What Killed Waterfall could Kill Agile.

Robert C. Martin
20 Nov, 2010

In 1970 a software engineer named Dr. Winston W. Royce wrote a seminal paper entitled *Managing the Development of Large Software Systems*. This paper described the software process that Royce felt was appropriate for large-scale systems. As a designer for the Aerospace industry, he was uniquely qualified.

He began the paper by setting up a straw-man process to knock down. He described this naïve process as "grandiose". He depicted it with a simple diagram on an early page of his paper. Then the paper methodically tears this "grandiose" process apart. In the end, Royce proposed a far more nuanced and insightful approach, leaving the reader to giggle at the silliness of the "grandiose" model.

Royce's paper was an instant hit. It was cited in many other papers, including several very important process documents in the early '70s. One of the most influential of these was DOD2167, the document that described the software development process for the American Department of Defense. Royce was acclaimed, and became known as the *father* of the DOD process.

There was just one problem. The process that DOD2167 adopted was Royce's straw man! Apparently the authors of DOD2167 did not actually read Royce's paper; because they adopted the "grandiose", naïve process that Royce's paper had derided. To his great chagrin, Dr. Winston W. Royce became known as the father of the waterfall.

Though Royce railed and fought against it, the snowball was in motion. It kept on growing as it rolled down the mountains of software companies and industrial countries. Year by year the waterfall gained in popularity leaving it's father to wonder about the justice of the universe and whether there was intelligent life on Earth.

By the middle of the 1990s, the waterfall process dominated the world of software. The field of Software Engineering was *defined* by it; and by the catalog of analysis and design documents that Architects, Designers, and Analysts were expected to produce. Coding was a detail – the least important part of the process. If you wrote your documents well, and drew all the necessary diagrams, then you were doing it right. You were an *engineer*. The code could be left to the to the unwashed minions in the cellar.

This attitude created a schism in the technical community. There were the *elite* Architects, Designers, and System Analysts who did the *real* engineering by satisfying the first two phases of the waterfall. And then there were the grunts who actually had to make everything work in the final phase. When the project got

behind schedule, it was the grunts who worked overtime. When the project failed, it was the grunts who bore the blame.

This was a great deal for the elite Architects, Designers, and Analysts! Who wouldn't want a job like that? You have the authority to specify everything, and none of the responsibility to actually make it work. You get to command a high-salary, the respect of your peers, and the envy of the masses; and there's almost no way you can fail. When bad things happen, you can always blame it on the programmers.

Yes, this is a bit of an exaggeration; but only just a bit. The attitudes of elitism were very real. Those who could analyze and design were considered too valuable to waste on mere coding. Code became the orphaned child of Software Engineering.

By 1998 the cracks were already appearing in the waterfall edifice. Programmers everywhere were starting to reject the elitism of the Architects, Analysts, and Designers. They started complaining about the rigidity and weight of the waterfall mantle they were forced to wear. Beedle, Devos, Schwaber, and Sutherland had published their seminal paper on Scrum, and Kent Beck had created a movement around eXtreme Programming (XP).

Scrum broke the waterfall apart. Rather than spending months or years creating reams of documents through a series of sequential phases, Scrum suggested that a team of developers should work in short 30-day[1] cycles to *implement features*. Scrum suggested that the development team should decide when and if a document was necessary. Scrum took the emphasis away from documents and artifacts, and put it squarely on getting features working, and on the decision making power of the team. Scrum broke the monopoly on authority held by the elites and put it in the hands of the development team.

XP took this a step farther by increasing the emphasis on the act of programming, and by declaring *code* to be important. XP is the integration of Scrum with a set of engineering disciplines. Those engineering disciplines have a huge effect!

Scrum is a day-by-day process. It provides a framework that describes what your day will be like, but it says nothing at all about how you should work in the hours and minutes of that day. XP is a minute-by-minute process. The engineering disciplines of XP fill your day. XP provides guidance for the creation of each line of code. It provides a framework within which coding and design decision can be made.

Scrum is a subjective process: The *team* rules. There is no objective measure of success, or of quality, or even of completion. It is up to the team to define these things. The engineering disciplines of XP add some objective measures to Scrum. XP defines design and code quality, and provides guidance on how to achieve it. XP defines the meaning of *done*, and how done-ness can be measured.

---

[1] Nowadays two weeks is more common than 30 days. Scrum and XP teams have found that too much can go wrong in a month.

By 2001 the software community was a-buzz with these revolutionary ideas. The Agile Manifesto had been written, and had become the centerpiece behind an energetic, enthusiastic, and growing movement. The very definition of Software Engineering was being challenged, and that challenge was succeeding.

The elitism engendered by Waterfall was being attacked. Neither Scrum nor XP had any role for an elitist who assumed authority without taking responsibility. In Scrum, the rule of the team is stronger than the rule of an Architect or Designer. Contributions are welcome, but not necessarily followed.

XP took this even farther. If you wanted to contribute to an XP team, you were welcome; but you took explicit responsibility for your contributions. For Architects and Designers that meant they wrote some code. For Analysts, that meant they wrote tests. *Everyone* on the team had the responsibility to make things work.

For a while it looked like Software Engineering elitism was dying; that authority and responsibility had been aligned once and for all. But elitism is a tough thing to kill. Whenever you rob Peter to pay Paul, you can be sure that Paul's outrage will be less than Peter's.

Both XP and Scrum defined the role of a coach. It is the coach's responsibility to defend the process. The coach reminds everyone of their commitment to their disciplines and to the process. When the schedule looms, and customers are angry, it is the coach who reminds the team that the best way to meet the schedule and calm the managers is to *hold to their disciplines*. In Scrum, this role was called "Scrum Master".

XP defined the role of coach quite informally. The role would float between members of the team. One month it would be Joe, the next it would be Jane. It was not a title, and it conferred no authority. There were no decisions to be made, and no power of enforcement granted. The coach had the responsibility to *remind*, not to command.

In Scrum, something different happened…

The very first Certified Scrum Master course was taught at the Object Mentor offices in Vernon Hills, Illinois. Ken Schwaber called me up and asked if I had a training room he could use. I told him I would be more than pleased to host his course. He kindly allowed many of the people who worked for me at the time to attend for free, and to become CSMs.

Frankly, I thought the idea was a bit silly. I didn't think thousands of people would be lining up to get their certifications. But I had not considered the lure of elitism. It didn't occur to me that this special training course, coupled to the term *Certified* Scrum Master, would become a wedge to break the alignment between authority and responsibility.

Who was it who lined up to take the CSM courses? Was it Scrum team members who wanted to help their teams? Was it programmers and testers? Yes, there were certainly some CSMs who came from existing teams. But the vast majority of CSMs have a project management background. In essence they have added CSM to the

PMBOK. They have become CSMs so that they have the authority to *manage* Scrum teams.

This was never the intent. The role of the coach was to act as a gentle reminder of process and discipline. The coach was never supposed to manage the project or the schedule! Indeed, these two roles were supposed to be adversarial! It is the project manager's role to remind the team about the schedule and to encourage them to change something so that the schedule can be met. It is the coach's role to remind the team to hold to the process.

True XP coaches are not project managers nor are they team leaders. They do not lead the team to success, and cannot claim credit for that success. Indeed, the role is considered optional because mature teams will probably not need frequent reminders. But CSMs often assume the role of team leader. They are viewed as a critical component of the team; without whom the team cannot function. In XP a team without a coach is no big deal; but a scrum team without a scrum master is an oxymoron.

Indeed, the role of Scrum Master is considered so important, that it requires *certification* to obtain. If your Scrum team does not have a *Certified* Scrum Master, then something must be wrong with you.

When a Scrum team succeeds, it is the CSM who steps forward to receive the award (on behalf of the team, of course). But what happens when a Scrum team fails? Is it the CSM who steps forward and falls on his sword? Does the CSM take the lashes and protect the team?

The elitism is back, and it's growing. More courses with certifications are available, and even more are envisioned. Other training companies are offering their own certifications. After all, the lure of elitism is a great moneymaker. The snowball is rolling down the mountain, and getting bigger with each turn.

And when the revolution comes… ?

I can only hope that when Scrum goes down it doesn't take the whole Agile movement with it.