Waterfall to Lean, with Agile In Between

Fri, 05/24/2013 - 16:38 — joedanmason

Waterfall to Lean, With Agile In Between

A Journey in Software Project Management

Software Management is Devilishly Hard

"*The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. (This very tractability has its own problems.)*" *- Fred Brooks*

"*In a room full of top software designers, if two agree on the same thing, that's a majority.*" *- Bill Curtis*

Software Project Management (PM) remains devilishly difficult, despite several decades of industrial software development and the emergence of new techniques like Agile-Scrum and Lean-Kanban. Waterfall-style project execution, the mother of all PM techniques, has taken a persistent beating from some camps in the software world who believe that Waterfall is ineffective, but some folks (like me) believe that Waterfall will always be effective for certain classes of project. In any case, the success of a PM method is dependent on the culture, people, and type of project - in other words, the *context,* which is bad news for those who imagine reducing software development to a predictable, standardized work process. At this stage in the evolution of software methods, each PM style offers useful tools and practices, as well as cultural principles, for the savvy development team to choose from. No methodology, new or old, works best in all cases. There is no Einsteininan "unified field theory" that handles all cases, all the time.

To demonstrate the current state of software development management, here are a few references:

Software Project Failures Hit 5-Year High [1]

Why Projects Fail [2]

Software Project Failures Cost Billions [3]

The State of Software Quality 2012 [4]

In this blog series, I attempt to chart a path of discovery over the past decade or two, starting with my experiences with Waterfall, followed by the game-changing principles of Agile, and finally the promise of Lean methods to handle software projects as part of an overall culture of Lean.

In **Part I** of this saga, we address ***Waterfall Project Management.***

PART I: Fun With Waterfall

A Case of "Engineering Blackmail"

When I was working as a development manager back in the 1990's, my boss once accused me of committing "engineering blackmail". He was a non-technical guy with an unpredictable temper, and prone to lashing out when frustrated, and I was at Ground Zero of a major eruption. Why is it "taking so damn long" and why is it that "you guys can never meet commitments". Youch! I had committed a career-limiting sin by introducing "reality" to the conversation.

I had never heard the term "engineering blackmail", and have not since. I think he meant I was holding the code hostage until he agreed to give us more time, which was, I guess, sort of true. But there was an underlying implication that I was lazy, incompetent, or diabolical, and would risk the success of the project to give myself more leisure time. In reality, the last thing in the world I wanted was an altercation with this guy. The second-to-last thing I wanted to admit was I simply didn't know how long it would take to complete the software. The plan had gone awry, though, and someone had to be held accountable.

This is a typical scenario in software development, especially when Waterfall Project Management is used. Looking back on this incident years later, I understand his frustration. After all, we had failed to meet an agreed-upon schedule, which was neatly documented right there in a black-and-white Gantt chart. It had been months since our last official software release, and he had nothing to show for his investment in our project; now we were asking for more resources (time) so that he could get something *he'd already paid for*, in his way of thinking. To him, it sure felt like we were holding the code hostage. Of course, we didn't really *have* the code, because it wasn't cooked yet, despite what the plan said.

What is "Waterfall" Anyhow?

Waterfall project management is a form of Defined Process Control, which requires that every piece of work is well understood functionally and clearly defined in terms of time, effort and sequence. Each step in the process is defined and estimated, resources determined, dependencies identified, and the sequence of activities established. Waterfall plans may include "phase gates" or "milestones" for checking progress.

All this information (and much more - risk identification, stakeholder identification, supplier interfaces, change management processes, and so on) is distilled into the form of a Project Plan. This planning

process is typically owned by Project Managers, who gather input from many individuals, craft a plan, and get agreement that the Plan is a good one. Then, Project Managers attempt to make the plan come true by shepherding people, rearranging tasks, manipulating resources, and tweaking the other "controls" at their disposal.

The key part of the project plan is the schedule. In theory the schedule is made from good estimations, which are based on well-defined software features and functions. However, lots of pressure weighs on project managers, engineers, and anyone else responsible for estimation, to make the schedule as short as possible.

Figure 1 - A simplistic representation of a Waterfall plan. As shown, major activity groups are executed in sequence. Some overlap is allowed, but the essence of Waterfall is to complete a set of activities before moving to the next. All of these activities are defined and described in detail as part of the planning process, resulting in a schedule that is "predicted" based on known information.

Waterfall, Waterfall, Everywhere...

Waterfall was how most folks managed software projects in the '90s. Although variants such as "Spiral" and "Staged Delivery" added some variety in the margins, Waterfall was the main game in town. So, we kept right on using Waterfall, expecting (insanely?) that it would eventually work for us, despite plentiful first-hand evidence to the contrary. In project after project, we spent loads of time and energy estimating and planning our work "up front", yet things seldom seemed to go as planned, and we would end up in a stressful dance around schedule, features, resources, and quality. Everyone wanted our projects to succeed, but elaborate contortions were often required to rationalize declarations of success, since (strictly speaking) we usually parted ways from the original plan soon after the project started.

As it turns out, while there are many parts of a software projects that can (and should) be planned and estimated accurately, the design and implementation parts cannot. Definition of features and functions is an imperfect process based on imperfect information and filled with speculation. Since this shaky definition process represents the foundation of all that follows, it's no wonder that we have trouble estimating the effort required to design and implement code for these definitions, and sometimes end

up developing the wrong stuff.

Why Waterfall Is So Sticky

No matter how sideways things went, we continued to bang our heads against the Waterfall. Why would otherwise intelligent people continue to use a process that seldom seemed to work? It's because "Waterfall Thinking" is intuitively compelling to humans, and firmly embedded in American management culture.

Waterfall methods are "sticky" (i.e. hard to get rid of) because Waterfall-style planning and control techniques actually work quite well for so many human endeavors, where well-understood tasks can be laid out in planned sequence and then executed. Waterfall reflects the way we naturally think about planning, i.e., in a sequence. We do Waterfall planning when we arrange our daily schedules. It's a good way to get things done. It works for a lot of things, including certain classes of software projects.

As managers we seek control over our work. Waterfall methods, with their seemingly precise Gantt charts and plentiful documentation, tend to establish the illusion of control over project execution. Over time, in order to make Waterfall work better, we've surrounded it with tons of monitoring mechanisms, documents, and project management tools and techniques (see the Project Management Body of Knowledge [5]). But, as we have learned through experience, this control model is based on assumptions that simply aren't valid. At its core, the Waterfall approach assumes we can predict the progress of an inherently creative activity, which in reality does not necessarily follow a linear, predictable path.

But, since the project methods seem so sound, why aren't we on track? Unfortunately, a common answer: it's the people. They aren't working hard enough to make the plan come true. They aren't smart enough. They aren't productive enough. This tends to be a common conclusion with Waterfall project fails, which is wrong. We should look for problems with the system first, and the people last.

Software is NOT CARS, OK?

Waterfall methods have a long and successful history in many industries, most obviously in assembly-line production of items such as cars, and also in any business where the activities can be measured and reduced to the most efficient sequence of predictable steps. Waterfall works fine when applied to

designing and building all sorts of products, from cars to foods and anything else that is made using a sequential process. The power of this "construction" metaphor rises from its successful application, and explains why it continues to endure in software, despite mountains of evidence that you can't always reduce software development to predictable steps. In projects developing net-new software, the tasks are basically a one-shot deal; the specific code hasn't been written before, and won't be written again.

Why Not Exploit the Goodness of Inevitable Change?

In fact, not only can you not reduce complex software development to predictable steps, but you don't want to. Proponents of alternate methods insist that *efforts to restrict the variability of software are actually discarding a chief value of software, namely, the ability to change it quickly***.**

Let's think about it. One of the cool (i.e. high-value) aspects of software is that it can be changed easily and quickly. But by freezing the software requirements at the beginning of a project, we essentially throw away this valuable characteristic, forcing project members to ignore information they discover during the course of developing and testing the code. This can result in releasing products that could have been better if the learning had been incorporated, or worse, releasing products with features that are off the mark and don't provide substantial value. Sometimes we even release features that are already obsolete (they were defined months ago), because we were sticking to the plan. More often, we develop stuff that never makes it to production, which is a terrible waste of resources.

In any case, change happens whether you like it or not. In practice, what often happens using strict Waterfall is that magnitude of change occurring during the course of the project can overwhelm the Change Management processes of Waterfall, which are oriented toward controlling change rather than incorporating it in an efficient manner. A large amount of change to the plan results in an equally large amount of hand-wringing, political intrigue, developer-bashing, and other non-value-added activities that sap the project of momentum and defocuses the team from their primary mission, which is to develop good software. The amount of time and energy expended in efforts to make current reality agree with the predictions of an outdated project plan can be astounding.

What is needed is a project management method that accommodates change and reduces the amount of work spent on futile attempts to predict the future.

Along comes Agile...and Lean

Eventually, some smart and/or frustrated folks realized that there was value in monitoring, learning, and adapting during the course of a project, rather than following a brittle plan cooked up at the start. That's how to take advantage of discoveries along the way, while ditching any parts of the plan that turn out to be faulty; by changing course quickly, one could always be on the best course, given that the end goals are clear.

These folks created the "Agile Manifesto [6]", which defines a set of principles for developing software in an "Agile" manner. These Agile principles do not prescribe any specific project management processes or Software Development Life Cycle (SDLC) models, but do define a different way of thinking about what is important when developing software. It's a cultural change as much as a process change.

Scrum [7] project management was introduced as an Agile framework for developing software (many people equate Scrum to Agile, which is not the case. Scrum is one of many Agile techniques, albeit the most popular one). Scrum was a revelation to those of us with deep battle scars from years of grappling with Waterfall projects. Unlike Waterfall, Scrum does not depend on a large up-front analysis exercise that produces reams of documentation (Market Requirements Document, Product Requirements Documents, Technical Specification, Architecture Document, various Design Docs, etc.). Rather, teams get to work more quickly writing code.

It turns out that developers can write code as fast as analysts can write documents, the difference being that code can actually be used by customers (internal or external). Modern software languages and programming frameworks allow developers to implement a lot of functionality very quickly. So why not write some software and try it out? It can also be re-written (fancy term: refactored) quickly, so the investment in the first version is minimal and can be thrown away if necessary.

With Agile techniques, teams get right to work with brief "User Stories" that define specific business functions that are needed by the user. Programmers develop code in short iterations that result in actual working software, starting with minimal functionality and adding to it with subsequent iterations, until the product is complete enough to show to users. Feedback is incorporated at the beginning of each iteration, so that the direction of the design is constantly adjusted to reflect learning and new information. A high degree of collaboration is required among developers and users.

This is called "Empirical Process Control" by those who care about such things. Basically that means trial-and-error, but is more positively called "Inspect and Adapt". You create some software, see how well it satisfies needs, then change it as required. Perhaps it is shouldn't be surprising that this method can be directly related to Deming's "Plan-Do-Check-Act (PDCA) process improvement cycle.

This Changes Everything

The emergence of Agile and Lean management practices (including Scrum [8] and Lean-Kanban [9]) has forever changed the way we think about planning and executing the design and delivery cycles of software, although Waterfall (or equivalent) will always be the process-of-choice in some, maybe most, industrial software settings. Most importantly, in terms of the management toolkit available to run projects, today's software managers have more and better options than we did back in the day.

Perhaps the most important "sea change" in managing software development is widespread acknowledgment of the inherent, large, and uncontrollable variability in the software development process. Our apparent lack of ability to plan things precisely in the first place, and then control change over a long development cycle, has been confirmed by reams of data now available on project success rates. No longer is the lone software development manager asserting that she/he simply can't predict how long it will take, and suffering the wrath of Waterfall-oriented executives; the data are there to back her/him up.

Next Installment: Agile and Lean Revolutionize Software Project Management

Part II will address how Agile and Lean techniques for managing software projects have changed the playing field. In the meantime, just for fun, here's a parody website that makes fun of all the conferences and gatherings around Agile - only this one is a Waterfall Conference [10].

- IT Management [11]

Source URL: http://www.ceptara.com/blog/agile-scrum-waterfall-comparison

Links:
[1] http://www.cbronline.com/news/software_project_failures_hit_5_year_high_220609
[2] http://calleam.com/WTPF/?page_id=1445&goback=%2Egde_80164_member_181716641
[3] http://www.galorath.com/wp/software-project-failure-costs-billions-better-estimation-planning-can-help.php)
[4] http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf
[5] http://www.pmi.org/en/PMBOK-Guide-and-Standards/Standards-Library-of-PMI-Global-Standards.aspx
[6] http://agilemanifesto.org/
[7] http://scrum.org/
[8] http://www.scrumalliance.org/learn_about_scrum
[9] http://www.leankanbanuniversity.com/what-lean-kanban-0
[10] http://www.waterfall2006.com/

[11] http://www.ceptara.com/taxonomy/term/6