Getting out of the comfort zone

Gary Pollice

November 15, 2007

We've all heard the term "comfort zone." Each of us has one. It's the place where our life is predictable, routine, and stress-free. When we're in our comfort zone, we feel safe. We block out those things that cause stress and make us anxious. Most self-help books, life coaches, business motivators, and others whose business it is to help us "achieve our potential" preach about the need to go outside of our comfort zones and expand our thought processes, skill sets, and so on.

I'd always been a big believer in moving outside of the comfort zone. I understand the angst associated with leaving the nest of comfort and predictability and being forced to spread our wings and fly solo. I've always thought, however, that once you make the move it's pretty easy to adopt even more new things and keep moving forward. Part of adopting new practices and habits requires us to free ourselves of old habits. This can take a bit more effort, but once the chains are broken, it seems like the effort required to prevent lapsing back to old behaviors should be minimal.

I'm not so sure about this anymore, however. This academic year, as in the last few years, my schedule started out by teaching a course in object-oriented analysis and design (OOAD). I tried something a little different this year and the results were surprising. They have caused me to re-evaluate the whole idea of comfort zones and the need for discipline in even the simplest things.

Background

At Worcester Polytechnic Institute (WPI), we have two undergraduate courses that focus on software engineering practices: CS3733: Software Engineering and CS4233: Object-Oriented
Analysis and Design. I've written about both of these courses before in this column. The OOAD course is usually taken after taking the software engineering course, although some students reverse the order due to scheduling constraints.

I make the assumption that the students taking OOAD have had software engineering. This means they've worked on a project with a team of ten or more people. They understand basic software development practices like iterative development, planning their work, requirements management, testing, and so on.

We have four seven-week terms at WPI. This is very different from most schools. A student typically takes three courses per term. For both of the courses mentioned above, the students have twenty-eight class meetings. In order to drive home iterative development in software engineering, each team must demonstrate working software each Friday rather than attending a lecture. At the end of the course they seem to really "get" the benefits of iterative, incremental development. Many students' reflections on the course point this out as one of the most useful things they learned.

The OOAD class focuses upon the technical details of developing object-oriented systems. There is a significant project developed in this course, but the teams are much smaller -- usually three to five students per team. The project is also different because the teams have to develop a framework rather than a complete application. The goal is to produce an executable architecture, such as might be produced during the Elaboration phase of a project following the IBM® Rational Unified Process® (RUP®). What I have done for the last two years is use this executable architecture as the starting code for the software engineering courses. This frees up a lot of my time because I only have to define a single project for all of my software engineering related courses each year.

Previous offerings of OOAD were structured similarly to the software engineering course. Every Friday the team demonstrated the work from the previous week's iteration. This year I discontinued the weekly iteration demonstrations in this course. I did this for several reasons. First, there are more teams and it simply isn't possible to spend enough time with each of them during a single class meeting. Second, there is more material that I want to present in the course and the extra lectures are more than welcome. I wanted to see how well the teams would work without the demand for weekly demonstrations. Would they set up weekly iterations for themselves? Would they manage requirements effectively?

OOAD 2007

This year the OOAD teams had two checkpoints during the term where they submitted their work before their final project presentation. They had to describe their designs, explain their architecture, and so on. We used a modified RUP Software Architecture Document as the outline for the teams' deliverables. They also had to submit their code. However, there was no demand that a certain amount of working code be completed by any specific milestone, except for their final submission when the architecture had to work. If they were practicing iterative development, working code would be a natural consequence.
The 2007 project was a bit more abstract than most of the projects I've used. I wanted the students to implement a framework for representing software development processes. My idea was to have the software engineering courses implement an executable process in the form of a multi-player dungeon game. Since we discuss software process in the software engineering course I felt the problem domain was somewhat familiar to the students. It turned out that this really wasn't the case. They understood how to decide what practices to use and when, but they hadn't had enough experience with processes in general to be comfortable working with a more formal process model.

Of course, the goal of the class is for the students to learn design and analysis, not to build software that I can use in other classes. The students definitely learned the core concepts and were able to apply them. But, I have to admit that the project is the first one where my students weren't able to get to deployment in seven weeks. I've always suspected this might happen and I finally found their limits.

**Learning is not knowing**

Each team in the OOAD course must make a 30-minute presentation at the end that describes their work. They have to talk about the technical details, the architecture, patterns used, and so on. They also have to talk about their process, the tools they used, and give a description of what went right on the project and what went wrong. For the things that were not optimal, they need to discuss how they might change for the next project they work on. Every individual must also turn in an individual retrospective on the project. This is a chance for each person to talk about his or her contribution to the team as well as any personal observations they had that might not be generally shared by the whole team.

Imagine my surprise when over half of the teams admitted that they followed a fairly straightforward waterfall process! This was, in fact, why so many of the projects did not have enough of their framework implemented. The teams didn't come right out and say that they followed a waterfall process. They said things like: "We should have started coding earlier," or "We took too long to get our design ready." Using these statements as a starting point for our conversation I probed a bit deeper into what happened in the project. In general, the results were not that surprising, but they indicated that I let my guard down and made an assumption about whether the students had not only learned the lessons in software engineering, but whether they had internalized them so that they would apply the knowledge appropriately. I found some common threads that back up the results of other studies that indicate how easy it is to fall back to a comfort zone, whether we realize it or not.

**Habits take time**

It's not surprising to learn that if you want to make something a habit, it's going to take some time. How long will vary according to the habit you want to adopt and who you are. People today, especially those who want to be better at something, seem to try to change themselves continually by adopting new habits. If you're overweight, you try to eat better. If you want to be in better shape, you have to make going to the gym or going out for a run a habit.
Habits are things that we just do without thinking. When faced with a situation where we have several courses of action, we want to take the proper course because we have made it a habit. Our ideal state is one in which we've got a large collection of good habits that we know will serve us well in our day-to-day living.

If you've made something a habit, then you've gotten to the point where you act instinctively. This is much different from learning a new behavior. When it's a habit, you've internalized the behavior. Clearly, in the case of my students, they had learned how to work iteratively but they had not made it a habit. So, this led me to think about the skills that are habitual in great software developers and about habits in general. I spent a little time researching habits and reflecting on my own experience. I offer the following observations about habits and hope they give you some things you can use on your current or future software development projects. I don't claim that any of these observations are necessarily new, but I suggest that if you are aware of them you probably don't think about them when you deal with your project teams and your own practices and skills.

**Practice, practice, practice**

You may learn a new skill or practice but that doesn't mean that you are able to perform well with it. It takes practice. We all know this. It's why athletes spend many hours a day practicing fundamental skills. Our muscles have to be trained to react in certain ways to certain stimuli. We know that repetition works to get the muscles in proper shape. Musicians require the same type of practice. Watch a world-class musician practice and you'll find that she spends a good amount of her practice time warming up with scales, chords, and various etudes. A virtuoso must first be an excellent technician who has made the basic practices habitual. They perform them without thinking.

Practice however, must be properly focused. If a distance runner practices speed work for the whole practice every day, he will not improve his endurance. There needs to be a mix of speed and endurance work. Musicians, analogously, must spend time practicing those things they already know as well as devoting time to learning new techniques and musical pieces.

Software engineering students must also practice what they learn. Whatever practice they have programming in any programming language will improve their programming style and add to their toolbox of problem solving techniques. Programming in a specific language improves their understanding of the capabilities of the language and how to best express ideas in the specific coding idioms. Students also have to practice other skills they're learning; so much so that their lives must seem to be spent simply practicing. Their problem is one of time management. When you have different skills you're trying to learn and make habitual, life gets much more complex. It also becomes harder to develop habits when you're trying to develop more than one at a time.

Professionals also need time to practice. You hire someone to do a job. If you have a specific problem that requires special skills you hire a specialist who has those skills. But what happens when that problem is solved and there are no more problems requiring that skill set -- what do you do then? If you're a smart organization or manager you give that employee time to learn and practice new skills.
Many, perhaps most, companies today have learning goals for their employees. Each employee agrees with the manager on the goals and together they develop a plan for how to meet the goals. The plan usually includes time and resources for the employee to attend a class. The class may be an internal education course or an external course or conference. But after the training, what happens? Too often the employee doesn't have time to practice the skill and it quickly becomes a non-skill. When the employee needs to use the skill on a future project they have to re-learn it, almost from the beginning.

Here's a simple test to prove my point. Many readers will have taken a calculus course in their undergraduate training. I'm sure you remember some of the simpler rules for differentiation and could quickly tell me that the derivative of $x^2$ is $2x$. Are you able to provide the answer to this problem?

$$\int_0^2 x^2 \, dx$$

I was a math major many years ago before I did graduate work in computer science. But, I use discrete mathematics much more than calculus these days, so I need to go back and look at how to find the answer. And let’s not even talk about differential equations. The point is that I have lost the skills that I had when I was a student because I haven't been practicing them.

We lose skills, both physical skills and mental skills, when we don't practice them. When we learn a new skill we have to practice continually in order to retain the skill. Maybe we don't have to practice as much when we've internalized the skill to the point that it's a habit, but we have to practice it somewhat or it will atrophy.

**Coaching**

I mentioned that practice needs to be focused. This is the role of the coach, advisor, or teacher. Whatever name they go by, this is someone who understands the process of incorporating a skill to the point of making it a habit. I came to realize how important this job was as I thought about my OOAD course and reflected on some of the experiences that I've had working with business organizations.

There have been many times that I've been asked to help a project team improve its performance. Usually when I meet the team I find that there are project members with as much experience and skills as I might bring to the team. The team is often trying to improve in a few areas, such as becoming proficient with iterative development and scheduling their work. This happened last summer when I worked with a team that had exactly these needs.

I don't want to trivialize any practice, but learning how to work in iterations and use a method like the XP Planning Game is not hard. The key is that the team needs to practice it and have someone there to critique their performance and help them get better. That's the biggest thing the coach can bring to the table. The coach helps the team members get out of their individual comfort zones in non-threatening ways. Yes, the coach should be technically competent, but the team
members are usually just as competent. The coach simply has to know when to push or prod and when to let the team members fly on their own.

As a teacher, I need those same skills. I am not going to give the students all easy problems. But if I give them all really hard problems, they don't get the chance to develop the good problem-solving habits that they'll get with the easy ones. The key is to balance it so the student gains confidence and have an easier time leaving their comfort zone the next time.

**Motivation**

I've always known that you have to want to change the way you do things if you're going to learn new skills and develop new habits. And I always equated this with motivation. But it's not the same. Wanting something does not mean that we will act to get it. However, motivation means that we are motivated to do something -- we have a motive. And the dictionary says that a motive is *something (such as a need or desire) that causes a person to act.*

A real difference exists between wanting and doing. The key is motivation. A good example of this is what happened with my OOAD students this year. They knew that developing iteratively was better than using a waterfall process. They knew that coding early was better. However, I gave them no motivation to step outside their comfort zones and act on what they knew. Unlike the software engineering course, they weren't motivated to have working software each week. I don't think the issue was one of falling back into old habits. It was rather staying in their comfort zones since they hadn't yet internalized iterative development. If I had demanded working software each week they would have responded and they would have been one step further into making iterative development part of their comfort zone.

I failed my students. After five years at WPI I'm still learning how to be a good teacher and am working at improving my teaching skills. I'm learning a lot about motivation. I wonder how many project leaders and managers realize how much motivational power they have. Motivation isn't about setting unrealistic goals and trying to force people to reach those goals through draconian rule. It's about giving the people a motive to act rather than just want to reach the goals. Giving people ownership of their goals and making them accountable goes a long way towards motivating them.

If you look around for ways of motivating people, you'll find them all around. Consider some of today's weight-loss programs. One of the more successful ones is where people meet regularly and learn how to eat better and encourage each other. The big motivation, however, is the weigh-in where you get to see your progress. If you have lost some weight there are congratulations and if you haven't there's no blame, but a repetition and reinforcing of the practices that will get you to the goal. This program is more effective than others because there is motivation.

**It's like riding a bicycle ...**

Whenever we try to perform some action that we haven't done for a while, some well-meaning friend might tell us that "It's like riding a bicycle, you never forget how." Well, I hate to debunk this belief, but it's just not so.
Just as we can develop new habits, we can get rid of habits. Sometimes this is a very difficult task. Some habits are aided by outside forces such as nicotine or alcohol addiction. Getting rid of these habits often involves adopting new habits to replace them. But there are other habits that we can get rid of very easily. Usually it's a habit we've worked hard to adopt, often one that replaces a less desirable habit.

Consider a person who has worked hard to lose weight and change the way they eat. They get to their goal weight and seem to be cruising along until they begin to make some slight modifications. Maybe it's a little bit of sweets after dinner or a bag of pretzels. Soon, the little extras become the normal routine and more extras get included. Pretty soon all the weight that required so much effort to lose is back and the person has to start over again. Losing an acquired habit is so easy. We lapse back to the old comfort zone because -- well, it's comfortable.

We also revert to old practices when we're under stress. Stress causes our “flight” response to kick in and we flee to our comfort zone. In my OOAD class that's exactly what many of my students did. Since they had not made iterative development a habit, when they were stressed by trying to tackle a particularly abstract and difficult problem, they chose to work from their comfort zone. There was no motivation to add the extra work (in their opinion) of iterative development.

**Conclusion**

I'm sure you'll agree that what I've said isn't new. I hope that you'll take a few minutes to reflect upon the thoughts above and identify some places where you can make a difference as a manager or a team member. Find a way to help someone adopt and retain a new habit and gently move them out of their comfort zone. You'll both be glad you did.

Meanwhile, I'm going to set my alarm so I get up an hour early every day this month and practice my writing. I want to be a better writer and maybe my muse will provide the motivation.

**Notes**


2 I don't like reusing projects since one thing students are very good at is locating previous similar projects. While we want our students to learn how to reuse rather than reinvent, we don't want to encourage plagiarism.


4 By the way, the answer is 8/3.
5 See http://www.xprogramming.com/xpmag/whatisxp.htm#planning

6 Merriam-Webster OnLine Dictionary: http://www.m-w.com/

7 I speak from experience here. If I don't strictly follow my weight loss and maintenance program I'm doomed to put on the pounds again.

8 My doctor has reminded me more than once that "fat tastes good," which is why I have such a desire for junk food.