ALVA COUCH

configuration management phenomenology

Alva Couch is an associate professor of computer science at Tufts University, where he and his students study the theory and practice of network and system administration. He served as program chair of LISA '02 and was a recipient of the 2003 SAGE Professional Service Award for contributions to the theory of system administration. He currently serves as Secretary of the USENIX Board of Directors.

couch@cs.tufts.edu

A QUANDARY IN MAPPING BEHAVIOR TO configuration is traced, in part, to a philosophical quandary rooted in the relationship between system administrator and user.

"Talk to the bomb. Teach the bomb phenomenology."

—The captain, in *Dark Star: The Spaced-Out Odyssey*

At the climax of the cult science fiction parody *Dark Star* [1], a "smart bomb" has decided to explode while still in the spaceship bay, rather than exploding on the planet that it is supposed to destroy. The spaceship crew attempt to "solve" this problem by engaging the bomb in a deep philosophical discussion of the meaning of life and exploding; they try to convince the bomb that it need not explode, because the importance of whether it explodes or not is subjective and not particularly significant in the larger picture of things.

This silly discussion somehow reminds me of the state of the art in configuration management. Tools act on the configuration as if it were the definitive representation of behavior and assume that this is enough for tools to do. Meanwhile, the behavior of a configuration management solution is monitored via mechanisms that—again—quietly assume that configuration *defines* behavior. But it might not, for many reasons. In autonomic computing parlance, the human system administrator is left to "close the loop" between configuration and behavior, and, when things go wrong, must rely upon intuition and experience to "close" this loop manually.

The situation, similar to the situation in *Dark Star*, is that the tools allow the environment to "explode," humans must intervene, and the apparent solution, as in *Dark Star*, is to "teach the tools phenomenology" by giving tools perceptive capabilities by which they can understand the effects of their actions. What does this mean, and is it even reasonable? In this article, we explore this question from several angles.

Beyond Semantics

This article might be considered the second in a series. In the first article [2], we discussed the semantic wall between high-level and low-level configuration specifications and how difficult it is to map between high and low levels of abstraction in a configuration. We commented on the difference between "specifying configuration" and "specifying behavior" as a problem of semantics.

Now, two years later, another problem looms on the horizon. Even if we manage to successfully bridge the gap between levels of configuration abstraction, the problem of bridging desired and observed behavior remains. This is not just a problem of semantics, but a deeper problem with the assumptions we make and the way we approach both configuration management and the profession. While the former problem arises from difficulties of meaning, this problem arises from the philosophy that we adopt in satisfying user needs.

Phenomenology

In a naive sense, "phenomenology" refers to the practice of relying upon one's senses to define the nature of the physical world. In like manner, I refer to phenomenology in system administration as the practice of trusting what one can observe the system actually doing, instead of trusting any abstract idea one might have of what it is supposed to do. Thus I propose that "a machine's identity is what it does," by contrast with the traditional configuration management view that "a machine's identity is how it is configured." For each configured machine (to paraphrase Sartre), I assert that "to do is to be," i.e., machines' behaviors define their natures. By contrast, a fundamental tenet of configuration management (attributed to Socrates) is that "to be is to do," i.e., machines' natures define their behaviors.

Although it might seem that I am splitting meaningless philosophical hairs, there is a world of difference between these definitions that strikes at the core of the assumptions underlying configuration management as a practice. We often comfortably and tacitly assume that the way a machine is configured defines its behavior. I beg to differ for a multitude of reasons. The reason that this assumption is false is more than a simple problem of semantics. Behaviors arise from sources other than the configuration.

Using Phenomenology

Phenomenology is not a new idea for system administrators; we use it every day. In tuning a configuration or troubleshooting a problem, we engage in controlled (or perhaps not-so-controlled) experimentation. We are intimately familiar with many cases in which what something does correlates poorly with what we think it is and—implicitly—we quietly modify our idea of what it is, accordingly.

My evidence is, however, that we do not go far enough in believing our senses. Our behavior is based upon hidden assumptions—deeply embedded in practice—that influence and sometimes cloud our thinking. One way to bring those assumptions out into the open is to consider how we philosophically approach the problem of system administration.

Verification and Validation

There is a subtle difference between what current configuration management tools do and what we tacitly assume that they do, which is similar to the difference between "verifying" and "validating" a software product in software engineering. According to software engineer Barry Boehm, the process of "verification" answers the question, "Are we building the product right?" This is the way most configuration management tools work. A configuration is "verified" if it accords with the system documentation. "Validation," by contrast, answers the question, "Are we building the right product?" A system is validated if it is doing what users need it to do (and—implicitly—not doing things they do *not* want it to do) [3].

In human terms, verification involves making sure that we have obeyed the documentation for a product in trying to manage it, while validation involves ensuring that the documentation is itself correct and definitive about the relationships between configuration and behavior. Current practice engages in the former and assumes that verification implies validation (so that explicit validation is optional rather than required). And this is almost always a bad assumption to make.

Consider, for example, that a non-functional email server can be broken in two basic ways. First, the configuration can remain unverified, e.g., the configuration tool fails to modify it properly. It is more common, however, for the configuration to be verified but not validated, e.g., the configuration looks as it should, but the system still fails to forward email. The latter is a validation problem.

In going around the table at LISA, I found that almost everyone has some story of getting burned as a result of incorrectly assuming that the documentation is correct. The simplest example is that of a manual page that describes the wrong syntax for a file, but there are much more subtle variants. As software is revised, the manual pages need not keep up with it, so that one is often reading older descriptions of newer software.

Closed-World Assumptions

The assumption that verification implies validation is just one example of a "closed-world assumption" that arises in system administration practice. Verification is necessary but not sufficient for validation. Verification is only sufficient when "what you ask a system to do" is always "what it does." This is an implicit closed-world assumption that all influences upon the managed system are known and accounted for. In other words, the kind of thinking that this represents might be paraphrased as "to be is to do."

There are many cases in which this implicit assumption fails to hold: when the managed software has a bug that affects behavior, for example, or when there are hidden unmanaged influences, such as a forgotten configuration file that can adversely affect behavior. In the worst case, a security breach can change all the rules and even replace the managed application with another unknown and hostile one.

Closed-world assumptions pervade our practice. We often implicitly assume that configuration completely determines behavior, and that a specific configuration tool completely controls configuration and thus behavior. I say "implicitly" because there is no conscious action on our part to assume anything, but the assumption quietly lurks in how we use our data!

Consider, for example, how we currently document a site's function. Usually, some description of the configuration suffices: either a description of how each machine is configured or some network-wide, tool-readable description. This seems innocent enough, until we consider that it is often the *only* documentation of site function. At a deeper philosophical level, a configuration description cannot be more than a statement of *intent* rather than fact. Anything we do outside its closed-world assumption is (implicitly) not documented.

Open-World Assumptions

Phenomenology, by contrast, implicitly adopts an open-world assumption that more or less any behavior can arise as a result of configuring a system. A system is what it does. The configuration might result in appropriate behavior, but it might not. Verification does not imply validation. In other words, we might think of an open-world assumption as equivalent to the philosophical stance "to do is to be."

One's philosophical stance can have a profound impact upon one's everyday practice. If one really considers validation as separate from verification, then there is no way to "prove" correct system function. As in software testing, one can never fully test a system, and the only solid evidence one can gather is that something is *not* working. But this philosophical stance also clarifies some of our thinking about behavior. By throwing away a tacit and common assumption that has been proven false countless times, we are freed to reason more clearly about configuration, behavior, and contingencies.

I consider it almost a tautology that the job of a system administrator is to "close an open world," i.e., to provide some concept of predictability in an otherwise unpredictable environment [4]. One starts with an "open world" (e.g., the Internet) and makes some adjustments to make that world "usable." Along the way, one forms "closures," islands of predictability in an otherwise unpredictable universe, where what you think you are telling something to do is what it actually does, i.e., verification implies validation [5]!

The Value of Philosophy

So far, this discussion probably seems abstract and impractical. What, you might ask, is the value of a philosophical stance? Isn't system administration what we do, and not how we think about it? I claim that simply refusing to "believe" that verification implies validation has profound implications for practice.

Particularly, if we refuse to blindly believe that verification implies validation, there is always a validation step after configuration management. That step involves observing behavior, and effective testing (manual or automatic) becomes a central part of system administration and our tools. Tools learn to observe the world as well as to configure it.

But less tangible benefits include the ability to ask new questions that our prior beliefs had sidelined. We must eventually ask, "What is validation?" and, more importantly, "What behavior is actually desired?"

Monitoring Is Not Validation

One might think that log monitoring is a form of validation; after all, monitoring does measure behavior rather than configuration. But monitoring records *symptoms* of behavior, not the behavior itself, and it is possible for a system with the proper symptoms to be behaving improperly.

Consider the common problem of a log message saying that an undelivered email was delivered. This can happen in many ways: for example, the file system on which the message is to be stored can fail *after* delivery. Symptoms can only be definitively related to causes if there is again an implicit "closed-world assumption" that the monitoring data is complete enough to represent what actually happened. In the above case, that assumption is equivalent to the assumption that "the disk does not fail," which is clearly ridiculous. Expected log entries are again necessary but not sufficient for proper operation; there are many cases in which the log is correct but behavior is wrong.

Monitoring *is* validation if an appropriate closed-world assumption holds. Thus, monitoring is sufficient if we have already verified (by some other mechanism) that a closed-world assumption is reasonable. But monitoring, by itself, cannot substantiate a closed-world assumption.

Real validation involves more explicit testing than most of us do. Are email messages really being delivered? Are services responding properly? This includes checking on the actual function of services, and not solely relying upon logs of past behavior.

What Is Behavior?

To achieve validation we must first understand what behaviors are desirable. Describing behavior might seem a daunting task, but we are aided by two simple ideas. First, user-level behavior is much easier to describe than the configuration that assures it. Behavior is a much higher-level thing to describe than configuration. A behavioral description can be written to be relatively portable and reusable for many sites, while configuration contains the (often hopelessly non-portable) methods for assuring that behavior. Configuration—because it is "how" and not "what"—contains details that have nothing to do with behavior. Second, most user needs are met by a set of well-known behaviors. Behavioral expectations are largely homogeneous over the whole Internet and thus more reusable from site to site than configuration details, which by contrast are highly heterogeneous.

Note that a so-called "high-level configuration system" as first proposed by Anderson [6] is *not* a description of behavior but, rather, an abstract (and hopefully more portable) definition of *configuration*. A "high-level" configuration language still describes "what a system should be" instead of "what a system should do." Any linkage between these two is again a closed-world assumption.

Facing Social Forces

Given that the system administrator has to use phenomenology on a daily basis, one might ask why implicit closed-world assumptions are so easy for us to accept. I believe the roots of our closed-world assumptions are social rather than scientific.

One social reason that it is "convenient" to sweep "behavior" under the rug is that we remain unaware, on average, of exactly how our systems behave. Users make changes, and thus behavior changes. There is "behavioral drift" (and even "behavioral rot") based upon independent actions of individuals, especially in a desktop environment. But at a deeper level, the system behaviors that users "need" are different from what they might "want." And facing *that* quandary, and the quandary of whether to give users what they want or what they need, remains "the elephant in the room" whenever we discuss behavior.

Our job is "closing open worlds." The typical user wants to be able to do "everything." And we can't close *that* world.

I think this social reason is the real force underlying our confusion between configuration and behavior, and between verification and validation. It is "convenient" and "comfortable" to assume that configuration determines behavior—and, implicitly, that verification implies validation—because otherwise we have some very difficult social questions to answer about what behavior "should" be. We can hide behind what tools do and escape the "should," by adopting a convenient closed-world assumption!

Do-Be-Do-Be-Do!

The old joke (which I first learned from scribblings on the MIT Math Department men's room wall) is that the response to Socrates' "To be is to do" and Sartre's "To do is to be" is Sinatra's "Do-Be-Do-Be-Do"! I think that Sinatra better describes current configuration management practice than Socrates or Sartre does. We make closed-world assumptions in enforcing and monitoring behavior, and open-world assumptions in troubleshooting. I believe that for the practice to evolve, we have to stop conveniently fabricating closed worlds where they cannot exist. But to do this, we must acknowledge and directly deal with the social forces that brought about our current philosophy.

Facing the social forces is uncomfortable, and the fuzzy relationship between user and system administrator can become even fuzzier when we try to document it. Users ask for "everything," implicitly or explicitly, and we find it difficult to say no. It is more comfortable sometimes to live in ignorance of user expectations and hope in return that users live in ignorance of our true limitations!

But I also believe that facing this "elephant"—and coming up with ways to precisely specify and guarantee system behavior—is crucial to the ongoing evolution of the profession. Without that step, system administration appears to undertake the theoretically impossible task of closing *every* world the user's heart desires. Making the task clearer to the user involves casting out our own closed-world assumptions in a first step toward encouraging users to cast out theirs.

Only then can we truly be partners with users and replace attempting the impossible with cooperating on the possible. To do this is to be.

REFERENCES

[1] For some reviews of *Dark Star*, see http://www.flixster.com/movie/dark-star.

[2] Alva L. Couch, "From x=1 to (setf x 1): What Does Configuration Management Mean?," *;login:*, vol. 33, no. 1, February 2008.

[3] Barry W. Boehm, *Software Risk Management* (IEEE Computer Society Press, 1989), p. 205.

[4] Alva L. Couch et al., "Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management," Proc. LISA 2003.

[5] Mark Burgess and Alva Couch, "Modeling Next-Generation Configuration Management Tools," *Proceedings of LISA '06: 20th Large Installation System Administration Conference* (USENIX Association, 2006).

[6] Paul Anderson, "Toward a High-Level Machine Configuration System," *Proceedings of LISA VII: 7th USENIX System Administration Conference* (USENIX Association, 1994).