DOI:10.1145/3121430 http://cacm.acm.org/blogs/blog-cacm

Assuring Software Quality By Preventing Neglect

Robin K. Hill suggests software neglect is a failure of the coder to pay enough *attention and take* enough *trouble to ensure software quality.*

Robin K. Hill
The Ethical Problem of Software Neglect
http://bit.ly/2roEDf1
May 31, 2017

Ethical concern about technology enjoys booming popularity, evident in worry over artificial intelligence, threats to privacy, the digital divide, reliability of research results, and vulnerability of software. Concern over software shows in cybersecurity efforts and professional codes.[1] The black hats are hackers who deploy software as a weapon with malicious intent, and the white hats are the organizations that set safeguards against defective products. But we have a gray-hat problem—neglect.

My impression is that the criteria under which I used to assess student programs—rigorous thought, design, and testing, clean nested conditions, meaningful variable names, complete case coverage, careful modularization—have been abandoned or weakened. I have been surprised to find, at prestigious institutions working on open-source projects, that developers produce no documentation at all, as a matter of course, and that furthermore, during maintenance cycles, they do not correct the old source code comments, seeing such edits as risky and presumptuous. All of these people are fine coders, and fine people. Their practices seem oddly reasonable in the circumstances, under the pressure of haste, even while those practices degrade the understandability of the program. Couple that with the complexity of modern programs, and we conclude that, in some cases, programmers simply don't know what their code does.

Examples of software quality shortcomings readily come to mind—out-of-bounds values unchecked, complex conditions that identify the wrong cases, initializations to the wrong constant. Picture a clever and conscientious coder finishing up a calendar module before an important meeting. She knows that the test for leap years from the numeric `yyyy` value, `if (yyyy mod 4 = 0)` and `(yyyy mod 100 != 0)`, must be refined by some other rules to correct for what happens at longer periods, but this code is a prototype … She retains the simple test, meaning to look up the specifics … but her boss commits her code. No harm is foreseeable … except that it turns out to interface with another module where the leap-year calculation incorporates the complete set of conditions, which is discovered to drive execution down the wrong path in some calculations. The program is designated for fixing but it continues to run, those in the know compensating for it somehow…

What sort of violation is neglect? It doesn't attack security because it occurs behind the firewall. It doesn't attack ideals of quality because no-one officially disputes those ideals. It is a failure of degree, a failure to pay *enough* attention and take *enough* trouble. Can philosophy help clarify what's wrong? An emerging theory called the *ethics of care* displaces the classical agent-centered morality of duty and justice, endorsing instead patient-centered morality as manifest real-time in relationships.[2,4]

The theory offers a contextual perspective rather than the cut-and-dried directives of more traditional views. While care can be construed as a virtue (relating to my prior post in this space[3]) or as a goal like justice, the promoters of care ethics resist a universal mandate. They may also reject this attempt to apply it to software, of all things; the heart of the matter for care ethics is the work of delivering care to a person in need.

Yet software neglect seems exactly the type of transgression addressed by the ethics of care, if we allow its reinterpretation outside of human relationships. Appeal to the theory allows us to identify the opposite of care, that is, neglect, as the quality to condemn. This yields our account of software quality as an ethical issue, especially piquant in its application of tools from the feminist foundry to the code warrior culture. But little credit is due! We are not solving the problem, only embedding it in the terms of a philosophical platform. This account raises issues in the ethics of engineering, such as individual versus corporate responsibility (and whether corporate responsibility can be rendered coherent and enforceable short of the law). For a concise summary, see Section 3.3.2, on Responsibility, in Stanford Encyclopedia of Philosophy entry on the Philosophy of Technology.[5]

The quality that has corrected for neglect in the past is professionalism, by which I mean that the expert does what's best for the client even at a cost to personal time, energy, money, or prestige—within reason! Certainly these judgments are subjective, and viable when the professional is autonomous, when that single person exercises control over the product and its quality. Counterforces in the current tech business world are (1) employment, under which most programmers are not consultants, but rather given orders by a company; and (2) collaboration, under which most software is the product of committees, in effect. Professionalism also depends on strong personal identification with disciplinary peers and pride in the group's traditions.

What sort of violation is neglect? It doesn't attack security because it occurs behind the firewall. It doesn't attack ideals of quality because no one officially disputes those ideals.

In the face of knotty difficulties enforcing or fostering ideals of quality, one possible resolution, odd as it may seem, is simply to acknowledge the situation, to admit to the public that software is not always reliable, or mature, or even understood. Given its familiarity with bug fixes, the public may not be unduly shocked. If we prefer to reject that fatalistic move, the pressing question is, are there some public standards that developers can and will actually follow? The collective response will determine whether software engineering is a profession. I urge all coders who wish to take pride in their jobs to read the draft professional standards,[1] which mention code quality in Section 2.1.

We see that ethical issues appear not only in the external social context, but in the heart of software, the coding practice itself, a gray-hat problem, if you will. We hope that the ethics of care can somehow help to alleviate those issues. **C**

References

1. Association for Computing Machinery. *Code 2018 Project.* https://ethics.acm.org/.
2. Burton, B.K., and Dunn, C.P. *Ethics of Care.* Encyclopædia Britannica, https://www.britannica.com/topic/ethics-of-care.
3. Hill, R.K. *Ethical Theories Spotted in Silicon Valley.* Blog@CACM, March 16, 2017, https://cacm.acm.org/blogs/blog-cacm/214615-ethical-theories-spotted-in-silicon-valley/fulltext.
4. Sander-Staudt, M. *Care Ethics.* The Internet Encyclopedia of Philosophy, 2017. http://www.iep.utm.edu/care-eth/.
5. Franssen, M., Lokhorst, G., and van de Poel, I. *Philosophy of Technology.* The Stanford Encyclopedia of Philosophy (Fall 2015 Edition), Edward N. Zalta (ed.). https://plato.stanford.edu/archives/fall2015/entries/technology/.

Note: While the Web encyclopedias, as cited, provide good surveys of current philosophical views, pursuit of any ideas in depth will require reading original research.

Comments

This is possibly the most important paragraph of the article, outlining the exact problem in the industry:

"The quality that has corrected for neglect in the past is professionalism, by which I mean that the expert does what's best for the client even at a cost to personal time, energy, money, or prestige—within reason! Certainly these judgments are subjective, and viable when the professional is autonomous, when that single person exercises control over the product and its quality. Counterforces in the current tech business world are (1) employment, under which most programmers are not consultants, but rather given orders by a company; and (2) collaboration, under which most software is the product of committees, in effect. Professionalism also depends on strong personal identification with disciplinary peers and pride in the group's traditions."

It sounds like, short of working for enlightened organizations, software developers should be leaning towards more autonomy and self-ownership.

I recently read Developer Hegemony *(a very bold title!), http://amzn.to/2pA18wB, and it addresses that side of the issue by encouraging more professionalism and autonomy.*

There's already a strong movement in favor of Software Craftsmanship, and the free software and open source movements both seem to care more about quality than most companies (though they do neglect documentation sometimes). For example, we already prefer software written by recognizably smart/professional developers.

Here's hoping to more autonomy in the future and the allowance of our professionalism to counteract the neglect of software.

—Rudolf Olah

Robin K. Hill is an adjunct professor in the Department of Philosophy at the University of Wyoming.