How to Fail at Agile Without Really Trying

Heather M. Wilcox

heatherwilc@yahoo.com

Abstract

Nobody likes to fail. We like to talk about it even less than we actually liked failing in the first place. However, if we don't talk about our mistakes, we don't learn from them. If we don't learn from them, then we are essentially guaranteeing that our failures will be repeated at some point in the future. The only thing that could be worse than failing the first time, is making the same mistake a second time for all the same reasons.

This paper examines the causes behind the failure of Agile at a Portland, Oregon software company. It presents anecdotal examples and observations of how and why problems occurred and, in doing so, provides a blueprint for a successful implementation of Agile.

Biography

Heather has spent the last 20 years working and learning in the software industry, choosing to focus primarily on start-up and small companies. As a result, Heather has had a broad range of job descriptions which include, but are not limited to: Tech Support Engineer, IS Manager, Technical Writer, QA Engineer, QA Manager, and Configuration Management Engineer. This has given her a wide range of experiences to draw from in her current role as a Senior Quality Assurance engineer.

1 Introduction

Nobody likes to fail. We like to talk about it even less than we actually liked failing in the first place. However, if we don't talk about our mistakes, we don't learn from them. If we don't learn from them, then we are essentially guaranteeing that our failures will be repeated at some point in the future. The only thing that could be worse than failing the first time, is making the same mistakes a second time for all the same reasons.

Interestingly, most success is rooted in failure. Very few scientists, inventors, or development teams get it right on the first try. However, when the "Great New Thing" is finally revealed, nobody talks about the bugs, the failed tests, the wasted time, or all the other things that went wrong on the way to the end result that was "Spectacularly Right".

This is the story of a company that failed at Agile and some of the more obvious causes behind that failure.

To be absolutely clear, the whole time period during which we were failing at Agile, we were also successfully releasing code. Our customers were satisfied with the product that we shipped and the rate of escaped (field) defects did not increase. In fact, rate of field defects may have actually been reduced due to the increased code scrutiny that is a benefit of the Agile process.

However, we also never received the full benefits that come with a successful Agile adoption. Releases were never easy and often were uncertain until the last moment. There were missed deadlines because of a misalignment of expectations between the parts of the organization that were waterfall and the parts that were Agile. This misalignment was often caused by terminology and methodology differences between the two approaches. There was also a lot of frustration within the teams because of the challenges that came with a partially implemented Agile strategy. Essentially, things could have been much easier had we adopted Agile and utilized Scrum fully and properly.

2 The Road to Failure

2.1 Lack of Training

The road to failure is truly the path of "Not Really Trying". Our adventure started with the initial decision that only the product development team would adopt Agile and Scrum. The rest of the organization (Including Sales, most of Marketing, Support, etc.) was made aware that Development and QA would be making the transition, but only the two affected departments and the designated product owners (Marketing Business Analysts and Product Managers) received any training. The rest of the organization was essentially left to its own devices to research and learn about Agile (or not). It was decided that the POs would manage the relationship between the Scrum Teams and the rest of the organization. Essentially, they would act as translators between the Agile and Waterfall worlds.

Additionally, not all the members of the Product Development team received Agile and Scrum training. The Systems Engineering (SE)/Delivery side of the department, who are responsible for maintaining the build systems as well as the test, staging, and production environments did not receive training. So, although some of the members of the SE team knew about and understood Agile and were familiar with Continuous Integration (CI) and Continuous Development (CD), not all of them knew or understood what CI/CD was about, how to implement it, or even how to interact with Agile teams.

2.2 Lack of On-Going Support

Once the designated Scrum Teams and Scrum Masters were trained, they were turned loose with a set of deliverables that were to be produced: A visible burn-down chart for each team, a visible task board with stickies for stories and tasks, and a formal demo at the end of every Sprint. Additionally, all teams were expected to hold daily Scrum meetings, a retrospective at the end of the Sprint, and a planning meeting

(that included planning poker) at the beginning of each Sprint. Extra points were awarded for backlog grooming meetings.

For about six weeks, we received ongoing support: Real time help from an onsite Agile consultant as well as additional trainings on running retrospectives and writing appropriate user stories. We were so Agile, it almost hurt. We had amazing momentum and it was good. However, once the consultant left us, there wasn't really anyone with the necessary authority or desire to drive back the "rot", so the process started to degrade almost immediately.

2.3 Lack of Belief in Agile

When the teams were originally assembled, we ended up with a couple of groups that were improperly constructed. Either the Scrum Master didn't believe in Agile or the majority of the team did not. For those teams, everything was a struggle because there were deep doubts within the team about the efficacy of the process. They "needed more training" (they could never have enough), the Sprints were too short (they were trying to do waterfall inside of two weeks Sprints – that's difficult!), and the Scrum Masters who actually believed in the process couldn't be effective because they were constantly fighting the majority of the team who didn't want to do Scrum in the first place.

2.4 Team Stability

Team stability (or lack thereof) was a huge contributor to our failure. For a variety of reasons, the membership of the teams has been significantly re-adjusted about once every 6 months or so since our initial implementation of Agile. Moving team members around in this way has had the unfortunate side effect of forcing a "reset" on team cohesiveness and output. It's commonly accepted that it takes at least two or three Sprints for a team to gel and really become productive after formation. It takes additional Sprints for teams to understand their velocity and begin to run smoothly. This means that we've lost approximately two months of intense productivity out of every 6 month period.

2.5 Timing and Communication Problems

In the case of this company, the final stake in the heart of Agile (and Scrum) was time. As previously explained, most of the organization did not adopt Agile. Consequently, external commitments continued to be made to customers in "waterfall" style, meaning that clients were promised that features would be delivered on a certain date without first checking with the teams. This is a very "top down" approach which is inconsistent with the Agile "bottom up" method of estimating and providing features.

The end result of this process was that teams were constantly behind and having to play catch up to get their work done in time to meet the commitments that were made for them. Given that situation, Scrum rituals were the first to go to make time to simply "get stuff done." For instance, many teams stopped tracking velocity. Since they were given "drop dead" dates for the things they were working on, team velocity became irrelevant. Things had to get done on time regardless of how long the work would actually take, so it made no sense to waste valuable work time on estimating velocity. Additionally, some teams reduced their Scrum meetings to just once a week, or quit holding them altogether. Time became such a valuable resource, that it was hard to justify the expenditure of time on Scrum when much of the necessary information could be communicated via an email update.

3 The Signs of Failure

Some of the more obvious signs that are visible on the Road to Failure have already been described, however, there are always more than just the basic indicators. Some of signs are a bit more complex or obscure and you need to look a little harder to see them so that you don't get lost. In this case, these less obvious signs took the form of aberrant team behaviors.

3.1 WaterScrum or Scrumfall

As mentioned previously, some teams made serious attempts to run waterfall-style development inside of their Sprints. We referred to this behavior as either “WaterScrum” or “Scrumfall”. A major complaint from these teams was that a 2 week Sprint was far too short. They wanted at least 3 week Sprints with 4 weeks being ideal. Instead of breaking an epic-sized feature into smaller stories and then working on the stories in a sensible order, the hybrid teams tried to do all of the planning for the epic before starting any work. They also had a hard time trying to figure out where to store their requirements because they felt that User Stories didn’t have any place for them. This confusion around requirements arose because they didn’t understand the concept of Acceptance Criteria. Even the task stickies for these teams were waterfall – tasks were documented in painful detail, including (but not limited to) how many hours were actually required to complete a task vs. how many hours the team thought it would take.

Planning for the Scrumfall teams always took hours and sometimes took more than a day, which is a lot of time to spend planning a two week Sprint! In true Waterfall fashion, these teams would assemble every last requirement and piece of information before moving forward on any actual development work. QA for stories often happened in the next Sprint, since the entire feature had to be fully completed before it was passed to QA. Their stories also had a tendency to be too big – one story often took more than a week of development time.

3.2 Top-Down Estimation

Again, as previously mentioned, a large part of the organization did not convert to Agile. As a result, release schedules for projects tended to be dictated from the top-down (Waterfall) instead of bottom-up (Agile). Although most of the scheduling was not unreasonable, chosen feature release dates tended not to take Agile process time into account. It quickly became clear to many of the teams that there was no value in estimating or calculating velocity - the features needed to be done in time for the designated release date and that was that. So taking the time to play planning poker and estimate team velocity made no sense.

Additionally, in order to stay on target for the pre-determined release dates, teams threw out any process that they felt served no direct purpose or slowed them down. In-depth retrospectives were abandoned along with some Scrum meetings, backlog grooming, and other Agile rituals.

3.3 Feature Creep and Schedule Slippage

The top-down scheduling also resulted in a top-down approach to feature inclusion. Additional features were tacked on to releases to suit customers without first checking on how that might affect the team. If a schedule slipped for other reasons, it was taken as an opportunity to add additional features. The reasoning behind this behavior was that, “since time was added to the schedule, there must be more time available for coding, right?” However, that wasn’t really the case since the schedule was slipping in response to the need to finish work that was already committed. But again, since there wasn’t a real feedback loop to the teams, the feature would be promised to the customer and added to the release. The end result of this was often an additional (but smaller) addition to the schedule on top of the original modification.

3.4 Allowing the Wrong People to Drive

The longer we allowed Agile to disintegrate, the more dysfunctional things became. One of the more ineffective choices that was eventually made was to have each Development Manager act as Product Owner for multiple Scrum teams. There were several things wrong with this scenario – many of which became clear almost immediately.

The first obvious mistake we made, was allowing someone to act as Product Owner for more than one team. Being a good PO requires a big time commitment - not only to managing stories, but also researching product requirements and customer needs. Spreading someone across two or more teams

meant that they were unable to do the Product Owner job well for any team. If one group was getting the time they needed, the other team(s) were getting completely neglected. So a group would have help periodically and then be on their own for a while. This created problems with keeping team backlogs in good shape. Stories tended to be inconsistent, poorly written, barely fleshed out, or not fleshed out at all. Teams also sometimes resorted to writing ALL their own stories and maintaining their own backlogs without PO input.

The second mistake was allowing the manager of development engineers act as their Product Owner. That fast became a case of the wolf guarding the hen house. As Product Owners, the Engineering Managers were being pressured to get product out. Because the POs were also the managers, the Engineers had nowhere to go when they felt like the project timelines were untenable.

The last, and possibly largest mistake, was having someone who had no marketing or customer knowledge drive the creation and prioritization of stories. Development Managers are good at development but aren't in tune to customer needs, since that's not their job. They had no clue what the product should look like or what the priority of features should be. The Development Managers also didn't understand how to turn technical requirements into stories that delivered business value. Teams floundered while the PO/Dev Managers acted as intermediaries between Product Management and the Scrum teams. Eventually, many teams ended up "firing" their POs and started getting their stories directly from Business Analysts or Product Managers.

3.5 Re-Inventing the Wheel

As the teams wandered further from Agile and good Scrum process, the Managers, in an attempt to "solve the problems" ended up trying to re-invent the wheel. They held meetings to discuss issues like "What the Job Description of the Product Owner should be" and "How Agile should work in the teams". The crazy part of these discussions is that these things are already very clearly defined. Unfortunately, the foundation of Agile had been lost, so the idea of going back and re-examining the actual defined roles and responsibilities to try and figure out where things went wrong didn't occur to anyone. The effort to "Fix the Process" became a twisted internal dialog that went something like: "Well, what we have isn't working right, so Agile must be broken. Maybe if we re-define the principles of Agile, it will work better." However, the crux of the issue was that the Agile implementation itself was broken, which is why the roles weren't working the way they were supposed to. If Agile was re-implemented properly, the issues surrounding the role of Product owner and Agile process itself would likely resolve themselves.

3.6 Fractured Scrum Teams

While the scheduling and feature creep problems continued to build on themselves, it became necessary to steal people from teams working on "future" projects to help make up the deficits in the "now" projects. Eventually, the endangered products were completed on their adjusted schedules. However the cost was pretty steep in that new development projects, which were needed sooner rather than later, were completely halted. The effects of this are still reverberating as the re-assembled "future" project teams are now trying to catch up on all the work that didn't get done while they were helping the "now" teams. Once again, schedules didn't shift to adjust to the lost work time, so the catch-up cycle has started again, with the "future" teams robbing people from the "now" teams so that they can meet the schedules that they've been committed to.

4 The Road to Recovery

Even with everything that's been described to this point, the one advantage of failure is always the opportunity to use it as a blueprint for success. You've already done it the Wrong Way and you intimately know that path. Simply not repeating those mistakes should get you moving in the right direction. Taking the time to really learn from and improve on the things that went wrong should dramatically increase your chances of "getting it right" the second time around.

Within our organization, we've identified several things that we could do that should fix many of the issues described and also generally improve the functionality of the Scrum teams.

4.1 Full Time Evangelist

It was clear that, as soon as our full time Agile consultant left, things began to disintegrate. Having an on-site Evangelist is critical. This person should exist only to regulate and reinforce the process, make sure the teams are "doing it right", ensure cross-team consistency, be vigilant, interact with management to set expectations, and teach new hires as well as those in the company who have not been trained in Agile process and Scrum.

The primary goal of this position should be to audit processes for all the Scrum teams. This would be done by periodically checking on the groups and verifying that they aren't wandering too far off the path or trying to "WaterScrum" or "Scrumfall". Although teams need to be able to adapt processes into something that works for them, everyone needs to be going about Agile the same general way. This keeps things consistent internally and makes it easier to regulate releases and Agile process. It also eases the transition for anyone moving between teams.

Additionally, support and consultation are a huge part of the role of the Evangelist. The person who fills this position needs to be available to help teams troubleshoot Agile and Scrum questions and mentor them through the transition. Not only does this person audit the teams and regulate process, but they also help guide the teams back to the right path. Any thought that this role is, in any way, punitive is entirely incorrect. The Evangelist is more of a "spirit guide" if you will, leading teams down the righteous path to Agile salvation.

If at all possible, the Evangelist should be chosen from outside the company. Ideally a new-hire or a consultant. When the position is an internal hire, they often come with a whole set of biases about projects and people that may influence how different teams are treated. If you must choose your Evangelist from within the company, said person should be as neutral as possible and also completely passionate about Agile and Scrum. You definitely want to choose someone that can fire up your teams and not only get them excited about Agile but keep them interested for the long haul.

4.2 "Weed the Garden"

As part of the work of reinforcing Agile, it is important to remove "non-believers" from authority positions on Scrum teams. They most certainly cannot be Scrum Masters or Product Owners. "Non-believers" should be broken up and distributed across the organization so that there isn't more than one or maybe two on a single Scrum team. If they are maintained at minority levels on teams made up of "true believers", then their potential for damaging the process can be minimized and eventually they may even convert. However, it is clear that they cannot be allowed to congregate and build a "Scrumfall" team.

4.3 "Seed the Garden"

Just as you don't want too many "non-believers" on a team, you can use "Agile Ninjas" or "True Believers" to bolster teams that are having trouble. Sometimes adding a person who really understands the process can help a team find their way out of the weeds. If it doesn't impact momentum, subbing a Ninja in as either a temporary or more permanent Scrum Master may really speed up the journey to team enlightenment. But, even if you can't get your True Believer into the SM role, just getting them on the team should be a big help. Our most successful teams had either an "Agile Ninja" or "True Believer" as their Scrum Master.

4.4 Spread the Fun Around!

Everyone on a Scrum team should have a chance to be the Scrum Master, even if it is just for one Sprint. Taking a turn as the Scrum Master gives people the opportunity to experience the job for themselves and develop some empathy for the people who do it regularly. Being a Scrum Master is often not as easy as

it looks and doing the job allows the other team members to find that out first hand. Ideally, this will make team members more receptive to the Scrum Master (or the Agile Evangelist) and more likely to acquiesce to their requests.

4.5 Re-Commit to Agile

To really make Agile successful, the whole process needs to be restarted and must involve the entire organization at appropriate levels. At the very least, there should be company-wide training (or a series of trainings) on the Agile methodology and the Scrum process. This will assist the rest of the company in their interactions with the Scrum teams and will help them to set expectations appropriately. It will also free the teams to actually work correctly within the Agile framework instead of having to spend precious time fending off inappropriate requests from those that don't know any better. Additionally, the Scrum teams should go through the process of re-chartering, so that they can re-calibrate and restart their internal processes and get themselves back on the right path. Finally, there must be an increased focus on tracking velocity, team output, and quality of output. These things are amongst the primary "rewards" of adopting Agile and, as such, should be watched closely and nurtured.

4.6 Focus on Consistency

Once the Scrum process is restarted, it is imperative that every effort be made to keep things consistent and the environment "distraction-free". Management rule breaking (such as temporarily re-directing team members to other projects or interrupting Sprint work for special projects) definitely should not be tolerated. If a problem arises that has special circumstances, figure out solutions that solve the issue within the existing Scrum teams or form a new team, but don't disrupt team membership or focus to solve a one-time problem. One possibility would be to form a "SWAT" team that's dedicated to dealing with real time issues.

4.7 Build Teams that Make Sense

When initially assembling Scrum teams, try to choose workers with complementary skillsets and (if possible) personalities. This process is described in more depth in Bhushan Gupta's paper, Waterfall to Agile: Flipping the Switch (Gupta, 2012). In addition to having appropriate personalities and skillsets on a team, good communication is imperative. Placing a person or two in the group that has excellent communication or facilitation skills is always a great idea.

Building a good team is not an accidental process. Sticking folks together and just "hoping it all works" doesn't always turn out the way you want it to. If you start with a concerted attempt to create a strong group, it increases your chances of success. Ideally, you want to build teams that will bond fast, reinforce each other, communicate well, and work hard together.

4.8 Keep Your Teams Together, unless….

Once your Scrum teams are assembled and working well together, leave them alone. Don't change membership unless it is necessary. As mentioned previously, making large changes (e.g. swapping 2 or 3 people from a team of 8) to your teams can really de-stabilize them and cause a loss of velocity and momentum. Additionally, it may take two or three Sprints before the reconstituted team is able to integrate the new membership and return to its former productivity levels. That said, people can get bored or feel trapped if all they ever do (or are expected to do) is work on the same thing with the same people. Be prepared to occasionally shift ONE worker off of a team and into a different group. In most cases, swapping out one person doesn't cause a lot of angst and a single new worker can be brought up to speed pretty quickly.

4.9 No Oddballs!

Don't allow "oddballs" onto your Scrum teams. It's easy to feel like everyone in the department should be assigned to a team, but that isn't necessarily the case. "Oddballs" are people with specialized skillsets or talents that aren't needed full time on a particular team (e.g. a UI design specialist.). Having these folks

on a team full time can be a distraction and a frustration to the rest of the group. "Oddballs" don't tend to be equipped to contribute appropriately to the team, so they often have "no status" to report, or they get loaned out to other groups so the status they report has nothing to do with their team's activities.

If you have "Oddballs", put them in a "rental pool" so that they can be "checked out" by teams when needed and then returned to the pool when their specialized work is complete. This allows all groups to have access to these unique resources without the burden of integrating them full time when they don't need them. It also helps out the specialized workers in that they can do the tasks they've trained to do without having to worry about being forced to do work that they're not qualified for or interested in doing. Finally, it avoids the worst case scenario of having highly paid, highly specialized employees sitting around doing nothing while waiting for an appropriate task to come along.

5 Thoughts from the Road

While doing research, it became clear that the problems we experienced were not unique. A lot of companies go through similar pains. It seems like the most successful organizations were the ones that kept trying, did a lot of thoughtful retrospection, and were finally able to internalize the tenets of Agile. There were interesting parallels between this company and other groups that are described in the online materials cited below.

Specifically, along with everything that has been discussed thus far, it is clear that we made pretty much all of the mistakes with all the resulting problems described in Martin Fowler's article, "Flaccid Scrum" (Fowler, 2009). We didn't pay attention to applying "strong technical processes". We did spend time talking about technical process, but we didn't make any real decisions and we definitely didn't implement processes in any consistent way. The tendency was to adopt a process, which some number of teams (but not all) would take on and follow until some newer and more interesting technology came along. For instance, during the first two years of Agile implementation, we went through at least 3 different configuration management tools which were implemented, adopted, and then abandoned at random points in the teams' projects. That's crazy!

As described in the article, the end result of behaviors is that it became more difficult for us to add features, because our code bases were too tangled up with each other. We built up a huge Technical Debt – to the point that we ended up (for a while) with a policy that 20% of all Sprint work would be aimed specifically at tackling it. And, as predicted, our Scrum went "weak at the knees". We officially became the poster children for "Flaccid Scrum".

Digging deep into the source of all the issues we've seen, it has also become clear that per Robert Martin's blog article "The *True* Corruption of Agile" (Martin, 2014) we failed to do a thorough job of adopting the real culture and practices of Agile. Initially at least, we took on the practices of Agile and Scrum and followed them religiously, but the entire group didn't take on the Agile culture. Even though some of the individual teams did culturally adopt Agile, the entire group did not, which made it very difficult for the truly Agile teams to work with the fake Agile teams.

Without a real and solid Agile culture to operate within, the teams that believed in the philosophy and process were forced to make compromises to meet non-Agile expectations. There were a lot of conversations that started with "I know this isn't the right way to do this but…." This single sentence fragment was the leading edge of what Robert Martin believes is the biggest problem within Agile – the "*elimination of practices*". A perfect example of this is the teams that quit tracking velocity because it didn't make sense. Top down scheduling, which is very much a non-Agile behavior, made velocity irrelevant. In a nutshell, the lack of adoption of a true Agile culture led to the rejection of the Agile processes and Scrum.

6 Where the Rubber Finally Meets the Road

One of the really nice things about the particular software company in question is that open-mindedness is encouraged and anyone with a good idea is welcome to put that idea out for discussion and implementation. That is how we originally came to implement Agile in the first place; Waterfall processes weren't working well for us and a couple of small teams had individually implemented it with reasonable success. It seemed like implementing Agile across the entire Engineering team might get us some pretty significant gains in productivity and quality. And, as I mentioned earlier, we have seen real benefit from making the switch to Agile, so the effort was not entirely wasted.

Happily, there has also been some recognition that things haven't worked as well as we'd hoped and we've started slowly working on fixes to some of the more obvious problems that have been described. There has been some "selective weeding" on the scrum teams to move some of the less amenable people into places where they aren't as harmful. We're also working on implementing a single CSM tool for the entire team to use. Once this step is complete, we will begin making the transition to CI/CD in earnest. As part of these efforts, QA is working closely with their development counterparts to implement smart and reasonable automation to complement the application code that is being developed.

Thus, change is definitely in the works. One of the reasons that this particular paper was internally approved was so that it could be used as an actual blueprint for improvement. As such, the content of this document is not theoretical, it is part of a greater plan to re-implement Agile process and Scrum properly and get things moving again in the right direction. With luck, there will be a presentation at the 2015 PNSQC detailing "How It All Worked Out."

7 Conclusions

Once you've learned all the mechanics, put good processes in place, and constructed strong, functional teams, developing within the framework of Agile is easy. However, not lapsing into old habits, especially within a resistive environment, is much more difficult. Constant vigilance is a necessity. More importantly, it's vital that your organization makes a top-down commitment to Agile and Scrum. Even if the entire company doesn't implement it, there must be a commitment to learn about the process, what to expect from it, and how to interact with it.

Without top-down Organizational buy-in, your team will constantly fight misaligned expectations, misunderstandings, and general issues that are the result of trying to align two methodologies that don't understand each other.

Within your Agile implementation, strive for simplicity and for processes that make sense. When a procedure becomes onerous, nobody wants to follow it or keep up with it. At the same time, don't try to re-invent the wheel. The processes are what they are for a reason. They work and they work well. Redefining roles and procedures only confuses things and hides problems. If things aren't working the way they are designed to work, there's a problem that needs to be addressed.

Ultimately, Agile isn't difficult, but it does require that your organization make the effort. As with all other development methodologies, if you don't try to do it right, you will fail. It's that simple.

8 Acknowledgements

My sincerest thanks go out to the following people, without whom, there is no way this paper would have been written: My husband Justin who, even though he doesn't understand Agile or really even technology, gracefully puts up with my babbling about it and putting gigantic sticky notes all over the house. Scott Richardson, who provided valuable feedback just when I needed it. Aaron Getz who pointed me at the articles I cited in this paper. My manager, Roger, who let me write this paper when I really should have been doing actual work. My PNSQC editors, Karyn and Bhushan, who gave me

valuable guidance and who taught me that Scrum and Agile aren't the same thing (DUH!). And, finally, my Scrum teams – you know who you are. Without all of you, I wouldn't have learned what I learned and I would most definitely not have been able to write this paper.

A final and special thanks goes out to the company that I work for. They stood behind me and supported me in this endeavor, even though I cannot reveal their identity. Thanks!

References

Fowler, Martin. 2009. "FlaccidScrum" Martin Fowler Website, article posted January 29, 2009, http://martinfowler.com/bliki/FlaccidScrum.html (accessed April 25, 2014).

Gupta, Bhushan. 2012. "Waterfall to Agile: Flipping the Switch". Proceedings from the 2012 Pacific Northwest Software Quality Conference (PNSQC) http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf (accessed August 13, 2014)

Martin, Robert. 2014. "The *True* Corruption of Agile" The 8thlight Blog, entry posted March 28, 2014, http://blog.8thlight.com/uncle-bob/2014/03/28/The-Corruption-of-Agile.html (accessed April 25, 2014).