Slack, Flow and Stealing Ideas

Enjoying Creative Programmers

Thomas Engelin
© Itancan Consulting AB 2004
1. Abstract

In this document, three parameters necessary for a productive and creative programming environment are explained. These alone do no guarantee success, but without them software quality will for sure drop over a longer period of time.

1.1. You should read this document if...

✓ You're in the software business.
✓ You care about your company's long-term economics (who doesn't?).
✓ You believe that happy programmers create better software.

The three parameters I'm talking about are:

Slack -- non-dedicated time in a project (project spare-time).
Flow -- a programmer's most productive and creative state of mind.
Stealing -- the need to find inspiration and the need to copy solutions.
2. Slack

There must be time for the programmers to sit back and think about the system they have built, or are building. A project leader that allocates the programmers to 100% in each new project is the one to blame when quality later drops so much that for each new function the programmers put in, at least one new fault is discovered.

To keep code quality on a decent level there must be time for refactoring. The need for refactoring usually increases with the age of the system. In each project there should be some slack so that when the programmers suddenly see the need for a larger redesign, there is a chance that they can do the job without risking a late delivery.

Slack also enables creativity. A very busy programmer is not a creative programmer. The programmer might still solve her daily problems in a creative way, but there's no chance for new ideas to grow or for new or better solutions to be evaluated.

If a programmer is never given the opportunity to reflect upon past work, she cannot learn from history, and the same defects and bad design will continue to go into the system.

Programmers are creative and quality-aware by nature. They don't like to mass produce code. Like other craftsmen they have pride in their work and feel bad when they see code that they would like to improve but are not given the opportunity to.

Keep you programmers stimulated. Give them slack.
3. Flow

It takes about 20 minutes for a programmer to enter “flow”. A typical indication that a programmer has entered flow is tunnel vision and the impression that time stands still.

During flow creativity is maximized and the quality of the code produced is high. The productivity peaks and the programmer fully enjoys the ride and feels very stimulated.

As soon as the programmer gets interrupted, this state of mind is lost. Another 20 minutes is required to get into flow again. Each interruption, no matter how small it is, makes the programmer irritated, increases the risk for bugs, and costs another 20 minutes.

At some places, programmers get interrupted all day long. So they never enter flow. Just imagine how much money they loose!

The only way to deal with this is to introduce interrupt-free time-zones or rooms. Disconnect the telephones, prohibit meetings, shut out annoying noises. Why don’t introduce a couple of meeting-free days per week? Or at least forbid meetings before lunch? Research has shown that the time before lunch is the most creative time during the workday.

This also affects the office facilities. A programmer must be given the opportunity to close the door to her room and work undisturbed for longer periods of time.
4. Stealing Ideas

At least 95% of all code that gets written is more or less a copy of what someone else has written before. Standard solutions with minor variations are used almost everywhere. I think it is fair to say that the entire software industry is based on a number of prevalent architectural patterns, design patterns, and coding conventions.

The “problem” is that these standards, patterns, and conventions are all but static!

Many companies don’t believe in education. They think that you as a programmer grow if you constantly work with your own system day by day, year by year. Well, you don’t. You learn that particular system but that’s all. As soon as a problem appears, you use your same old toolbox to fix the problem because your employer can’t understand the reason for learning about other tools, systems, or languages.

Even worse, when your current product gets outdated and management decides to build the next generation of your product, it will be built using yesterday’s toolbox. The same solutions and the same problems will for sure appear in this “new” generation as well.

No wonder that programmers constantly move around looking for exciting projects and understanding employers. They say that’s the way programmers are made – they never stay for more than a few years. I say that’s the way management makes them.

Even if you work as a web designer, you can benefit from a course in real-time design. Maybe you won’t create a better web solution the following day, but you will for sure get a broader understanding of programming as an art. You grow as a programmer and you start to look at your system from another angle.

You see new solutions to old problems. You see new problems in old solutions.
5. Conclusion

It doesn't require much to satisfy these basic needs for a productive environment.

Regardless of if you choose to follow my guidelines or not, don't neglect the boost in quality, creativity, and productivity that follows with a happy and stimulated programmer!