Programming Management 101

Version: 1.1 Date: November 25, 2008

In my experience, many people without significant experience in programming harbor a number of misconceptions about programming. By far the most common is the seemingly common sense expectation that experienced programmers should be able to accurately predict the schedule of the project. This leads to many misunderstandings, bad decisions, and frustrating experiences.

Most programming is unpredictable. Sincere cost and schedule estimates tend to be low, often by factors of three or four. Most substantial programming projects encounter unexpected problems, so-called "unknown unknowns" or "unk unks" in technical jargon. This happens in part because one almost never does the same programming project twice.

The most common source of misunderstandings about programming are analogies to repetitive or mostly repetitive physical tasks. Both the popular and technical programming literature is full of analogies to construction projects such as building a home, erecting a large building, or ship-building. Most popular project management techniques such as Gantt charts, Program Evaluation and Review Technique (PERT), and Critical Path Method (CPM) have their roots in large construction projects. These project management methods were adapted to Research and Development and later software development by military organizations such as the Air Force and Navy. These methods have been widely adopted in the private sector and are incorporated in widely used project management products and systems such as Microsoft Project.

The underlying analogy of R&D and software projects to physical construction projects is, however, flawed. Military and other government R&D and software projects managed using these techniques derived from construction are notorious for large cost and schedule overruns. For example, the Department of Defense's Space Based Infrared System (SBIRS) is currently (2008) at least 400% over its original budget. Software development cost and schedule overruns have specifically been cited as a major contributor to SBIRS problems. Other recent R&D programs with major cost and schedule overruns include the Air Force's Transformational Satellite or TSAT, the BASIC

reconnaissance satellite program, the NPOESS meteorological satellite, and the classified Future Imagery Architecture (FIA). In part, this is because these project management techniques simply don't work even when applied honestly and sincerely.

In many physical tasks, once the performer has learned the task and performed it several times, they can perform the task very predictably over and over again. For example, at a restaurant, once an employee learns to make a sandwich, they can make the sandwich over and over again in a few minutes with only a small variation in the expected time to complete the task. Similarly, some construction projects are very predictable. It takes so much time to hammer a nail into a two-by-four, so much time to assemble several two-by-fours into a frame, and so forth. If the requirements are not changed in the middle of a construction project, the construction project manager can often predict the required cost and schedule accurately because he (or she) is performing essentially the same task over again.

In software, the end product, the program, can be replicated at essentially no time and cost (e.g., copy program.exe program_copy.exe). Hence, there are almost no repetitive or mostly repetitive physical tasks. Programming is analogous to intellectual tasks such as learning something new, developing a new architectural design, and so forth. Most non-trivial programming projects require a significant amount of trial and error. With more experience, the amount of trial and error required is reduced, but it cannot be eliminated.

The bottom line is that programming is significantly less predictable than many forms of physical work such as construction. Unexpected problems, “unknown unknowns”, are normal. The original budget and schedule estimates for research and development and software projects usually do not include adequate reserves for unexpected problems, which often should be larger than the identifiable costs. A common joke is: to get the real schedule, multiply the official schedule by PI (3.14) for running around in a circle. In general, large reserves should be budgeted up front for unexpected problems.

About the Author

John F. McGowan, Ph.D. is a software developer, research scientist, and consultant. He works primarily in the area of complex algorithms that embody advanced mathematical and logical concepts, including speech recognition and video compression technologies. He

has many years of experience developing software in Visual Basic, C++, and many other programming languages and environments. He has a Ph.D. in Physics from the University of Illinois at Urbana-Champaign and a B.S.in Physics from the California Institute of Technology (Caltech). He can be reached at jmcgowan11@earthlink.net.