**WHITE PAPER**

# How Agile methods resolve chaos and unpredictability in software projects

Author:
Jack Milunsky
Scrum Master and COO
Brighstpark3

January 2009

---

## INTRODUCTION

This paper attempts to show why an Agile methodology makes sense for managing software projects, as opposed to traditional waterfall methodologies, commonly used to date. The paper will make the case that software projects are inherently chaotic and unpredictable, and as a result, cannot be managed by processes that are best suited to well-defined problem domains. The paper will also make the distinction between the status quo in waterfall development and hyper-productivity, where organizations, such as Yahoo, have experienced ROI's in the order of 666% on one project and 250% overall, since adapting Agile Software Development methodologies.

At the heart of any decision to adopt Agile Software Development, is facing up to the harsh realities of your own current software development experiences. If you are satisfied with how your projects are being managed, and your ability to achieve results, perhaps you should not be thinking about Agile Software Development in the first place. However, the chances are that you are one of the many that have encountered some degree of bad experience with either late delivery, buggy software or, dare I say, software that is either incomplete or does not meet end user or customer needs. Most of us, unfortunately, have experienced all of the above and more - not to mention very stressful working environments that characterize or underscore these experiences.

## THE AGILE METHODOLOGY

Agile Software Development methodologies evolved out of a need to address the dire problems facing the software industry. The most notable "state-of-the-union" report on the software industry was published in the now famous "Chaos" report by the Standish Group in 1995.

Hard facts from the report include:

- Average project success rates 16.2% - this means that a staggering 83.8% of projects were either challenged or impaired
- An average of 52.7% of projects cost 189% of original budget

- Only 61% of originally specified features and functions were available in failed or impaired projects
- Over 1/3 of challenged or impaired software projects experienced time overruns of 200 to 300%

In 1995, more than 250 billion dollars were spent on software projects in USA. These failures represent a significant impact on product expenditure, which in turn affects the economy as a whole.

This picture is not attractive and of course many projects still fail today. I am aware from my own personal experiences and talking with many of my colleagues that most projects do not proceed as expected. A close working colleague in the medical software business spends huge amounts of time on up-front planning and preparation of very detailed project Grants. A tremendous amount of valuable time is invested, and very rarely do they ever reach their milestones as planned. This begs the question as to what function is being fulfilled by all this up-front planning.

Fortunately, today there are real solutions to these problems, as those provided by Agile software development methodologies. Most of the aspects or characteristics of Agile Project Management, including Scrum, have their roots in either the suggestions found in the Standish report or from further studies performed by field experts in the field (Hirotaka Takeuchi and Ikujiro Nonaka, Jeff Sutherland, Ken Schwaber et al).

Clues on how to fix the problems were provided in the Standish report and are listed below.

In the top 5 were:

1. User involvement
2. Executive management support
3. Clear requirements
4. Proper planning
5. Realistic expectations
6. Smaller project milestones

All of these aspects are addressed by Agile Project Management processes like Scrum, the details of which will be discussed in a paper that will follow.

It is important to understand a second fact about non-Agile projects, the origins of which are best understood in industrial process control theory, and provide scientific research in favor of Agile Software Development methodologies.

There are two types of process control systems: defined processes and empirical processes.

Defined processes are those that, given a certain set of inputs, and by providing a certain set of controls, can always attain a specified outcome and be repeated. These types of systems are referred to as "white-box" systems, as the processes are well defined and understood.

Empirical processes, on the other hand, are referred to as "black-box" systems. These processes are generally complex in nature, not well understood and have no defined set of controls that can be applied to repeatedly generate the desired outcome. Such processes have unpredictable outcomes, and can only achieve desired outcomes empirically. In other words, one has to apply a certain degree of control, measure the output, adjust the controls

and repeatedly do this until the desired outcome is finally reached - like a missile homing in on a target.

Software is considered to be such a complex system, as there is no way in which one set of controls can be put in place in order to provide a predictable or repeatable desired outcome. Some of the reasons for this lack of predictability are in large part due to the high degree of uncertainty surrounding the technology, people interaction and changing requirements. From my own experiences as a software engineer, even with highly detailed upfront user interface designs, specifications and plans, the software produced turned out different from its original intent. I attribute this to the fact that once end users see the software and use it, one realizes there are often different and improved ways of doing things. Software development is a creative process. Merely changing one member on the team or sending a developer on a technology course can yield a different outcome.

So, applying defined process methodologies to intrinsically unpredictable and unrepeatable systems is not going to work. Waterfall methodologies, which most software teams are currently using, are a form of defined process, as all of the unknowns are expected to be solved up-front. Waterfall methodologies pre-suppose that software development is a defined process, i.e. well understood. Yet, this is furthest from the truth. Consequently, any "large" up-front effort to fully understand the problem domain is considered wasteful. If one borrows from Lean thinking, excessive upfront planning can be thought of as inventory on the shop floor, which is a liability rather than an asset.

Software development, on the other hand, depends heavily on ingenuity, invention and creativity, and as a result leads to a climate of much uncertainty and chaos. Uncertainty is not something that you can just "plan away" (Mike Cohn) with a ton of up-front research and design. One has to accept that there is uncertainty and that one has to rather provide tighter controls (inspect and adapt points) and more fluid processes to deal with these shortcomings.

Successful teams, as pointed out by Hirotaka Takeuchi and Ikujiro Nonaka (New Product Development Game 1995), all exhibit a high degree of instability. This allows development teams to operate on the bleeding edge. Teams that are faced with tough competitive situations have to be revolutionary rather than evolutionary. Thus, the business of "new" product development is about attempting to produce the best and to be the best, – i.e. responding to competitive situations, leapfrogging the competition, breakthroughs in technology etc. and most importantly, reaching a state of Hyper-productivity (Jeff Sutherland). This requires the teams to be on the high end of the risk curve. This baked-in instability can be somewhat chaotic and Scrum/Agile provides mechanisms to better cope with this chaos by embracing it.

Jeff Sutherland provides insight into what triggers this hyper-productive state - a theory called Punctuated Equilibrium. Punctuated Equilibrium is a theory of evolutionary biology which suggests that evolution tends to happen in ‘fits and starts’, sometimes moving very fast, other times moving very slowly or not at all. If one studies fossils of organisms found in successive geological layers, one will see relatively long intervals in which nothing changed ("equilibrium"), "punctuated" by short, revolutionary transitions, in which species became extinct and were replaced by wholly new forms.

In the same way, revolutionary transitions in software development lead to big breakthroughs in either technology or efficiencies, or both. This is something that should be encouraged. As a result, more chaotic projects that are managed well will transcend barriers and bring about orders of magnitude in progress and technological advancement. The

Scrum process underscores these situations of both rapid change and steady state. The Backlog representing the rapidly changing requirements (environment), and the Sprint representing the period of stability, where the team is left to its own devices to get real work done.

Jeff Sutherland also reflects on how software development can be compared to Complex Adaptive Systems theory in which the world is viewed as a non-linear place. This theory is based on relationships, emergence, patterns and iterations. Rather than being planned or controlled, the agents in a system interact in apparently random ways and from these random interactions patterns emerge that impact the system in positive ways. All of these concepts are reflected in Scrum, which relies on a high degree of coupling between individuals through small teams, broadband communication and collaboration, daily Scrums and reflectives as a way to tolerate chaos. Out of this chaos, new architectures, features and functionality evolve over time.

## THE EMERGENCE OF SCRUM

Scrum Project Management emerged as a process that best reflects what can be found in nature and provides the rigor, controls and measurement to manage unpredictable software projects and environments, in a simple, meaningful and repeatable way.

In order to understand how Scrum Project Management operates, it is important to first understand what Scrum is. Scrum is a rather simple and loosely defined framework. It is not a methodology as Ken Schwaber points out. Methodologies typically spell out exactly how to do things. On the other hand, Scrum provides a learning and feedback system for teams to figure out for themselves how best to manage and deal with problems that arise during the course of the project.

As mentioned earlier in this paper, one needs rigor and control in order to manage complex systems development. Scrum provides this rigor and control through the implementation of best practices as well as very specific Inspect and Adapt points at various stages in the Scrum life cycle, which is further described below.

Scrums best practices are all centered around sound engineering discipline and technical excellence as a means to cope with changing environments and still provide sustainable progress. Best practices such as good design, continuous refactoring to keep interfaces and code clean, automated and continuous integration of code, automated unit tests and integration tests are some of the rigid development practices that are implemented by most Agile Software Development teams. It takes hard work to allow for sustained throughput in such inherently unstable environments. And it is this hard work that separates the best teams from the mediocre ones!

Scrum provides specific touch points for learning. Ken Schwaber calls these "Inspect and Adapt points". Initially, the Sprint Planning Meeting takes place in which the team works with the Product Owner to figure out what they are going to build and then cross-functionally determine how they are going to go about doing this. Each day, during the Daily Scrum Meeting, the teams have the opportunity to evaluate how they are doing in relation to their initial plan, and figure out how to adapt based-on knowledge acquired from the previous day. At the end of the Sprint, the Sprint Review Meeting provides the Team with a chance to show the Product Owner what has been developed. This provides further opportunity to learn how the Team has succeeded relative to the goals set at the beginning of the Sprint. At this point, the Product Owner can accept or reject the completed work. Any changes required are re-prioritized and fed back into the system via the Backlog.

After the Sprint review meeting, the team has accumulated all the information necessary to evaluate performance and further enhance productivity.

All these Inspect and Adapt points are designed to help teams learn, adapt, improve and deal head-on with information that emerges at each of these points, whether the information is good or bad. Scrum is therefore a true test of a team's character.

In addition to this, Scrum also provides a mechanism to deal with all this change. The Sprint is really the container in which all the chaos behind the backlog is tamed. The Sprint provides "quiet" time for developers to work uninterruptedly on goals set at the beginning of the Sprint. This is the time during which the Team self organizes around a common set of goals and hunkers down on solving tough development problems.

## CONCLUSION

So, if you are working on simple projects, projects that are well defined and their scope is understood, then waterfall will suffice. Alternatively, Agile is the way to go, and I recommend Scrum as the best Agile implementation available, for new product development projects. Scrum is practical and simple, and functions well with other processes like Test Driven Development and XP, and most importantly, it provides visibility of a team's progress throughout the development process.

## ABOUT THE AUTHOR

Jack Milunsky, COO Brightspark3

As Chief Operating Officer and Scrum Master Jack Milunksy heads software implementation at Brightspark3, he leads the teams' efforts in building and implementing innovative products through Agile Project Management and Scrum tactics.

Jack combines 18 years of experience managing software development teams both large and small. He is an early adopter of Agile and has a great passion for early stage startups.

Prior to joining Brightspark 3.0, Jack was VP of R&D at Tira Wireless where he was responsible for driving technological innovation in the company. He has held many senior positions in Hi Tech companies across diverse industries. He was the Director of R&D for Delano Technology Corp and he also served as the Director of R&D for Symantec/Delrina Corp.

Jack holds a BSC Elec (Eng) degree from the University of the Witwatersrand in South Africa and completed Business Administration from the Stanford University Business School.

For more information, visit blog.agilebuddy.com or email jack@brightspark3.com.