system to automatically resist change from its normal, or desired, or equilibrium state when the external environment exerts forces to drive it from that state.* ***Software homeostasis*** *as a software system property refers to the capacity for the system to maintain its normal operating state, or the best available approximation to that state, as a result of its normal operation. This operation should both maintain good normal operation and implicitly repair abnormalities, or deviations from expected behavior.*

Common examples of homeostasis in software systems include background garbage collection (which heals the free space pool whether or not space is low) and Internet packet routing (which, by virtue of its normal policy of finding good routing by dynamically selecting the most promising route for each packet, also routes around network problems).

These are, respectively, examples of ongoing background maintenance and dynamic selection of a required resource. As Simon notes, homeostasis can also result from maintaining excess capacity: an elementary result from queuing theory states that for a common class of systems, as the average time between arrivals in a queue for service approaches the average service time, the queue length tends to infinity. System slack and safety factors are similar examples.

This approach to self-healing has two advantages over reactive and feedback models. First, the mechanisms operate over a wide range of performance, improving performance independent of health status. Second, it does not require the effort of developing precise specifications of the distinctions between internal states, freeing that effort for other aspects of design.

4. MAJOR POINTS

A given system may be used in different ways with different requirements for correctness or quality. As a result, the criteria for system health depend on the task at hand.

Unfortunately, most users are inarticulate about their criteria for correctness, performance, dependability, and other system qualities. Decision theory offers guidance for improving the quality of requirements, and utility theory offers a means of expressing multiple and nonlinear preferences.

The need for self-healing arises because the system may be subject to unpredictable external forces or because the system is too complex to predict its internal behaviors precisely. It is often possible to design resilience to this uncertainty into normal operation.

Heavy dependence on precise specifications, including optimizations that wring slack out of the system, are likely to make a system brittle and subject to failure when unexpected conditions arise.

Homeostatic mechanisms offer a complementary approach to reactive and feedback mechanisms for self-healing, especially for dealing with short-term fluctuations in conditions.

5. ACKNOWLEDGMENTS

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*.

[2] Michael Jackson. *Problem Frames*. Addison-Wesley 2001.

[3] 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*.

[4] 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.

[5] 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.

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

[7] Herbert Simon. *Sciences of the Artificial*. MIT Press, 1996.