IJITKM Volume 8 • Number 1 June-Dec 2014 pp. 25-29 (ISSN 0973-4414)

Agile Methods vs. Traditional Methods

Rajesh Popli
Manager (Quality), Nagarro Software Pvt. Ltd., Gurgaon, INDIA
rajesh.popli@nagarro.com

Abstract
Software engineering is a relatively new field. Technology is going through rapid improvement and best development practices and methods are constantly being created. These characteristics are reasons which make software projects difficult to manage. Developers and managers have struggled to find suitable models and practices, but troubles have persisted. Missed deadlines and failure of projects is common in the software engineering industry. Software projects cannot be carried out in the same way as projects in traditional industries and thus new methods and practices are required. Agile software development has brought new insight into the way software projects are being approached and offers effective ways to deal with the complexity of software development. Agile is a framework that promotes development iterations throughout the life-cycle of a project. It minimizes risk by developing software in short amounts of time. This paper presents an overview of Agile Project Management & Development Methods.
The purpose of this paper is to compare traditional waterfall project standards and deliverables with those in Agile.

1. Introduction

For decades, software development projects have followed the classic "waterfall" methodology in which software development initiatives were carefully analyzed, designed, documented, coded, tested, and ultimately delivered to the customer. It may sometimes take years after inception. By then, it was not uncommon for business needs to have changed and for the resulting system to fall short of customers' expectations. According to the Standish Group, software development projects have an overall success rate of 34%. In response to this rather disappointing approach to software development, "Agile" methodologies – which are light on documentation and formality - began to emerge in the 1990's. Scrum, which is one of several Agile approaches, was first developed and presented in 1995 so it is relatively novel when compared with traditional software development processes which have been used for decades.
Software development is not a defined process; it differs from physical engineering processes. A physical engineering process consists of distinct phases. The **design** phase is difficult to predict and requires creativity and imagination. The primary activities in the design phase are intellectual. The design phase is followed by **planning** for the **construction** phase, and then by the construction phases itself. The primary activities in the construction phase are physical, and it is much easier to plan and predict than the design phase. In many physical engineering disciplines construction is much bigger in both cost and time than design and planning. For example, when you build a bridge, the cost of design is about 10% of the total job, with the rest being construction. For the analogy between software development and physical engineering to hold, we need to be able to separate design from construction, be able to produce a predictable schedule, design artifacts that are complete enough for construction to be straightforward, and be able to perform construction with people of lower skills (who are hopefully cheaper to employ). Construction also needs to be sufficiently large in time and effort to make this worthwhile. The situation in software development is quite different. Different experts report the following breakdown of Software development by activity:

- Analysis 16%
- Design 17%
- Code/Unit Test 34%
- System/Integration Test 18%
- Documentation 8%
- Implementation/Install 7%

2. The Pitfalls of Traditional Methodologies

Traditional methodologies try to be predictive - to create a schedule at the beginning of a project and to conform to this schedule for the life of the project [1].Complex software systems can be built in a sequential, phase-wise manner where all of the requirements are gathered at the beginning, all of the design is completed next, and finally

the master design is implemented into production quality software. This approach holds that complex systems can be built in a single pass, without going back and revisiting requirements or design ideas in light of changing business or technology conditions. Yet a common complaint is "the problem with this project is that the users keep changing their minds". In the physical world people accept that requirements need to be fixed because it's intuitively obvious to them that, because of the expensive construction phase, it's very expensive to make changes after a certain point.

Figure 1: Traditional Sequential Approach.

However, software is much less tangible. Not only is it hard to be precise about what's needed, it's also hard to see why it should be difficult to change later. Customers **expect** software to be soft. Traditional methodologies establish procedures that discourage requirements changes, they resist change. This helps them maintain a predictable schedule, but it does nothing to ensure that the final results meet the customer's real, changing needs.

3. Emergence of Agile Software Development

During the 90s a number of different people realised that things had somehow changed [2]. These people became interested in developing software methodologies that were a better fit with the new business. Although the details of these methodologies differ, they all share certain underlying principles, to the extent that these methodologies are often now grouped under the title "agile methodologies".

Agile practitioners pride themselves on highly productive, responsive, low ceremony, lightweight, tacit knowledge processes with little waste, adaptive planning and frequent iterative delivery of value. Agile follows iterative approach having overlapping phases of software development as in Figure 2.

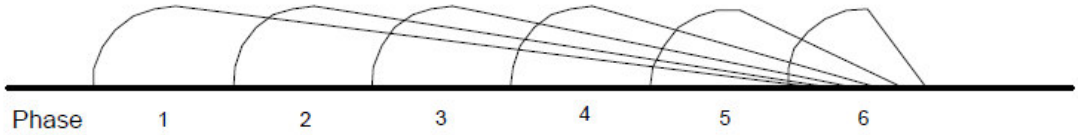

Figure 2: Iterative Approach

The methodologies falling under the "agile" umbrella all address these issues in slightly different ways, however the agile methodologies do have common underlying values and principles. The Agile Alliance expressed the values in the **Agile Manifesto** [3].

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

4. Agile Principles

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices that allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Some of the principles behind the Agile Manifesto [4] are**:**

- Customer satisfaction by rapid, continuous delivery of useful software.
- Working software is delivered frequently (weeks rather than months).
- Working software is the principal measure of progress.
- Even late changes in requirements are welcomed (this does not mean code & run). Instead removing an existing feature or moving a deadline forward to accommodate late/unplanned feature requests.
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (Co-location)

- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

5. Agile Life Cycle

Agile methods break tasks into small increments with minimal planning, and don't directly involve long-term planning. Iterations are short time frames that typically last from one to four weeks. Each iteration is worked on by a team through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders[6].Multiple iterations may be required to release a product or new features as shown in Figure 3.

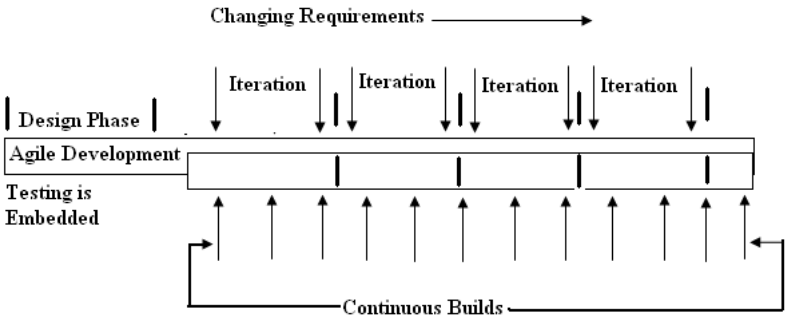

Figure 3: Multiple iterations to release a product in Agile development

This helps minimize overall risk, and lets the project adapt to changes quickly. Stakeholders produce documentation as required. An iteration may not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.

Agile methods emphasize face-to-face communication over written documents when the team is all in the same location. When a team works in different locations, they maintain daily contact through videoconferencing, voice, e-mail, etc. Most agile teams work in a single open office (called bullpen), which facilitates such communication. Team size is typically small (5-9 people) to help make team communication and team collaboration easier. Larger development efforts may be delivered by multiple teams working toward a common goal or different parts of efforts. This may also require a coordination of priorities across teams.

6. Working with Agile Software Development

In the case of working with agile software development, the activities will be done as

- Meet with the customer; create a high level list of features get sign-off on the requirements. Chunk these into features that could be delivered in six-week intervals.
- Ask the customer to prioritize the list.
- Create a detailed plan to implement the first feature including a detailed user acceptance test case.
- Implement the plan for the first feature. Deliver the feature to the customer, involving them with questions whenever necessary.
- First feature is delivered, and approved by the customer. The time it took to deliver is used for a baseline to predict the sizing of future features. The customer is asked again to reprioritize, and pick the next feature to be delivered in six weeks based on the new estimates, Repeat steps 3, redoing existing features as necessary to add new features until the entire solution is in place. The various steps followed in ASD are as shown in Figure 4.

The traditional methodologies aim at making software development more predictable and more efficient. They do not support changes of requirements and the complete system has to be known at the beginning of the project.

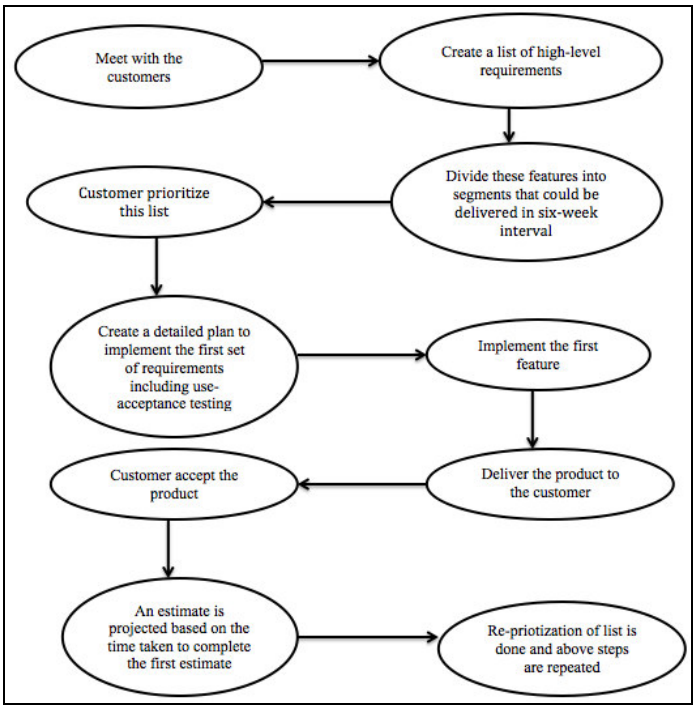

Figure 4: Steps followed in ASD

7. Agile Burn-down Chart

The Burn-down Chart is used as a tool to guide the development team to successful completion of a Sprint on time with working code that is potentially shippable as a product. On a Agile project, the team tracks its progress against a release plan by updating a release burn-down chart [7] at the end of each sprint.

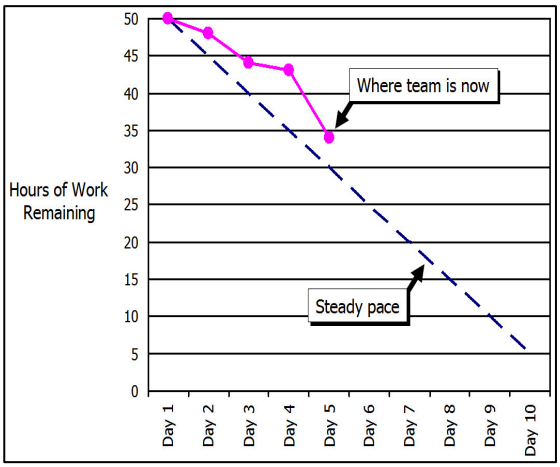

Figure 5: Agile Burn-down Chart

The horizontal axis of the release burn-down chart shows the sprints as in Figure 5; the vertical axis shows the amount of work remaining at the start of each sprint. Work remaining can be shown in whatever unit the team prefers--story points, ideal days, team days, and so on. The cost of curve in Agile is as shown in figure below.

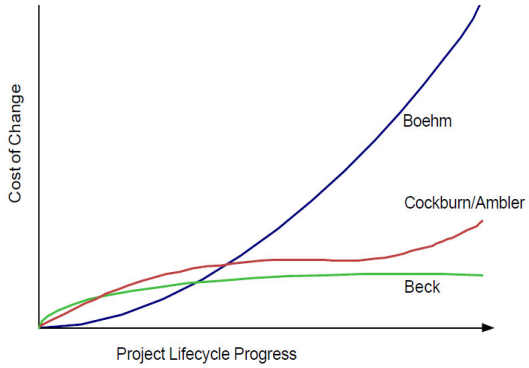

Figure 6: Cost of Change Curve

Agile visionary Kent Beck challenged the traditional cost of change curve evidenced by Barry Boehm [1] over twenty years ago. Beck's method espouses that the cost of change can be inexpensive even late in the project lifecycle while maintaining or increasing system quality as shown in Figure 6.

Conclusion: The demands of the world we live in require more information to be available faster than any time in history. To remain competitive, an organization must be able to respond to this demand. Agile software development methodologies have demonstrated their ability to help information technology organizations respond to this growing demand for their services. Agile methodology is intended to achieve results that are often not possible using traditional approaches. These methods build in processes intended to clear the tracks for the express development train to move at extreme speeds. Agile methods get results by removing impediments to progress, Assuring prompt and timely decision making, Isolating the project and team members from irrelevant issues, Utilizing all resources and expertise required to achieve success and Focusing the attention of team members on the extreme project and nothing else.

References:

[1] **"Agile Software Development"**, Alistair Cockburn, Pearson Education, 2002.

[2] **"Agile Software Development Methods",** Pekka Abrahmson, Outi Salo, Jussi Ronkainen, Juhani Warsta, VTT Publications 478, ESPOO 2002.

[3] Beck, K. Manifesto for Agile Software Development. WWW page of the Agile Manifesto: **http://agilemanifesto.org/.,** Oct 2007.

[4] Beck, K. Principles of the Agile Manifesto. WWW page of the Principles of the Agile Manifesto: **http://agilemanifesto.org/principles.html**., Oct 2007.

[5] Agile Alliance. WWW page of the AgileAlliance: **http://www.agilealliance.org/,** Oct 2007.

[6] **"The Agile System Development Life Cycle"**,Scott W.Ambler, Ambysoft, Managing Agile Projects, 2005.

[7] **"An Introduction to Agile Software Development"** Victor Szalvay, Danube Technologies, November 2004.

[8] **"Agile Software Development Methods: Review and Analysis,"** Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen, and Juhani Warsta *Proceeding of ESPOO 2002*.