What Do Software Developers Need to Know about Business?

Warren Harrison

A recent discussion I had with colleagues from my university’s business school and computer science department focused on identifying the most critical knowledge for software developers. My computer science colleagues’ perspective was quite interesting. They acknowledged that once a software developer has managed to climb into a second- or third-level management position, maybe an MBA wouldn’t be such a bad idea. But for the most part, they held the strong belief that anyone smart enough to be a computer science graduate must be able to easily pick up this “business stuff” on the side.

Type a or type b?

I felt the same quick surge of irritation that I usually get when dealing with students who want to bypass prerequisite courses and enroll directly into an upper-division software engineering course. They seem to think that software engineering can’t be all that difficult compared to the mysteries of homomorphisms and context-free grammars. It’s either (a) the height of conceit or (b) the height of ignorance about the depth of the field to suggest that, without serious study, someone could pick up a field that has taken others years to master. Over the years, I’ve come to realize that (b) is more common than (a).

So, whenever I hear someone talk about either business or computing issues as things to be “picked up on the job,” I can’t help but wonder whether the person is “type a” or “type b.” That’s not to say that someone can’t learn material from either discipline through self-study—but it requires a serious application of discipline and effort.

Understanding the context

I’m less concerned about those who have chosen to pursue a management career path. Over time, I’ve come to believe there’s a real need for the average software developer to understand and appreciate the economic context in which their company operates.

Without a thorough understanding of the business context, we tend to operate in a fog, believing things happen arbitrarily by chance and can’t be predicted. Primitive cultures’ lack of understanding of physical phenomena led to a belief that people became ill, soldiers won or lost battles, and storms devastated villages because of angry “gods.” Individuals felt a loss of control over their daily lives and resorted to illogical attempts to placate these “gods” through sacrifice and the construction of grand temples.

I’ve observed developers’ bitter resentment when they try to come to grips with senior management’s seemingly illogical decisions and actions. Often, as in ancient mythology, they attribute these actions to angry “gods” who ar-
FROM THE EDITOR

A single principle can make all the difference.

Sunk costs are costs that have already been incurred and cannot be recovered. For example, if a company has invested money in a project and then that project is canceled, the money spent on the project is a sunk cost. Sunk costs are often cited as a reason for continuing a project even if it is not profitable, but this is not always the best course of action.

Instead, it is important to consider the opportunity cost of each project. Opportunity cost is the value of the next best alternative. For example, if a company has invested money in a project and then that project is canceled, the money spent on the project is a sunk cost. However, that money could have been used to invest in another project that would have been more profitable. The opportunity cost of the canceled project is the potential profit that could have been earned by investing the money elsewhere.

It is important to remember that sunk costs are not relevant to future decisions. Once a decision has been made, the past cannot be changed. It is important to focus on making the best decision for the future, not just to avoid losses in the past.

Ultimately, it is important to consider the opportunity cost of each project and make decisions based on the potential for profit, not just on the sunk costs that have already been incurred.
So what can we do about it?

Many years ago, when I was in graduate school (1978–1984), it seemed that one person could know a little bit about almost every aspect of computer science. The body of knowledge was simply that small. Nowadays, with more and more subfields sprouting up, it’s hard for students to know even what all the areas are, much less a little bit about each one. So, it’s difficult to justify a large number of nontechnical courses simply to “gain context.” Luckily, software developers can get by with far less—after all, the idea is to establish a context, not to turn out investment analysts.

Several years ago, Bruce Schafer (founder of PC-Kwik, for you old-timers) and I put together a course for the Oregon Master of Software Engineering program called “Understanding the Software Business.” This class undertook the Herculean task of introducing students to the marketing, financial, and legal aspects of the software industry in 10 weeks. It involved general pricing strategies, basic macroeconomic concepts, capital budgeting issues, and in-strategies, linear algebra, or physics.

We found that students (all professional software developers with several years of experience) came into the class with a lack of exposure to economic and financial topics, a misunderstanding of the concepts of time value of money, and little knowledge of pricing strategies.

Although such a class clearly can’t turn students into experts in any of these areas, it can provide them with the tools to better understand the business aspects of developing software and how decisions are made at the executive levels of their organizations. Such a course can go a long way toward preparing students for a career in software development and is likely to be far more valuable than many of the traditional undergraduate requirements such as differential equations, linear algebra, or physics.

Feedback welcome

What do you think? Have you had any experiences dealing with executive-level management at work? Do you find that your company uses formal capital-budgeting techniques in evaluating projects? If you have training in business, economics, or finance, have you found that it has benefited you in your role as a developer? Please write me at warren.harrison@computer.org. 

Next Issue

Predictor Models in Software Engineering

Software engineering is a decision-intensive discipline. Can we build models that make explicit the knowledge hidden in software resources? Can we use models to make better decisions? Can we assess those models? Do different researchers working on the same data arrive at similar models? Are there better, faster, cheaper ways to build software engineering models?

Many predictor models already exist but information about them is proprietary, so model calibration in different development environments is difficult. This issue will look at how to build predictor models and what we’ve learned from open source data sets.

Visit our Editorial Calendar at www.computer.org/software/edcal.htm