The Programmer Life-Cycle

Russell Ovans
Department of Computer Science
University of Victoria
Victoria, British Columbia V8W 3P6 Canada
rovans@csc.uvic.ca

Abstract

The traditional analysis of programmer productivity ignores the reality that work rates of programmers are variable over time. Not unlike the software systems they construct, programmers follow a predictable life-cycle. However, the programmer life-cycle is not comprised of activities but rather by phases that directly affect and predict productivity. The sequence of phases is: euphoric, productive, irreplaceable, resentful, bored, and unproductive. Overall productivity is characterized by an initial six month period of intense interest, at which time productivity rates are often an order of magnitude higher than the oft-quoted 500 LOC/month average. After a short period of volatility, the programmer then enters a prolonged phase of steadily dwindling interest, resulting in productivity rates that mimic the average. Each time a programmer switches employers or begins a significantly new project, the life-cycle starts anew.

Keywords: Programmer productivity, software life-cycle model.

Introduction

Most textbooks on software engineering make at least passing reference to the notion of programmer productivity. The conventional wisdom is that an average programmer completes 500 debugged lines of code (LOC) per month [3]. To students and professionals alike, this number seems absurdly low. The standard set of explanations for this phenomenon includes, among other things, the communication overhead of working in teams (i.e., the Brooks' Law [1]), and the time allotted to non-programming activities like requirements gathering and design.

These explanations tell only part of the story. While it has long been accepted that productivity rates can differ by an order of magnitude between the best and worst programmers [2], little has been said regarding the reality that work rates of the same programmer can likewise differ over time. Not unlike the software systems they construct, programmers follow a predictable life-cycle. However, the programmer life-cycle is not comprised of activities (e.g., requirements, design, implementation, testing, and maintenance) but rather by phases that directly affect and predict productivity. The sequence of phases is characterized by an initial six month period of intense interest, at which time productivity rates are often an order of magnitude higher than the oft-quoted 500 LOC/month average. After a short period of volatility, the programmer then enters a prolonged phase of steadily dwindling interest, resulting in productivity rates that mimic the average. Each time a programmer switches employers or begins a significantly new project, the life-cycle starts anew.

This conjecture is based purely on my experiences and observa-tions over the past six years while working as the senior software engineer for two successful Internet startups (one in Canada, the other in the United States). The basic premise of the proposed relationship between phases of the programmer life-cycle and productivity is the simple observation that employees are most productive when interest and satisfaction in their jobs is at its highest.

The Phases of the Programmer Life-Cycle

The programmer life-cycle is comprised of six phases:

- Euphoric
- Productive
- Irreplaceable
- Resentful
- Bored, and
- Unproductive

While this particular life-cycle model is perhaps most likely to apply to highly productive individuals (so-called *star programmers*) working in the 24/7 world of web development and e-commerce, it is my belief that there are fundamental truths in its structure that render it applicable to many software development situations and domains.

Euphoric. The first phase involves the start of a new job or significantly different project. The programmer is stimulated by both the newness of the environment and the challenges ahead. In many cases the programmer's euphoria is fueled not only by these changes, but also by the coincident escape from a previous work situation that had become routine or was underutilizing his/her talents. This introductory phase is quite short in duration, lasting only as long as necessary to acclimatize to the new surroundings, learn a new development environment, and become familiar with the domain.

Productive. Once acclimatized, the programmer's work interest reaches a peak. Because of this acute interest, productive is at its highest. This phase - which lasts about six months - is when the programmer develops or takes ownership of some mission-critical software system. This coincides with steadily rising value to the organization.

These first two phases constitute the "honeymoon period". The next two are best described as the "volatility phase".

Irreplaceable. Management soon recognizes that the programmer has become a valuable commodity. The programmer's stock within the organization is at an all-time high. As such, significant increases in compensation and fringe benefits are doled out in an effort to keep the programmer from leaving. The programmer feels on top of the world. Unfortunately this will not last.

Resentful. Management - sensing a sudden asymmetry in the employer-employee relationship - begins to bear resentment that a single individual (the programmer) is now responsible for the on-going success or failure of the venture. Fearing a loss of control, management begins to act in a manner suggesting ownership of the programmer's time and space. Symptoms of this malaise include requiring the programmer

to carry a pager, work weekends, install broadband connectivity at home, and never under any circumstances take holidays. The programmer starts to receive email at all hours of the day requesting new features and emergency bug fixes.

In return, the programmer develops feelings of resentment towards management. This is exacerbated by management's policy of rewarding the programmer's competency not with bonuses and time off, *but with additional workload and responsibilities.* The first signs of complacency begin to appear in the programmer's workplace attitude.

This unstable time of mutual resentment is necessarily short-lived, as emotions run too high for the process to carry on for more than a month or two. The working relationship can implode during the resentful phase, particularly if volatile personalities are involved. In the worst case, the programmer quits; the additional workload coupled with the stress of being irreplaceable yet resented becomes too much to take. However, in most cases the resentful phase merely settles into an equilibrium of mutual need: management's need for the star to carry on keeping the software running, and the programmer's need to be a star.

Bored. The post-resentment equilibrium sees the programmer's activities shift more towards ongoing maintenance, consultative meetings with management, and internal knowledge transfer to other programmers and customer support staff. Because the initial challenges of the new project, environment, and technologies have all been met, the intellectual stimulation has dropped. This leads to boredom. Coupled with the excessive mental context switching demanded of the new activities, the programmer's productivity (as measured by LOC/month) experiences a significant drop. Despite the tedium, however, this phase can last indefinitely provided the productivity remains above the minimum expectation level given the programmer's current remuneration.

Unproductive. Like a manic depressive who goes off his medicine because he misses the occasional euphoric episode, or a love junkie addicted to the adrenaline rush of the first six months of a new relationship, the programmer is unlikely to remain in a state of boredom forever: something has to give. The change is triggered by a slide into the unproductive phase, characterized by the programmer working on his/her resume and visiting job sites on the 'Net, while management views the programmer as "coasting", overpriced, and expendable. One of two outcomes is inevitable: the programmer finds a new employer, or management moves the programmer to a significantly new role or project. Either way, the life-cycle starts again.

Conclusion

This life-cycle model should serve as a cautionary tale to both programmers and managers. The lesson for the programmer is to be aware of each phase and its effect on productivity levels, for ultimately one's success as a software engineer depends on one's perceived productivity. By recognizing the symptoms of boredom leading to unproductiveness, the programmer can proactively search for remedies, usually in the form of a frank discussion with management, or seeking out new projects and technological chal-lenges.

Conversely, managers must understand the causes and effects of this life-cycle in order to combat high levels of attrition and declining productivity. To get the most out of the organization's stars, managers must avoid the resentment trap by resisting the tempta-tion to over-burden the irreplaceable programmers with additional responsibilities. Instead, managers should look for challenges that will keep the stars at their peak performance level.

References

[1] Brooks, F. P. Jr. (1975). *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley.

[2] Sackman, H. H., W. J. Erikson, and E. E. Grant (1968): Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, 11 (1), pp. 3-11.

[3] Van Vliet, H. (2000): *Software Engineering Principles and Practice* (Second Edition), John Wiley & Sons, p. 175.

Editor's Filler

Want More?

Sure you do!

We all do.

You can see we have some room.

You can be part of the solution, not part of the problem!