Embedded System Software Quality

Why is it so often terrible? What can we do about it? (Keynote Talk)

Philip Koopman
ECE, ISR, RI Depts., Carnegie Mellon University & Edge Case Research LLC
Pittsburgh, PA USA
Koopman@cmu.edu

***Abstract*— Failures of embedded system software increasingly make the news. Everyday products we rely upon are suffering from safety issues, security issues, and just plain bugs. While perfection is unrealistic, surely we can improve this situation. Two key ideas apply: (1) embedded products often aren't created by computer specialists, and (2) teaching application domain specialists just how to code is more of a problem than a solution.**

Keywords—embedded software, safety, security, quality, CPS

I. Introduction

Embedded software failures are on the rise. Thermostats leave homeowners in the cold. [1] Attackers drive a production passenger vehicle off the road via cell phone link. [2] A car company is found liable of "reckless disregard" for a computerized throttle control that resulted in a fatal car crash. [3] Malicious attackers take down a computerized power grid. [4] Without some fundamental change, problems with software quality, safety, and security will continue to grow. In an era when embedded software and Cyber-Physical Systems (CPSs) permeate our everyday lives, we have to do better than this.

II. When "Good Enough" Isn't

Software developers often eschew heavy software process. In principle, it's difficult to argue with the entrenched idea of "good enough" software. [5] A rational tradeoff approach of ethically weighing costs, benefits, constraints, and risks makes perfect sense. However, that presumes deep understanding of the unique nonlinearities, discontinuities, and technical challenges in creating robust CPS systems. In other words, "good enough" tradeoffs are only defensible when software engineering literacy is present. Otherwise, it's just guessing.

Based on our experience with about 200 design reviews of industry projects, development teams often don't have sufficient software engineering literacy to make informed "good enough" tradeoffs. Non-computer application domain experts often lack respect for the inherent difficulty of writing high quality software, and do not have the skills needed to pull it off. Computer specialists often lack respect for the challenges of deploying huge numbers of low-cost but mission-critical unattended computing devices that run 24x7 in the real world.

It is common to see a development cycle that boils down to a vague product description, writing code, and product testing. Usually missing are: design, peer reviews, unit test, and process quality assurance. Many projects also suffer from a lack of good coding practice, poor traceability, and missing difficult-to-reproduce transient software failures.

The general reaction to high profile software defects escaping to the field often includes yet more testing, and perhaps insecure on-line patching. All too often missing are the more productive strategies of: instituting effective peer reviews, using static analysis tools, actually doing a design, requiring traceability from testing back to requirements, and following applicable software safety and security standards.

III. Are They Software Professionals?

Improving the state of embedded software requires product developers to embrace the fact that they are all *software* developers. That requires them to get serious about reasonable software engineering practices. Learning how to write code – even good code – is not enough. It's necessary to know how to engineer appropriate quality software and how to tell when the result actually is "good enough" in the context of the product.

Instituting adequate software development hygiene is not so simple. Teaching computer specialists how to create good embedded software won't reach most application domain specialists. This implies that all STEM practitioners (and faculty who teach them) should learn at least the following: good coding practices; methodical testing skills; software process literacy; security literacy; and software safety literacy. Most should also be exposed to embedded software technical skills and CPS literacy as well. Mid-career training is essential even for extremely talented domain experts who find themselves thrust into the midst of large software development projects with a skill set that amounts to only how to write code.

References

[1] Billington, J., "Nest not working: Smart thermostat bug plunges customers into cold ," 1/14/2016, http://goo.gl/ZC4xP7

[2] Greenberg, A., "Hackers remotely kill a jeep on the highway – with me in it," 7/21/2015, http://goo.gl/8UHyuq

[3] Isidore,C., "Toyota settles acceleration case after $3 million jury verdict," 10/25/2013, http://goo.gl/PocJcy

[4] BBC, "Hackers caused power cut in western Ukraine – US," 1/12/2016, http://goo.gl/XYWcmw

[5] Bach, J., "Good Enough Quality: beyond the buzzword," *IEEE Computer*, August 1997, pp. 96-98.

Prof. Philip Koopman teaches embedded systems at the Carnegie Mellon University ECE Department. He specializes in improving robustness, safety, and security of cyber-physical systems. Current activite areas include stress testing autonomy software, automotive software safety, and training industry practitioners on best practices for embedded software. Previously he worked for Harris Semiconductor, United Technologies, and the US Navy submarine force. He is a co-founder of Edge Case Research LLC.