Teaching Ethics in Software Engineering Curricula: An Industry Perspective

Robert Bruce Kelsey, Ph.D.
Manager of Product Validation
Siemens ILG, Cedar Rapids, IA
robertbruce@ieee.org

Abstract

Ethics components in software engineering curricula need to focus on development discipline and risk mitigation techniques rather than on the more dramatic and socially far reaching issues of corporate deception and corruption. The students and the software industry will be better served if future software engineers understand their professional obligations and have the technical skills they'll need to solve the real challenges they will face in software development projects.

Introduction

With the publication of the IEEE/ACM *Software Engineering Code of Ethics and Professional Practice* [IEEEce], educators can point to an industry standard code of professional conduct when discussing ethics in software development or software project management courses. The Code is a laudable effort and will certainly help improve development discipline in industry. And for educators, the Code will provide both guidance and legitimacy to the classroom discussions of real and imagined dilemmas students will face in the course of their careers.

We all know that ethics discussions in the engineering classroom have to be focused on real-life issues. Kant's *Groundwork of the Metaphysics of Morals* just can't match Yourdon's *Death March* for provoking heated debate about business goals, work environments, people management, and survivor skills. There's a tendency, however, to focus ethics discussions on the "big issues" – clandestine accounting methods, public disclosure of security or data integrity flaws, etc. This is not surprising, since the profession's code and even its media focus on the social or societal aspects of ethical behavior, not on day-to-day decisions and actions [e.g., Stone].

Yet it is precisely those day-to-day decisions that collectively, perhaps even Chaotically, add up to corporate level crises and media-driven debacles. I think software engineering departments have generally looked at this day-to-day behavior as out of their scope. How their graduates behave in their cubicles is up to their employer and whatever quality management system is in place. I can understand that view, but I want to take issue with it – on both practical and ethical grounds. First, I want to explain why software quality management systems often fail to foster professional and ethical behavior. Then I will show how the quality issue is really an opportunity for software engineering curricula to emphasize a particular form of ethics training that will benefit the student, the industry, and our ultimate customer – the software's end user.

The Flaw in Software Quality Management Systems

The software industry spends millions of dollars each year on quality management systems, trying to instill the discipline and consistency in software engineering that is taken for granted in the other engineering disciplines. Unfortunately, all of the standards-based quality management systems – ISO 9001, CMM, and IEEE 12207 – suffer to greater or lesser degrees from a fundamental flaw. They're not detailed enough to help development organizations really improve processes, nor are they business-savvy enough to show cost-conscious senior management that they are getting their money's worth.

When a company's senior management commits three-quarters of a million dollars a year to a CMM consultant, they expect to see an immediate return on that investment in the measures they are most accustomed to: headcount costs, billable hours, and decreased operating costs. Instead, they see a rise in headcount to support process engineers, they see a rise in non-billable work to implement the process controls, and they see a corresponding decrease in operating margin. Typically, senior management responds to this situation by putting pressure on the software organization to cut costs and to have defect free releases.

This tension between the business metrics and process improvement costs has an immediate effect on development staff. The developers and their line-level managers find that their paychecks, their bonuses, and their career options are suddenly based on just one thing: meet the numbers. To meet the project cost estimate and end date, corners are cut, testing time is compressed, people put in absurd amounts of overtime, and time sheets and defect logs are fudged. There's no motivation for full disclosure of defect data, or of estimation accuracy, or of

cost of poor quality. There's no motivation to change processes and risk an implementation failure that causes a project slip. In short, the software quality management systems that so prominently emphasize "senior management support" often create an environment that suppresses the very behaviors the system was intended to promote: discipline, disclosure, objective analysis, etc.

Why does the software industry have to pay to teach development organizations how to do their job efficiently and effectively? That's a legitimate question, but I think it misses the point: why do the software developers and managers need the consultants and the external quality system in the first place? Why don't they know how to do their job professionally and responsibly on the date they are hired? And what has this to do with teaching ethics in software engineering curricula?

The Intersection of Quality and Ethics

The answer lies in what ethics, in action, really is. Ethics is not a body of knowledge; it is a unique form of cognition. As the philosopher Francisco Varela expresses it, ethical action is know-how, not know-what; it is closer to performing a skill than it is to analyzing the performance [Varela]. When faced with an ethical decision, we project ourselves into a course of action based on our values, our vision of what our actions will produce, and most importantly for our purposes here, our knowledge of the means to achieve our desired ends. For the most part, software practitioners in the industry today don't know the means to the end. They learned the programming and modeling languages. They can recite the PMI project phases. But they don't know how to manage software development or how to improve the development process. They don't know how to proactively plan for, or how to mitigate, negative consequences, even though that is perhaps the most critical skill required in software development!

So I whole-heartedly agree with Donald Gotterbarn of the Software Engineering Ethics Research Institute when he says we need to teach "technical ethics" [Gotterbarn1, 2]. The goal of teaching technical ethics is to have students understand and adopt the standards, methods, and values that make for highly reliable and fully functional software products. For our purposes here, that means that students need to learn basic quality assurance and risk mitigation tools – estimation techniques, peer review methods, root cause analysis techniques, and statistical process control. They need to learn how to diagnose and correct failures – in development

processes as well as in developed product. They must understand process improvement models and techniques, defect removal strategies, and how defects in one lifecycle phase propagate to subsequent phases. Students must be able to plan incremental process improvement efforts to balance cost, productivity, and risk. They must be reasonably proficient writers and presenters, and they should have a basic understanding of psychology and communication theory as they relate to requirements development and product ergonomics.

I understand that this exceeds the usual curriculum for a BSCS. But there's no such thing as a "software developer" anymore. Every software developer is also, for at least some part of his or her day, a systems requirements engineer, a technical writer, a test engineer, a project manager, and a marketer. And I also understand that my wish list above doesn't look very much like a course module on ethics. But that's exactly the point – we don't need to teach students to be good, we need to teach them to be capable, productive, and proactive software engineers. Don't use Yourdon's *Death March* to spur discussions of the big issues. Use Humphrey's *A Discipline for Software Development* to teach the students the Personal Software Process, and in the process show them how to act, daily, as professionals, aware of and outfitted to tackle their obligations to their employers, to their software's end-users, and to society at large.

REFERENCES

[IEEEce] IEEE/ACM. "Software Engineering Code of Ethics and Professional Practice," //computer.org/certification/ethics.htm

[Gotterbarn1] Donald Gotterbarn. "An Evolution of Computing's Codes of Ethics and Professional Conduct," http://www-cs.etsu.edu/gotterbarn/artge1.htm

[Gotterbarn2] Donald Gotterbarn. "The Moral responsibility of Software Developers: Three Levels of Professional Software Engineering," *The Journal of Information Ethics,* Spring 1995, pp. 54-64.

[Stone] Adam Stone, "Software Flaws: To Tell or Not to Tell?", *IEEE Software*, January/February 2003, pp. 70-73.

[Varela] Francisco Varela, *Ethical Know-How: Action, Wisdom, and Cognition*, Stanford Univ. Press, 1999.