Do Not Flounder

I have no idea whether or not most developers using Agile have actually read the "Agile Manifesto." Here it is:

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

This post is more about career growth than Agile, but somehow I think the Agile approach is relevant here. Agile, as we know it, is an approach to software design. It can also be an approach to managing one's own career.

Floundering

A friend of mine recently said this: "Your attitude determines your altitude." Although he was speaking in a general sense, I couldn't help but think of the application of this motivational advice in relation to software engineering. It is relevant because as software engineers, our career growth is in our hands – perhaps to a greater degree than in any other field.

My first "real job" after college was working for a large defense contractor. I had interned with the company and was offered a position after graduation. Naturally, I was thrilled about the offer and the comfort of knowing that I had a job waiting as soon as I wrapped up my last year of Computer Science at Ball State University.

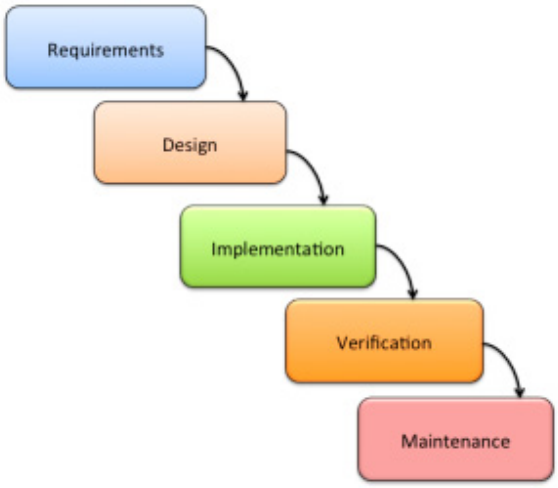

Figure 1. *Remember This? (Probably all too well)*

My internship had been typical of many: Learning the ropes of the corporate world, learning that the process of creating software is very different than what is done in college and generally performing a number of small tasks (the stuff that the 'real' engineers didn't want to do). As simple as the internship had been, I knew that the company did cool stuff, and that I wanted to work on that cool stuff. The following year, I was so excited to begin my career that I didn't bother to take any time off before starting. I graduated on a Saturday and started my new job two days later. Looking back, it probably wouldn't have been a bad idea to take a few weeks off, but I was sick of Ramen Noodles and ready to start making real money.

Although I already knew a bit about the company, I wasn't fully prepared for what happened next. I was ready to hit the ground running. Contribute! Write lots and lots of code! The first week of work – the entire week – was spent in various orientation activities – meeting other new employees, watching videos, completing the required HR classes and so on. Boring, sure, but part of the requirements of working for a corporate giant. Many of us have been there.

This corporate giant had a traditional approach to engineering, one that came from a legacy of experience with electrical engineering and driven by the best perceived best approach at the time: The dreaded old Waterfall SDLC. Agile, RUP, Iterative Design – these were things that I had never heard of. Even my Computer Science teachers hadn't the slightest idea about the emerging approaches to software engineering. To be clear, I'm not trying to be critical here: Software engineering as a discipline was well established at the time, but the silver

bullet to good software design and development was still something of a constant question. (As much as we've all grown to love Agile and its variants, approaches will always continue to evolve. Let's hope so, anyway).

It appeared to me as though some people in the workplace had an old waterfall approach to their career: It starts with requirements gathering (figuring out what it is I want to do) and ending in maintenance (just keep doing this until I retire). To be fair, I know I'm oversimplifying it a bit the waterfall approach a bit. Even the most old fashioned approaches often iterative through the traditional waterfall. Bear with me...

Orientation complete, I was eager to get started – to be a *real* software engineer!

But that didn't happen. I was on a project that required *heavy* up front design, and significant learning (on my part) about digital signal processing and real-time embedded programming. A well-meaning manager, trying to give direction in the only way he knew how, provided me with piles of documents, I suppose with the expectation that I would learn by reading.

Read this. Done? Read this. Done? Read this... Repeat

All the documents looked the same, and it was difficult to keep my eyes open, much less comprehend the words. The more I sat and read, the more frustrated I became, and soon I began dreading work, and counting the hours to the end of the day. As far as I could tell, I was getting paid to do *nothing*. That summer I saw the movie Office Space (I'm proud to say that I'm one of the few who saw it in the theater), and I became even more distraught. That movie showed the dreaded side of the corporate world, and I left the movie theater feeling awful.

Had I chosen the wrong career? Would it always be this boring?

I could barely stand the thought of reading yet another 100 page document full of words and acronyms that meant nothing. The thought of spending the next 30 or 40 years like this was downright depressing.

There were a number of problems:

- I was a new programmer without the necessary background to get my hands dirty.
- My manager didn't know what to do with me.
- The project I was on was in the early stages of design.
- Although I was eager, I was clueless.
- The enthusiasm I once had was quickly fading away.

Design, at least there and then, meant constant reading, re-reading and review of hundreds of pages of extremely specific requirements. The SRS (Software Requirements Specification) was the master, and it had to be *perfect* before anyone considered writing a line of code. This phase of software could take many months – years even, the goal being the creation of perfect software by way of big (massive) up-front design. Looking back, maybe it worked for the corporation (although I think it did more to hold back). For all of its flaws, the traditional waterfall approach was the best anyone had come up with. The defense contractor had plenty of money to spend on very long design phases.

As I continued reading, bored and annoyed, and growing increasingly jaded, I wondered if I would remember how to dereference a pointer after so much time spent reading documents with no opportunity to write code. I knew one thing: If what I was doing was 'Software Engineering,' I didn't want to do it.

The real problem not as much with the employer as it was with me: I didn't know how to take charge of my position. I didn't even know it could be done! College hadn't taught me how to *be* a software engineer in the real world. I was following the lead of others – folks who were content to take things nice and slow, living out the maintenance phase of their careers.

I floundered

Again, I was new to this world. Sure, I got good grades in my CS classes. I enjoyed them. But applying that knowledge to my first steps into a career eluded me. My job, as I understood it, was to do whatever my boss told me to do – nothing more, nothing less. I wrongly suspected that doing more might, in some way, be an annoyance. Coupled with a boss who didn't have anything for me to do (at least nothing that a junior developer could jump into), it created an awkward, negative setting (and I'm not being dramatic when I say that it was depressing). The career I had once imagined to be fun and exciting as proving to be anything but. The best part of my day was lunch, when I would step outside for a nice long walk, thinking about what other careers I could pursue.

One day, while sitting at my desk reading more confusing documentation, I nodded off. That's right – I actually fell asleep at work! My head hit the desk and I felt like I was in high school all over again – bored, tired and perplexed. Falling asleep, ironically, served as a wake up call. This could not continue.

Doug to the Rescue

After several months of monotony, I worked up the courage to talk to my boss, and tell him of my desire to do something else. This was no small feat. My high-school writing teacher used to say, "Don't rattle my cage!" I didn't wish to rattle any cages. (Mrs. Hemminger, if you're reading this, you were full of wonderful quips!)

Ultimately, after that awkward situation, I think my boss may have been relieved – After all, he was giv-

en a young graduate as a new hire, but he didn't have sufficient tasks for that hire. It hadn't been his decision to hire me. The company simply extended an offer to an intern and then placed that former intern (additional headcount) somewhere, anywhere, when the time came. Any small employer would never hire an employee under such circumstances, but in the world of giant corporations, this happens.

Soon enough I was moved to another department. Although it was in the same building, this particular department had a number of people with a different mindset. It was a more vibrant group with much more enthusiasm. Even the lights in this new section of the building seemed brighter. Why was it different? I'm not sure. Perhaps it had to do with the employees in the group. Maybe it was because the project was newer and seemed more interesting. Whatever the reason, it was different.

One guy, Doug (who I thought was old at the time, but I realize now that he probably only in his 30s), was a key engineer, and he was eager to spend some time with me. I can only imagine that he saw something of himself in me – an eager engineer who just needed a little direction.

He took the time to share with me some of the things he was working on. He wasn't simply doing that which was asked... He did much more. He explored ideas, created prototypes and presented his findings. He was constantly busy, not because he wanted to impress management, but because he was a curious and energetic engineer. It was clear that he enjoyed what he was doing.

One conversation between Doug and I shifted my mindset. While I don't recall the actual words of the conversation, it went something like this:

"Doug, how do you get to do all of this cool stuff? I'd like to get involved with this."

"Matt," he said, "nobody is going to ask you to do it around here. Just go ahead and do it, and then you'll get to do more of it."

As I think about this now it seems so very obvious, but at the time it was not. Perhaps my mindset had much to do with the world in which I grew up. Most of the people in my small home of Auburn, Indiana were union factory workers earning an hourly wage that seemed substantial compared to my mother's $15,000 per-year income as a receptionist. I was told from an early age that the foundry was a great place to work, and the labor union looked after its employees. These people went to work for their shift, worked hard and went home. They did exactly what was asked of them. Nothing more. Nothing less. Hard workers by any measure, their jobs were not ones that required ongoing career growth.

Work, in my mind, was a place where you went and did whatever your boss tasked you with. Sure, I pursued a degree in computer science, but only because (from an early age) I knew that I really liked writing computer programs. I wanted to be a "computer programmer" when I grew up. But as far as turning that desire into a career, my outlook was limited. I wanted to work someplace where the boss asked me to write software – cool software – and other than the task given, I didn't understand that this was very different than working in a factory.

It wasn't until I met Doug that my eyes were opened to the fact that the direction of my career was in my hands, not my boss's. I learned something else: Doug was moving forward and contributing in big ways to the success of the department by doing more than what was asked. He was prototyping and experimenting, and using what he learned to guide the up-front design. He was doing Agile before any of us knew what Agile Software Design was. And while he was as smart as anyone else, I also learned that motivation is a much more powerful driver of success than smarts.

Doug had all the same tasks on his plate as any other engineer. He too had to sit through long peer reviews and read through miles of documentation. But he was never bored. It wasn't long before I got my hands on a TI DSP Prototyping kit. I wasn't presented with the kit one day out of the blue. I had to ask people about it, how to get one and how to get started. I began writing code – experimenting – learning what it all meant. And I tapped Doug on the shoulder for help often. He never seemed annoyed.

Soon the words on the documents started to become clear. The words on the pages meant something. I wrote a state machine in C. Then I wrote an AM modulator... Then an AM demodulator. I learned that the Fast Fourier Transform that my professors rambled on about actually had some sort of purpose (but don't ask me about it these days).

I pursued Computer Science in college because I was fascinated by programming. It was fun and amazing to write code and then see what it does – to discover what one can make happen, and build upon those discoveries. I wanted to go to college so that my job would be a job of interest. In college, however, one does not learn about the skills of being a part of the workplace. I didn't fully comprehend the difference between a 'job' and a 'career.' I certainly had no solid understanding of how to move a career in a desired direction.

Is software engineering always fun? Let's be honest: No. There are deadlines, and more often than not these deadlines are tight. There is documentation and (still) lots of reading. There are annoying bugs (its always a thrill to find one and squash it). There are personalities... Different personalities. People often think of software engineers and folks who stare into a computer screen with little or no interaction with others. Not so!

The ability to interact with people, to understand and communicate is imperative in this field (but I digress).

So yes, there are certainly real-world needs in software development that aren't always fun. There are parts that can be downright tedious, sure. But it is imperative to maintain that initial fascination that led to this career path in the first place. This may mean choosing your place of work wisely. It can be tricky, but I suggest against moving into a role for which you have all of the skills and experience required. (Likewise, I think its important for employers to recognize the need of employees to face a challenge. So many roles that I see posted seem to indicate that the hiring company wants a candidate with each and every skill listed. As for me, I don't wish to have a job with nothing to learn). Once the position is chosen, it means continuous improvement – the personal kind.

Integrity

Every single motivational book out there speaks of integrity. I am currently reading Becoming a Person of Influence, by John Maxwell and Jim Dorman. About 4 pages in the words, predictably, are the words, "Integrity is crucial for business and personal success." One need not be a successful businessman and author to write such words. But are these words true? I've certainly spoken with many people over the years who think not. We all have. We've all thought it! In post Sarbanes-Oxley world, it sometimes seems as though cynicism rules the day. I think it is the corruption that makes the news, and vast majority of workplace leadership is of great integrity. This is my personal experience. (If this isn't so, I'd rather not know).

Integrity is a two-way street, and it means that, while we should expect it of our management, we should act with it, showing those relying on our output assurance that we are not just doing our jobs, but doing our jobs well. Perhaps it makes more sense to say that all workers in all positions, those in leadership and those just starting out, fresh out of school, should perform with integrity. It sure sounds nice!

The thing about having integrity is, its generally very easy to maintain with just a bit of effort. I'm sure we can all think of scenarios where "doing the right thing" isn't necessarily an obvious choice. But most of the time its very easy. There is a real need for integrity, buzzword or not, in many situations as a software engineer, decisions small and large that can cause some sort of internal conflict. We are almost constantly asked for time estimates, project status and testing results from upper management. Do we paint a rosy picture, saying what we think they want to hear, or do we tell the truth? Will that ticket *really* take 12 hours to complete, or am I just padding it with a bit of 'just in case' time? Acting without integrity in this regard can be tempting, sure, but there is little question about which path is the *right* way. With regard to that nice, happy word, 'integrity,' the fact of the matter is that *knowing* right from wrong is easy. If knowing which way is the right way, *doing* should follow.

Explore

I realize that I am painting a rosy picture here, and some reading this may roll their eyes a bit. That said, acting without integrity, even if it means a small lie, can come back to sting. All the integrity talk really touches on a much more broad subject. The only point in bringing it up here is to note that, minimally, a software engineer should be motivated to stay on task for purposes of integrity. I say *minimally* because I think it can be very easy to stay motivated and highly productive in this career – and because, as people who pursued this career because of a love of writing software, it seems that it should be even more easy to perform very well as a software engineer.

We have jobs where we get to play (perform a hobby) most of the day. We get to put together a big, complicated puzzle. And we get to find new, interesting ways to put things together. We are, in a sense, *artisans* (and I hope the word doesn't sound melodramatic). How can this be anything but motivating?

We've all floundered. We've all had our off days, where we just cannot focus. So it goes. But, in general, it seems to me that focusing on something you love doing in the first place should not be an ongoing problem. Not with so much cool stuff to learn!

I guess I could sum this entire post up with this: *If you're bored at work, you're doing it wrong*!

Maybe Doug had to learn the same way that I did. Maybe Doug went through the same early struggles that I did (I never asked).

Your boss is *never* going to be upset when you learn new things and take initiative – on the contrary! You're boss will love you, and he or she will be thrilled to have a team member that makes the entire time look good. Remember, your boss is very busy too. Don't expect your boss to constantly check to make sure that you're challenged and learning. Additionally, many of us have bosses who are either non-technical or *previously*-technical. They have moved into roles that require attention elsewhere, outside of the details of software specifics. In this regard, whether your job title is software engineer, computer programmer, software developer, junior programmer, senior programmer, development specialist – whatever, we all need to be software architects. The term 'code monkey' should not apply to any individual who has been hired for his or her ability to evaluate a problem, determine a solution and implement. This requires constant interest and learning (and hopefully it is gusto, because then the motivation comes naturally).

We all know the conversation from the movie Office Space:

Peter: Our high school guidance counselor used to ask us what you would do if you had a million dollars and didn't have to work. And invariably, whatever you'd say, that was supposed to be your career. So if you wanted to fix old cars, then you're supposed to be an auto mechanic.

Samir: So what did you say?

Peter: I never had an answer. I guess that's why I'm working at Initech.

Michael: No, you're working at Initech 'cause that question is bull**** to begin with. If that quiz worked, there would be no janitors, because no one would clean shit up if they had a million dollars.

In the movie, Peter Gibbons was a struggling programmer working on the Y2K bug. His job was tedious. It never changed. There was nothing new or exciting to learn. I can't imagine anyone is such a role being very happy. But Office Space, as hilarious as the movie is, does not convey the reality of what software engineer *can* (and should) be. For a software engineer working on something that is fascinating and enjoyable, a 'case of the Monday's' need not apply. I'm not going to pretend that I look forward to coming to work each and every Monday, but I can honestly say that there are Sunday evenings, many of them, when I start thinking about what I'm going to work on tomorrow, and looking forward to it. That's a good feeling. Sometimes it feels like those times when I reserved time on the Apple IIe at the Auburn, Indiana public library so I could tinker and learn to program. Back then I did it for free.

Software engineers must explore – try new things – mess around with new stuff. I hope its obvious that this isn't to say that one should intentionally pursue extraneous distractions. There is a big difference between a distraction and learning. Facebook is great, but its a distraction. Learning Bootcamp, exploring Github, reading Stack Overflow, trying out the new features in Java 8 – these can be highly relevant, if not crucial, activities in the work of a software engineer. As far as integrity goes, contrary to my previous assertion, can an engineer of any type have the knowledge necessary to propose a solution without leading edge knowledge? This is advice to management as well: If you expect your engineers to offer the best solutions, they must be empowered to to explore.

This advice isn't for the sake of your boss – it is for you, the enthusiastic software engineer. Taking initiative is a sure way to prevent boredom and floundering – and an even better way to stay current in a world that is in a constant state of change. We're fortunate in this regard: Our jobs change and evolve. We are forced to challenge ourselves. Maybe "don't get bored" should be a part of our job description. There is a good chance that if the approach being taken on a project is boring, a simple repeat of something you have done in the past, there is a better way to go about it. Learn about those new things! Use your empowerment, the trust your boss has placed in you, wisely.

No matter how smart we think we are, how much experience we have or how much we *think* we know, those of us involved in software engineering have plenty to learn with regard to any task at hand, be it simple or complex. We are among the fortunate few who have careers that are directly tied to our hobby. People do this for fun – and we get paid for it! I can think of few attainable careers that offer such a benefit.

Rock star sounds like a great career, but it isn't attainable. Fiction writer sounds fun too – but again, there is little assurance of success. I may be biased, but I feel like most people have to go to a job that is boring by its very nature... I won't go naming any of those careers, but we can all think of things that our friends do that sound like an dreadful way to spend the day. We get to go to work and pursue something that we loved doing (hopefully) long before we were ever paid for it. Software engineering is unique. With a four or two year degree (in some cases with no degree), we can make a solid income doing something that is downright fun. A career is software engineering is not a waterfall. It is agile.

Also, if you're sick of Ramen Noodles, give it some time. *You'll grow to love them again.*

MATTHEW RUPERT

Matthew Rupert lives in Wake Forest, NC, and is a software architect with Ateb Inc., (Raleigh, NC). He has been designing and developing software for 15 years. Rupert graduated with a bachelor's degree in computer science from Ball State University. He has worked in a range of software development and lead roles ranging from defense contractors to healthcare. He blogs regularly on software subjects at http://matthewrupert.net. Rupert lives in Wake Forest, NC.