When to Eat Your Own Dog Food

Warren Harrison’s column “Eating Your Own Dog Food” (May/June 2006) was interesting. I have found dogfooding to be useful when the development process is agile and the dog food is being released in short cycles with small incremental changes.

If the company doesn’t continue to use the product internally after release, then dogfooding is just a form of beta-testing. If the “testers” know that it’s not going to continue to be used internally, then the effort they expend will be minimal. If the only customer is internal, then I think it too would constitute not dogfooding, but a form of beta-testing.

Some very good products have been developed internally and then released externally, and these fall into three broad areas:

- software development companies that build software products to assist their businesses’ nondevelopment facets (such as finance and customer relationship management),
- non-software development companies that build software products to help their businesses’ core facets (for example, an engineering firm that develops an application to assist with engineering), and
- software development companies that build software products to assist in their businesses’ development facets (such as tools for testing, issue tracking, and so on).

The scenario in which a company decides to create a product and then edicts that it be used internally is somewhat unusual and potentially dangerous. The decision to dogfood products should be made by individual users (within reason). Management’s job is really to sell the benefit of dogfooding the product to the internal users.

Dogfooding can’t eliminate or even reduce the need for other forms of testing. In my opinion, it’s not a form of testing but is more useful in determining potential new features and identifying different usage scenarios. This is helpful only if the developers treat internally identified issues with the same seriousness as those identified by paying customers. Dogfooding can also help identify usability issues but only if the developers can sit with the users (which usually isn’t an issue if the developers are themselves doing the dogfooding).

As far as identifying bugs is concerned, this shouldn’t be the only, or primary, bug-identification method, but it’s a good supplement to more formal techniques. Also, in scenarios where the organization doesn’t have a standard operating environment, it can expose the application to a variety of hardware and software configurations, which can isolate bugs that might not otherwise be discovered until release.

William Bartholomew
Developer productivity specialist
william@bartholomew.id.au

Warren Harrison responds:
Good points. Although I agree that dogfooding can augment more traditional testing schemes, I have two concerns:

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.
■ You might think you can bypass those other schemes because you’ve got internal users, which actually makes dogfooding dangerous.
■ You might not address problems you’ve found because these aren’t “paying customers,” which makes the rationale for dogfooding specious.

I agree it’s bad to dogfood in place of thorough testing. But quality doesn’t equal “passing a set of thorough tests.” Classical testing typically involves a set of scenarios, with stimuli applied to the software under test. The software’s responses are then compared to the expected or required results to generate quality metrics. But quality includes such product attributes as functionality and usability, which this sort of testing often leaves uncovered.

Consider a tool to manage software development. Suppose that the engineers in the company that makes the tool refuse to use it because it’s too cumbersome or rigid. I don’t really care what percent of the code was covered by execution testing—if they won’t use it, why should I even think about using it to manage software development in my organization?

Dogfooding isn’t everything, but it can be pretty important.

Collin Park
Senior engineer, Network Appliance
collin@rc5.us

Warren Harrison responds:
Management might say “our software process will be managed using our tool, no ifs, ands, or buts about it” (of course, as an external customer, you have no way to have any insight into this). Given this very plausible situation, why would dogfooding make a software consumer any more confident in the product?

If the company’s engineers won’t use their own product, this might say something about it. It might say that the product is hard to use, but it might also say that the company uses the right tool for the job, and that although their tool might be exactly what you need for your job, it isn’t what they need for theirs.

On the flip side, if the company’s engineers do use the product, it doesn’t really tell me much—who’s to say whether they use it because they like it or because management has mandated that they use it?

**Password management**

In “Passwords and Passion” (July/August 2006), Warren Harrison wrote, “most people aren’t going to commit a huge amount of resources to achieve these goals”—that is, to adhere to a set of rules in the context of password security. In spite of a tendency for using the single-sign-on technique nowadays, the security implications of using the same password on multiple systems are crystal clear: once a password is compromised, the system administrator will take a real hammering, and an attacker will be able to breach multiple systems with ease.

As both a system administrator and security analyst, I would prefer that users keep passwords unique to each system they access. As long as they safeguard their password lists, the increase in password security will often offset the decrease in the passwords’ physical security. Meanwhile, it’s important to ask users to write down strong, complex passwords to make brute-force and dictionary-based attacks fruitless and remind them to maintain their password lists’ physical security.

An egregious security problem is that passwords can be broken by simple guessing processes such as password cracking, in which secret passwords are recovered from stored or transmitted data by repeatedly guessing the password. Therefore, a good password must be hard to guess but easy to remember. Although I agree with Warren’s point that “users are more likely to view complicated password rules and mandatory change schedules as simply more bureaucratic overhead rather than as an important part of sys-
tem security,” there are still so many ways to create a good password. A feasible way might be to combine the first character of each word from a phrase or a sentence and add a digit or special character to the end. Nonetheless, it’s essential to educate users to take responsibility to protect their passwords’ security as well as to help protect the systems they use.

A lot of computers’ default configurations are still on the loose for weak passwords. For example, password policies in Windows Server systems aren’t set by default. The default anonymous user account for FTP is “guest” in many systems. More often than not, these accounts (some with permissions to certain system folders) are left over from installation, and hackers can use them to gain access to or even compromise the whole system. Many such default accounts are easy to locate and collect.

Users should also avoid using their usual passwords on a bulletin-board system where passwords are most likely stored in clear text format. By the same token, it’s not a good idea to send passwords through email that other people might read; a system with good passwords alone isn’t yet a secure system.

I believe that both voluntarily changing passwords from time to time and forcing users to use fresh passwords regularly are imperative parts of the password security policy.

And lastly, never use the account name as the password!

Hong-Lok Li
Information technology manager
University of British Columbia
lihl@ams.ubc.ca

Like many people, I use the same password for multiple accounts. In my defense, I’ll cite the 80/20 rule I’ve seen in other contexts: Twenty percent of the items on just about any list warrant 80 percent of the attention.

I have several online accounts whose security doesn’t matter all that much to me. If somebody hacks my free account at some magazine and reads articles in my name, why should I care? If they want to pretend to be me when buying groceries, why should it matter to me as long as they pay for whatever they buy? So some publication’s circulation department ends up with biased statistics. So the supermarket thinks I have weird shopping habits. So what?

It would be very annoying if someone managed to disrupt delivery of my morning paper. Most of my online accounts are at this level of concern. But that’s still relatively minor and not the kind of thing most serious bad guys would be interested in. They’d be more likely to want to crack the host to get lists of credit card numbers and such. That’s more the provider’s problem.

True, I do have accounts with banks
and pharmacies, where intrusion can have serious consequences. But those are relatively few, and I can afford to give them the majority of my attention with respect to things like password management. They’re also prime candidates for new authentication technologies, while my other accounts can continue to muddle along with old-fashioned passwords.

Perhaps the lesson here is that we should remember the 80/20 rule and not sweat the small stuff.

Tom Digby
bubbles@speakeasy.net

Warren Harrison responds:

Given the number of sites that require authentication these days, that 20 percent is probably high—I suspect only five percent of the sites I want to access are critical enough for serious password management effort.

The problem is that we never really know when one of those 80 percent sites will become a 20 percent site. Surely when opening up an account for e-banking, purchasing stock, or accessing your retirement account, you’ll want the strongest password you can come up with. But what about sites that initially don’t store any information about you but start maintaining more and more sensitive information over time?

More than one e-commerce site I regularly use has gone from not retaining any information about me to keeping my name and address (I suppose to maintain my shopping cart) and requiring a username and password. In general, I’d consider this a low-significance site—after all, you can call information and get my name and address anytime you want. So, ordinarily, I wouldn’t give much thought to a strong password.

But sure enough, one day I went to purchase an item, and the site “helped me out” by asking if I wanted to use the same credit card I’d used the last time. All of a sudden, this went from an 80 percent site to a 20 percent site. But I can’t help but think that the average Internet user wouldn’t be thinking password security at this point and go looking for the “manage account” link. More likely, they’d just be happy to make the purchase so they could move on to their next task.

So, while I like your 80/20 observation, you can’t rule out the possibility an 80 percent site today might be a 20 percent site tomorrow.

Programming language security

In the July/August 2006 “Tools of the Trade” column, Diomidis Spinellis discusses several factors to consider when selecting a programming language for a particular project. I suggest one more: security.

By security, I mean the properties of a language that make secure coding either easier or more difficult. Many of the security vulnerabilities that can occur in programs written in C can’t happen in other languages such as Java or Perl.

Security properties should be included in the balance along with the other factors Diomidis lists. Just because C is prone to more vulnerabilities doesn’t mean that the language should never be used. Just because Java is immune to many of those vulnerabilities doesn’t make it the right choice every time.

Language security properties’ impact on a project relate mostly to how the software is developed. The choice of a fast but vulnerable language like C requires greater care to catch and correct problems in the development process before the project releases the software.

Some software vulnerabilities can manifest themselves regardless of the programming language. All projects must take steps to prevent them. The choice of programming language only influences the effort required to produce safe, secure software.

Craig E. Ward
Systems analyst parallel systems
Information Sciences Institute
cward@isi.edu

Boston University. The Department of Electrical and Computer Engineering (ECE) at Boston University anticipates openings for faculty positions at all ranks in all areas of computer engineering, including computer systems, embedded and reconfigurable systems, distributed systems, trusted computing, design automation, VLSI, computer networks, software engineering, and related areas. The ECE Department is part of a rapidly developing and innovative College of Engineering. Excellent opportunities exist for collaboration with colleagues in outstanding research centers at Boston University, at other universities/colleges, and with industry throughout the Boston area. The Department has approximately 44 faculty members, 200 graduate students and 250 BS majors. For additional information, please visit http://www.bu.edu/ece/. In addition to a relevant, earned PhD, qualified candidates will have a demonstrable ability to teach effectively, develop funded research programs in their area of expertise, and contribute to the tradition of excellence in research that is characteristic of the ECE department. Applicants should send their curriculum vita with a statement of teaching and research plans to: Professor Bahaa E. A. Saleh, Chair, Department of Electrical and Computer Engineering, Boston University, 8 Saint Mary’s Street, Boston, MA 02215. Boston University is an Equal Opportunity/Affirmative Action Employer.