Drowning in the Waterfall:

The Benefits of Agile versus the Predominance of Waterfall

Guy Wyant

CS 390: Software Engineering

October 29, 2008
The waterfall model was created in 1970 as an example of a software development methodology that does not work [1]. The founding document of the waterfall model, created by Winston W. Royce, included two figures: one of the model itself and a second illustrating the major problems inherent in the model, or the reasons not to use the waterfall model. It might seem a strange thing, then, that the waterfall model became one of the most popular programming methodologies after its publication and remained such for years. Even today, according to some surveys, the waterfall model is being used by a large portion of the software engineering world [2]. Why would so many software companies still be employing the waterfall model for developing software, especially since agile systems, such as XP and Scrum, have been the “hot topic” of the software engineering community for years? There are numerous reasons why, but the most important is a natural resistance to changing a tradition and the complications surrounding that issue.

In software engineering, new methodologies are generally advertised as carrying generous improvements in time to release, efficiency, and number of faults per release. But the very word *improvements* implies that there must be some established system or body of results against which to compare the new methodology’s claims. Many times, the compared system of software development is the waterfall model, and the improvements are measured against the existing advantages and disadvantages of that model. Thus, when proposing a system that is, for instance, “more efficient” and produces fewer errors per line of code, one must append “than the waterfall model” or something even more generic, such as “than the previously used method of software development” to allow the claim any credence. To establish a motive for our interest in how many developers are using the waterfall model versus agile programming, we must establish that there are some grounds for expecting developers to choose one or the other. The benefits of agile over the waterfall model in numerous areas of the software development process are grounds enough to warrant interest.

A survey conducted in 2008 showed that 67% of agile practitioners experienced improvements to their software development process [3]. Improvements brought by agile programming come in many of
the different stages of development, and often do not take long to show themselves in the overall process – indeed, some software architects experienced benefits from the very outset [4]. Instead of spending large amounts of time forecasting the duration of the project, predicting all possible features to factor them into the predictions, and writing up specifications and documentation, developers could get started right away after being informed of the first set of functionality. This approach reduced overall time spent on design, since developers did not have to document features that could not be developed anyway due to deadline restrictions. In a sense, the agile methodology advocates “getting the ball rolling,” causing productivity quickly while also giving management time to better define a direction for the entire project. The short iterations in agile programming also help ensure that the development team is a close-knit task force tackling the central goal of each iteration, rather than drifting apart in purpose as is common in the waterfall model.

The phenomenon of drifting apart in purpose is one problem when speaking of the programmers, and another entirely when speaking of a disconnect between client and developer. One of the real strengths of the agile programming methodology is the focus on increased communication between the development team and the client [3]. The increase in communication results in several benefits. Unlike the waterfall model, agile takes into account the fact that most clients do not know what features they want their product to have, or even exactly what the product should be, at the outset. A team using the waterfall model might collect an idea of the client’s expectations, only to find that those expectations were misinterpreted, after months of production [5]. In the agile system, such a misinterpretation would be found directly after the iteration involving the expected feature (if not during the iteration itself), which by the nature of agile, is never a long period of time. The constant stream of communication between the client and the developer helps refine the product bit by bit over time, rather than making the developer guess at what the client wants to achieve.

Since the clients themselves often do not know what they want, the prototypes created as frequently as after every iteration in the agile methodology help them narrow down their plans for the
project, thus helping the developers focus on the most important parts of the product. This unswerving focus on the most important parts of the software leads to a decreased likelihood of a delayed release date, since the more trivial features can be scrapped or released as an update if release dates come sooner than anticipated [6]. This is in contrast to the waterfall model, in which important system infrastructure may not be fully constructed until late in the project. At that point it may be too late to implement a major feature on top of that infrastructure if any problems are discovered along the way, and the waterfall model has then caused another late release.

Delivering the product on-time stands right alongside releasing a fault-free product in the discipline of software engineering [7]. Fault catching and correction is another area where the waterfall model fails to compete with agile programming. With agile, each functionality is rigorously tested during and at the end of each iteration, thus catching errors when they are highly localized and cheap to fix. In the waterfall model, errors cascade down the process waterfall, only to land in a heap at the end of the project. Thus, at the time when the software developers should be excited about the release of their product, they are instead scrambling around at the base of the waterfall, wondering how they will dig out all the errors before the clients arrive on deadline day to see what they hope will be a glistening, elegant waterfall. All imagery aside, the math also supports agile’s superiority in fault correction: in the aforementioned survey, 45% of participants noted a decrease in defects in their products [8].

Every software engineering methodology has its proponents and opponents, and the above reasoning does not intend to show that the agile methodology is free from criticism. One of the largest issues is the lack of focus on documentation and specifications that some agile methods prescribe [3]. The documentation process can become chaotic, for instance, when after seeing prototypes, the client would like to make numerous changes to the product. Versioning these requirements and keeping documentation up to date is a real issue, and some have had to resort to less than ideal methods, like keeping notes on paper, to keep up when the project is going full speed ahead. Another problem with the agile methodology that opponents cite is the difficulty inherent in attempting to run an agile project when
its team members are not all working out of the same physical location – not a small criticism, since 48% of survey respondents said their teams were distributed [8]. Teams spread over time zones have an especially hard time with effective communication between members, and find it harder to receive the client’s feedback, thereby diluting some of the strengths of agile. Other criticisms of the agile methodology center on its seemingly chaotic nature, with its lack of distinct milestones and apparent disregard for discipline in the planning of design and documentation [5]. From a reasoned point of view, taking pros and cons of both agile and waterfall methodologies into account, the benefits of agile programming clearly outweigh its criticisms. Those criticisms do, however, give at least a beginning explanation as to why more software developers are not using agile.

One might not gather the full importance of the disparities between the agile and waterfall methodologies without understanding how widespread each is today. A study conducted in 2002 showed that even though agile methods have been around since the 1980s, the dominant software development process was the waterfall model, used by over a third of the 200 developers that responded, accounting for thousands of software projects [3]. The rest of the participants used either agile programming, a custom-built methodology, or used (as was predominant in this category) no particular method of software development at all. An update to that survey was conducted in 2008 with barely a shift in the results [2]. The numbers using the waterfall model remained around a third; however, those that used agile methods did outnumber those that used the waterfall model. Clearly there is some formidable force that is preventing more software development companies from converting to agile development.

Many times senior management will be the hindrance to the necessary change [9]. A software development team may convince their management to give them go-ahead to use agile methods on their next project. The problems begin when management does not understand the full implications of moving to an agile system, and does not allow the developers enough freedom to implement the system, with all its differences to the legacy methodology, whether it be waterfall or no methodology at all. When a precise notion of progress cannot be determined, management may pull the plug, insisting on a restart.
using traditional methods, or simply not allow future projects to be developed with agile. From then on senior management has had a bad experience with agile that they will remember anytime someone proposes a shift to trying such methodologies again, likely leading them to refuse while citing the logic of “at least the waterfall model is reliable.”

This unwillingness to change is actually the major issue slowing the expansion of agile methods. The new methods are considerably different from the strict sequenced approach of the waterfall model, and thus a lot of preparation – mentally and procedurally – is required to give the agile system a chance [6]. The iterative approach that agile methods propose needs a commitment to a greatly increased degree of communication that is not required by the waterfall model, and without this paradigm shift, the benefits of agile fall to the wayside [4]. If the client has not had experience with the agile development system, they may be wary of the amount of input they must give, or of how little design work they will get to see before the project takes off. Due to this uncertainty, converting to agile development can be more complicated than one might think. That is, some of the problem in migrating from the waterfall model to agile lies not just in getting software development companies to change, but also convincing the clientele that the change is for the better. Without a wide client base supportive of agile, it is hard to make the benefits of agile programming appealing enough that developers will consider it a valid option. This disconnect in methodology between developers and clientele remains one cause of industry stagnation, and a reason why more have not switched to agile programming [6].

One might find it surprising that more software developers have not switched to using one of the various agile programming methodologies, considering the improvements they provide. Agile methodologies encourage a rapid start on the project, and keep team members closer in sync with one another. Using agile, the software development process is responsive to the inevitably changing requirements of any project, and does not have the number of errors or the panicked rush to eliminate them when nearing product release that waterfall model projects find difficult to avoid. Though there are several problems with the agile methodologies, the main issue that seems to prevent developers from
embracing agile remains the unwillingness to do just that – to embrace a methodology so different from what they, and their clients, have used before. Until agile moves from being a buzzword in every headline to being a process studied seriously and developed by people not looking to cash in on a trend, the software development industry may be stuck in the rut of using clearly ineffective methods like the waterfall model – a method developed solely to showcase a weakness in software engineering processes.
Works Cited


