Agile Software Development Processes
A Different Approach to Software Design

- Everette R. Keith -

Abstract

In software development there exists a tension between quality, cost, and time. Delivering cost competitive quality software in today's time constrained market is a difficult task. Many traditional software processes are top heavy with documentation and rigid control mechanisms making it difficult applying them to different software projects. New families of processes, referred to as Agile Processes, are making headway into the software industry. These processes focus on code rather than documentation calling themselves agile because, unlike the traditional processes, they are adaptable, not rigid. This paper discusses several of these Agile Processes, the philosophy driving them and the challenges faced when implementing them.

Introduction

Software has been part of modern society for more than 50 years. Likewise, so have software development processes. The software process is the foundation for engineering software. Within the context of his book, Pressman defines "a software process as a framework for the tasks that are required to build high-quality software." [1].

There is a tension between quality, cost, and time impacting software development processes. Software developers have to address two prevailing questions. 1) Why does software cost so much? 2) Why does it take so long to develop a software product? Today's software processes stress quality over efficiency. Studies' show that fixing problems after product introduction can cost from 60 to 100 times more than finding and eliminating the problems during the design phase.

Agile Software Development Processes
A Different Approach to Software Design

- Everette R. Keith -

Today's mainstream software processes do not effectively address the two major questions of cost and time. There is, however, an emerging philosophy producing new processes known as "Agile Software Development". The new processes focus more on people interactions and early development of code than on documentation and planning.

This paper introduces and discusses Agile Software Processes. The paper begins by taking a brief look at today's market and how it impacts the software industry. Next, agile process philosophy is explored followed by an overview of four of the more widely used agile processes. Following this, the challenges associated with implementing agile processes are presented and a case study describing one company's experience with agile processes is outlined. The paper concludes with a summary and the author's views concerning the value of using agile processes cautioning that while valuable they are not a panacea capable of solving all problems associated with software design and development.

A Brief Look at Today's Market

There is a stark difference between yesterday's markets, where survival often depended on restrictive practices applied in relatively stable business climates and today's markets which move at blinding speed. "The accelerated comodification of goods and services in addition to the simple and cheap replication of the business models of others lead to a constant redefinition and fragmentation of markets in the endless search for the next high-margin differentiated products and services." [2]. The pressures introduced by the rapidly changing markets introduce pressures into the software industry requiring them to be as adaptable as the markets into which their products are used.

Agile Software Development Processes
A Different Approach to Software Design

- Everette R. Keith -

A rapidly changing market introduces greater choices into the market. Users and producers of software must contend with issues concerning features, those to include and those to exclude. What technology should be pursued? How can companies differentiate themselves giving them a competitive edge? These, and other business related questions, are always difficult to answer; trying to predict answers to them in rapidly changing markets compounds the difficulty. In fact, from a software system perspective it is highly probable that making long-term predictions in rapidly changing markets is impossible. [2]

Agile Process Philosophy

Agile Software development philosophy has it roots in the reality of today's markets. The emergence of agile software processes attempt to deal with the issues introduced by rapidly changing and unpredictable markets. Reading the "Manifesto for Agile Software Development" [3] the basic ideas of the philosophy are introduced through four basic values:

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

The items on the right have value, however, the items on the left define the agile philosophy. Exploring each of these values will aid in gaining knowledge of the agile process philosophy while exposing how applying the philosophy to defined methods will enhance software development aligning it with today's volatile markets.

Individuals and interactions serve an enhanced role in agile processes. It is a belief among agile process proponents that people can respond quicker and transfer ideas more rapidly when talking face-to-face than they can when reading or writing documentation.

It is an underlying expectation that agile process development occurs in a co-located team environment. An environment where teams work in pairs developing code. The concept of synergy (i.e. The interaction of two or more agents or forces so that their combined effect is greater than the sum of their individual effects.) takes hold because a few designers, sharing a common space, working together, can produce more code quicker than can the same individuals working alone. [3].

Another important aspect of individuals and interactions is that the team is empowered to make all technical decisions. The decision-making process can severely impact a software project because it can be a time consuming process bogged down in a management quagmire. Having the power to make technical decisions in the hands of the technical people is imperative. The implications of this, however, can be far reaching.

Reading [4] we are advised that team members and management must have an equal place in the project. This does not mean that technical people take the role of management. Management still must function as a body to remove roadblocks standing in the way of progress. It is, however, required that management recognize the expertise of the technical team to the point where they are fully empowered to make the technical decisions. Permitting this means, the organization must open other paths of communication between the technical team and the business expertise. Furthermore, the communication must be continuous with no constraints to its access. Decision making, shared responsibility with management and continuous access to business expertise allows individuals to organize around a set of rules operating in an adaptive environment creating innovative and emergent results.

Working Software over comprehensive documentation is a radical concept to many managers in the software industry. This philosophical construct emerges from the idea that the code is the key piece of the documentation. Traditional software processes (big design upfront) on the other hand view the requirements document as the key piece of documentation. A prevailing belief with big design upfront (BDUF) processes is that it is possible to gather all of a customer's requirements, upfront, prior to writing any code.

This is classic engineering. Its success in mechanical and other engineering disciplines makes it attractive to the software industry. Gather all requirements, get a sign-off from the customer and then put into place the procedures (more documentation) to limit and control all changes. The gathering of all requirements up front gives a project a level of predictability and predictability does have value. In fact, software projects that are life critical must have predictability making the gathering of requirements essential. For many other projects, however, this exercise adds a layer of documentation slowing the project down.

In volatile and changing markets it is not possible to gather a complete set of user requirements that are stable and unchanging. Often customers do not recognize what they really want until a system is functioning. "Formal specifications can help to achieve precise and logically consistent requirements, but cannot guarantee their accuracy." [5] Essentially, "In today's economy the fundamental business forces are changing the value of software features too rapidly. What might be a good set of requirements now, is not a good set in six months." [4]. Agile processes link code and requirements tightly together. Users of agile processes view requirements as fluid and changing.

Agile processes introduce the idea of simplicity. Never produce more than what is necessary and never produce documents attempting to predict the future. These documents will invariably become outdated. "The larger the amount of documentation becomes, the more effort is needed to find the required information, and the more effort is needed to keep the information up to date." [6].

Embracing the philosophical premise that the code is the actual design permits developers to move into a coding phase a lot more rapidly, developing code that is representative of the customer's immediate requirements. Rather than agreeing to some abstract design, the customer interacts directly with the system during its design. Doing this, however, does introduce risk because it requires the customer to behave differently as well.

Customer collaboration over contract negotiation is a way of doing business that is new to customer and supplier alike. The philosophy behind this is that the customer must play a participatory role during the design (i.e. coding) of the project. Literally, the customer effectively joins the development team. He or she is on site with the developers working closely approving decisions and guiding the project from a customer perspective. This definition of the customer's role in a project impacts the business side of the project: the contract. In a traditional business setting a fixed price contract is put into place, the development team generates a set of (supposedly) stable requirements leading to a predictable outcome applying predictive processes to achieve the goal. Agile processes do not subscribe to predictive processes and stable requirements. Because of this a new type of contract is required. It is a contract not based on fixed price delivery.

A different business model is required to support agile processes because the customer has a much finer grain of control over the project making changes based on feedback received from the functional code. Obviously, this would never work with a fixed price contract. Instead of coding a system, the developers would end up spending much of their time managing change request documents. On the other hand, from an agile processes perspective, if the customer is not intimately involved with the system as it is being coded the outcome will be less than what was expected hurting both the customer and the developer alike. It is advantageous for both customer and supplier to arrive at a business arrangement supporting collaboration over contract negotiation.

Responding to change over following a plan is one of the key, philosophical constructs making agile processes successful in today's market. As was brought out earlier today's markets are volatile and constantly changing. This makes it impossible to implement a predictive process or to provide a set of stable requirements.

The idea of responding to change rather than following a predictive plan is not something managers embrace, it sounds dangerous, it sounds like hacking code. These attitudes no longer hold in many situations because BDUF process methodologies attempting to control and reduce cost by eliminating change fail. Today, the challenge is not stopping change but, rather, determining how to better handle inevitable changes that occur throughout a project. Change and variation are not results of errors. "External environmental changes cause critical variations. Because we cannot eliminate these changes, driving down the cost of responding to them is the only viable strategy." [7].

The values derived from the philosophy of agile processes serve to manage change and reduce cost. Co-located teams working together producing code instead of high maintenance documentation can help to increase productivity. Furthermore by extending the team's technical decision making capabilities and granting them continuous access to business expertise both internal and through the use of on sight customer participation the code being produced will better reflect the true requirements of the customer. Finally, instead of enforcing a rigid plan using predictive process methodologies with rigid and unchanging requirements embrace change; recognize it is part of today's business climate and work to handle the changes as they occur. Doing this does not mean that there will be a bunch of coders creating chaos with no regard to processes. Agile processes are rigorous requiring a lot of hard work to make them successful.

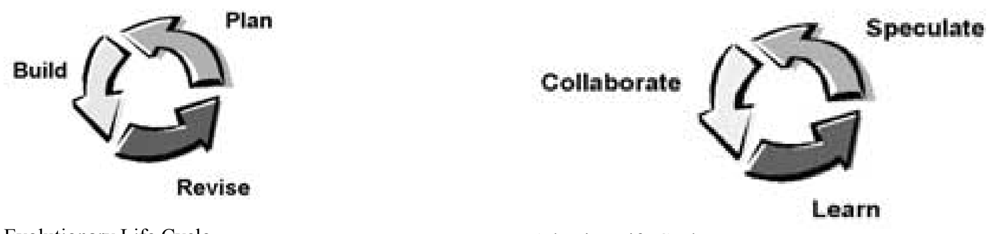
Agile Processes

There are many variations of agile processes this paper provides an overview of four of the most popular ones used today. This paper introduces Adaptive Software Development (ASD), Extreme Programming (XP), Crystal, and the Rational Unified Process (RUP) to the reader.

Advanced Software Development

ASD, developed by Jim Highsmith[1] does not provide the details often associated with a process. Milestones, methods, and deliverables are not specific elements discussed by ASD. Instead, ASD places its emphasis on applying ideas originating in the world of complex adaptive systems (i.e. Chaos theory). ASD provides the fundamental base to develop adaptive systems from which arise agile and adaptive processes.

Reading [8] it is learned that at the core of ASD is the premise that outcomes are naturally unpredictable and, therefore, planning is a paradox. It is not possible to plan successfully in a fast moving and unpredictable business environment. Adaptive development is essential when you have developers, customer, vendors, competitors and, stockholders all attempting to interact with one another at such a pace that linear cause and effect rules cannot assure success. ASD replaces the evolutionary life cycle model with the adaptive cycle model (fig 1).



Evolutionary Life Cycle

Adaptive Life Cycle

Figure 1.

[1] http://www.adaptivesd.com/index.html

Comparing the two models, ASD recognizes that it is not desirable to wander around experimenting endlessly hoping to find success. On the other hand, the word planning is too deterministic for our mostly unpredictable world so ASD prefers the word Speculate. Develop a good idea of where the project is heading, and put mechanisms in place to adapt to changing customer needs, changing technology and a changing market.

Collaboration replaces build because of ASD's recognition that people are the essential ingredient to producing a successful product. Collaboration is the activity of balancing: managing a project, such as configuration control and change management, with creativity the act of trusting people to find creative answers in an unpredictable environment. Completing the cycle is Learning.

Learning is preferred over revise because revise is backward looking. Revise, in the context of the evolutionary life cycle implies that while change may be necessary it is change based on the original plan. Learning, however, is the act of gaining knowledge through experience. Placing the product under scrutiny and questioning all previous assumptions, using the results of the cycle to learn which direction the next cycle should take. ASD is not presented as a methodology for doing software projects but rather it is an approach or an attitude that must be adopted by an organization when applying agile processes. Adopting the premise that the world is fast paced, and unpredictable requiring adaptive processes to compete and be successful is the message communicated by the author of ASD, Mr. Jim Highsmith.

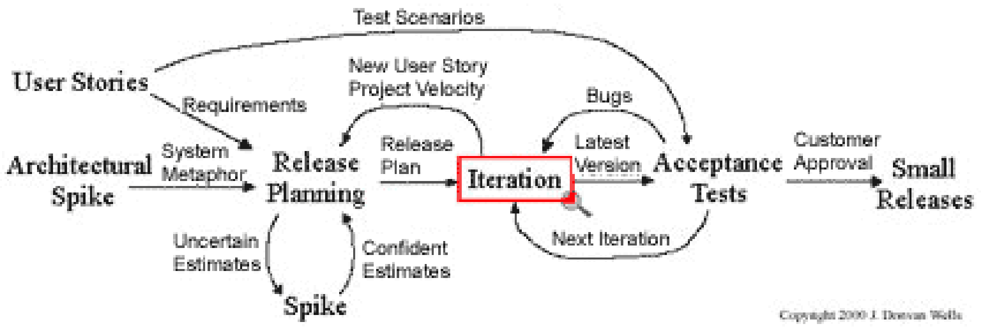
Extreme Programming

Extreme Programming (XP) is a high profile agile process known to many advocates and novices alike and is likely the most widely used (fig. 2). There are four basic values supporting XP:

- Communication – Without communications project schedules slip, quality suffers, and the customer's wants and needs are misinterpreted or overlooked.
- Feedback – The need to check our results is important. Without feedback, a project will most likely fail. Feedback tells how a project is doing while providing direction for future iterations.
- Simplicity – Do not add unnecessary artifacts or activities to a project. Eliminate everything not completely justified.
- Courage – Putting faith in the people over the process requires courage. It is important to realize, however, that if processes do become more important than the people do, a project is headed toward failure.

Extreme Programming has defined practices and guidelines that implementers should follow. The process begins by gathering stories. These are short use cases, small enough to fit on an index card. Each story is business-oriented, testable, and estimable. From the stories, the customer selects the most valuable set. This set comprises an iteration and is coded first. Coding is done in pairs, two people coding on one machine, and an iteration is typically one to two weeks long. Once complete the set is tested and put into production. "The goal of each iteration is to put into production some new stories that are tested and ready to go." [9].

Extreme Programming Project



Extreme Programming: A Gentle Introduction[2]

Figure 2

Testing plays a major role in XP. Each iteration is subjected to unit testing. Writing all unit tests prior to writing any code is mandatory. A particular iteration must pass its unit testing prior to going into production. Customers determine system wide tests. Considering their needs and referencing the stories, customers think about what it would take to satisfy them that the iteration is successful. These needs are translated into system wide tests. Testing regularly and often at the unit level and system level provides feedback and confidence that the project is moving ahead and the system is functioning according to the customer's requirements.

This process of selecting a set of stories, doing short iterations, working in pairs to code, test, and integrate is repeated until the project is complete. Working in short iterations with constant feedback gives the project the chance to adapt to changing needs. The focus is always on the current iteration. No design work is done in anticipation of future requirements. XP is a highly disciplined process. To be successful the organization implementing XP must embrace the XP values and principles. Its proponents would advise that it should be implemented for the

[2] http://www.extremeprogramming.org/index.html

first time on small to medium sized projects and that all process steps should be explicitly followed until experience is gained using it.

Crystal

Crystal is a family of processes each applied to different kinds of projects. The idea of having multiple processes stems from the thinking that some projects require fewer rigors than others do. Small and non-critical projects can be developed using less rigorous Crystal methods. Larger and more critical projects, however, demand more attention and therefore, a more rigorous Crystal process is used.

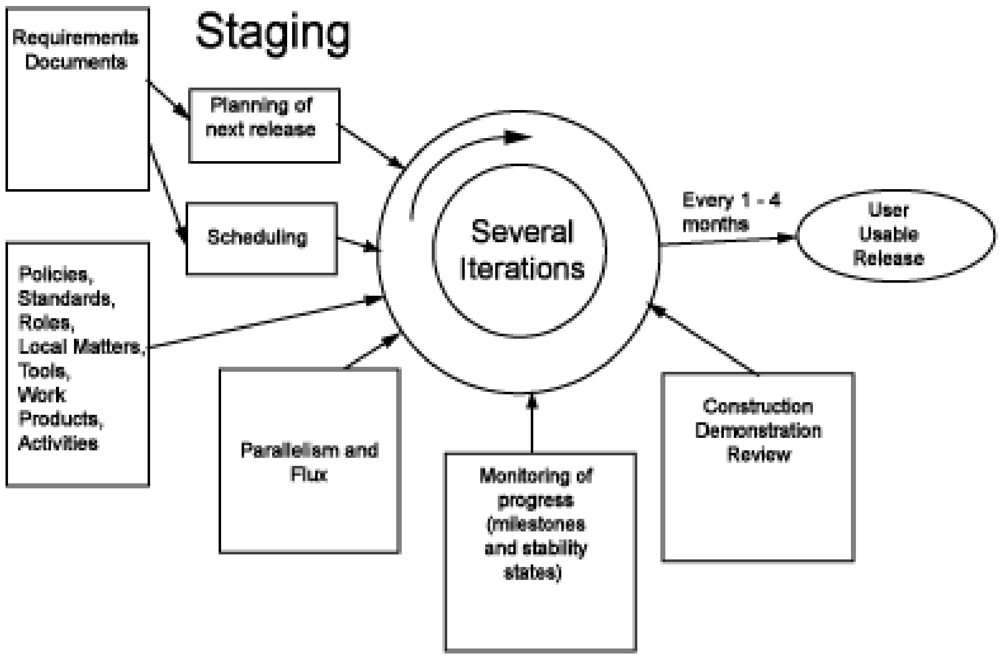
Selecting a Crystal process requires that a project be matched to one of four criticality levels.

- Comfort
- Discretionary money
- Essential money
- Life

A system failure for the first level may cause a loss of comfort whereas a system failure for the fourth level may cause a loss of life. Using this as an example a less rigorous process is applied to the former while the latter would demand a highly rigorous process. Each of the processes shares common policy standards. [10]

- Incremental delivery
- Progress tracking by milestones based on software deliveries and major decisions rather than written documents
- Direct user involvement

- Automated regression testing of functionality
- Two user viewings per release
- Workshops for product and methodology-tuning at the beginning and in the middle of each increment.


Requirements Documents
Staging
Planning of next release
Scheduling
Several Iterations
Every 1 - 4 months
User Usable Release
Policies, Standards, Roles, Local Matters, Tools, Work Products, Activities
Parallelism and Flux
Monitoring of progress (milestones and stability states)
Construction Demonstration Review

One Crystal Orange Increment: Agile Software Development Methods [10]

Figure 3

While each of the Crystal processes share the standards the rigor involved is dependent on the project and the chosen process. For example, less critical projects suggest two-month incremental delivery time spans whereas critical projects, demanding a rigorous process, may extend time-to-delivery increments to a maximum of four months. Projects are comprised of multiple increments (fig. 3). Crystal processes define the functions contained in an increment.

The functions or practices are explicitly defined. Staging, this is the scheduled time frame for one increment ranging from one to four months duration. Increments include several iterations during which construction, demonstration, and reviews are included activities. Iterations are monitored and team deliverables gage the progress and stability of an iteration. User viewings, (i.e. reviews by users), are conducted one to three times per iteration depending on the criticality of the project. Methodology-tuning is a basic technique used to fix and improve the process applying knowledge gained in one iteration to the next iteration. In addition to the methodology-tuning workshops there are reflection workshops conducted at the start of an increment and midway into it.

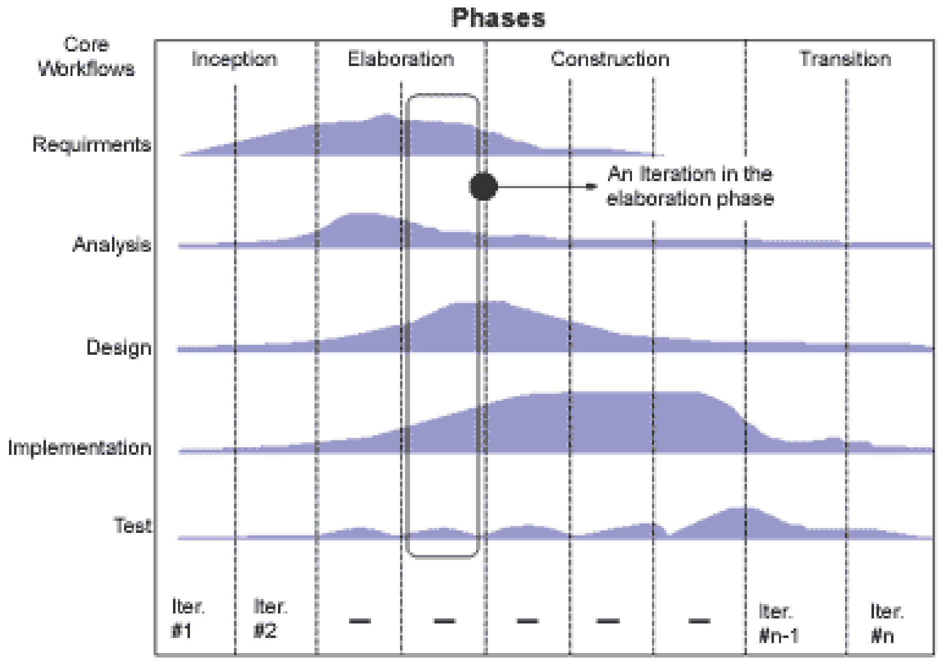
The multiple processes offered by the Crystal methodologies are adaptive and agile. They are adaptive because they offer multiple solutions for projects having different criteria. They are agile because they deliver work products at the completion of each project increment and are able to apply lessons learned to the next iteration. Additionally, they demand customer involvement, and decisions are made based on outcomes rather than trying to force conformance to a plan developed early in a project's phase.

Rational Unified Process

The Rational Unified Process, (RUP), is a generic process framework that uses a specific methodology to accomplish the tasks associated with it. The RUP uses the Unified Modeling Language[3] developing use cases for the software system design. The nature of its versatile framework is something that different organizations can apply to different applications using people with different skill levels working on projects with varying levels of complexity. How it is used depends on the environment and how the project chooses to tailor it. [11].

[3] UML Resource Page http://www.omg.org/uml

In its most simple form its workflows mimic classic BDUF waterfall processes. Gain knowledge of your customer's business, translate the business needs into a set of requirements, do analysis and design creating the software architecture, implement the design and test it extensively prior to delivering it to the customer. These attributes are core to RUP's framework. The core attributes are applied to four unique phases occurring over the lifecycle of a RUP project.



The five workflows take place over four phases [11]

Figure 4.

The tailoring occurs by controlling the iterations (fig. 4). The organization and the project complexity drive the iterations. Organizations could use RUP in its most simple configuration, one iteration for each phase, creating a waterfall style process or use it in a more agile manner creating multiple iterations, having lengths of one to two weeks. Ultimately, "[t]he goal of each iteration is to develop some working software that can be demonstrated to all the stakeholders, and that the stakeholders will find meaningful." [12].

Each phase iterates through each workflow while the project lifecycle moves through the phases. Inception is concerned with gathering requirements and putting together the project plan. During elaboration the software architecture is defined and the detailed plan for future iterations is established. During Construction the software system is built and tested and the user documentation is completed. Finally, the Transition phase closes the project and delivers the product to the customer.

RUP is tailored according to an organization's culture and by project complexity making it highly versatile. It uses defined workflows to complete its work products and has four phases defining the process life cycle. In many ways it resembles a classic waterfall styled process, however, because of its versatility it is generally tailored for use as an agile software process.

Implementing Agile Processes

One of the most difficult tasks involved with using agile processes is successfully introducing them into an organization (i.e. overcoming resistance to existing organizational structures). "Part of [the BDUF] culture is the creation of fiefdoms within the program organization. Adopting [agile processes] will radically change the functions of the organization within the program and consequently change the staff and funding profiles of the organizations." [13]. The traditional roles played by management, Quality and Assurance, test, financials, and Software Engineers (SWE) will all change creating resistance to the introduction of agile processes.

Reading [14] much knowledge is gained concerning the problems experienced by organizations wanting to transition to agile processes. Software Engineers (SWE) are either over zealous or highly skeptical. Over zealous engineers may misinterpret the meaning of agile, taking it to mean “moving quickly” leading toward minimal discipline and turning the project into a hacking free for all. It is important to understand that users of agile processes are making decisions with forethought and reason. On the other hand, there are SWE resisting agile processes because they are strong proponents of design artifacts created using BDUF processes and are most familiar working to a structured plan having a defined time schedule. They do not believe using agile processes produces quality products.

Quality and Assurance (Q and A) and the testing staff often resist agile processes because in the BDUF environment they do not get much attention from management. In the agile process environment, however, this changes because Q and A and testing is a high profile activity occurring after each iteration. Viewing this attention as micromanagement may cause them to resist any change.

Management, too, is often skeptical of agile processes. They are uncomfortable with not having Gantt charts and other documents used to manage projects. It is common for management to judge the progress of a project by looking to see if a particular document exists or not. Recall, these are artifacts associated with BDUF processes and do not exist in similar form for agile processes. Another area that causes distress for management is not having a final commitment date of delivery, a bottom line cost, and all features documented.

Other areas causing resistance to agile processes revolve around the lack of technical documentation associated with agile process methods. "Agile process critics point out that the emphasis on code can lead to corporate memory loss because there is little emphasis on producing good documentation and models to support software creations and evolution of large, complex systems." [15]. Noting these perceived shortcomings offered up by organizations whose experience lays in BDUF processes there are, however, organizations that do transition to agile processes successfully employing them on software projects.

A Case Study

In a case study described by [16] a company (left unnamed at their request) was willing to use Extreme Programming (XP) on a project replacing an old legacy system with a new system. Management voiced many of the concerns noted above. Addressing each concern resulted in a compromise solution in several areas while agreeing to adopt the XP process as defined in others. This goes against Kent Beck[4], the author of XP, who advises that following the process as defined is imperative until experience is gained using it. This was not an option for this organization or the project had the compromise been rejected XP would have been rejected.

Prior to implementing XP the project had started using a BDUF process and quickly ran into trouble. The requirements were not well written, the process was creating a lot of overhead and deadlines were tight something had to be done. It was decided to toss the BDUF process and use XP. Almost immediately, however, it was attacked because of the lack of documentation

[4] http://www.cutter.com/consultants/beckk.html

required by the process. Answering this attack it was brought to light that XP was not anti-documentation. XP recognizes that documentation has a cost. Prior to doing any documentation first answer the question, "What do we want from our documentation?" Create only the documents that are required and nothing more!

This was especially difficult to contend with concerning system maintenance because, initially, this was perceived as being essential documentation. This is contrary to XP, which views the code as the documentation. It was successfully argued that since maintainers go directly to the source code to solve problems rather than to stacks of binders of text documents. It would be more productive to create well-written source code with little text documentation. If too much detail is put into the documentation two things happen: 1) It becomes very costly to maintain as the system evolves or 2) It is never maintained making it of little value to any future need. The text documentation should describe the system at a high level guiding the future maintainers to the right part of the source code.

Another problem arose around the idea of documenting each iteration's requirements on index cards. There was a large concern the cards would invariably get lost. It was agreed that UML would be used in place of index cards to document requirement's use cases.

The review process for the designs addressing the requirements turned out to be a blending of the organization's current process and one more acceptable to XP. The existing process placed the review directly on the critical path. The design had to be signed off before work could proceed. To minimize the impact this had to the project it was agreed that less critical designs would be implemented in code during the iteration following the XP process while the critical designs would go through the existing design review process.

Aside from these process deviations, most other XP principles were followed. Using the principles of paired programming, test first programming, short iterations, continuous integration and customer team member and the modified process steps the project was considered a success. After the fourth iteration the project manager was surprised and delighted to have functioning and tested code whereas for other projects, measured in the same time frame, he would've only had a stack of documents.

There was another added benefit resulting from using the XP process. The project manager discovered he did not need to spend the time and effort usually required to coordinate priorities and task dependencies. The reason for this is that the features are done in an order defined by the customer instead of some internal project schedule or software framework. The project team was agile and found it was able to adapt to the changing needs of other subsystems making themselves self coordinating.

Summary and Conclusion

Agile processes can improve both quality and time-to-market but are they processes that all organizations can effectively use? The answer to this question is probably no. There is a decided preference among managers to choose BDUF processes over agile ones. Large organizations have financial and professional investments made in BDUF processes, business models requiring fixed price contracts are mandated, and the strong history of software engineering concepts and the industry wide adoption of the Capability Maturity Model[5] as the standard defining software development practices make the adoption of agile processes into some organizations doubtful [17]. Moreover, there may always be a place for BDUF processes. Projects addressing life critical systems must have a rigor often associated with BDUF processes. With this rigor, however, comes a cost in both time and money to manage the large overhead associated with the processes. Typically, this is a cost most organizations can no longer sustain and remain competitive. Possibly, with a lot of work, the answer to the question may also be yes. Organizations may have to learn to use agile processes. It may be necessary for their survival.

This paper outlined four of the many different agile processes currently in use (Strictly speaking, however, ASD is not a process like the others). Although different in some aspects there are common threads flowing in all of them. They speak of developing code sooner, they talk about multiple iterations, the need for customer involvement, the importance of people over process and, maybe the most important, the need to adapt to change. It is imperative for organizations to wake up to the fallacy of gathering all requirements upfront and creating a rigid plan for implementing these requirements. Just doing this will be a big step towards changing current development practices.

Agile processes are not a panacea and they are not easy to implement. An important step to adopting an agile process methodology is to read and embrace the philosophy discussed by Jim Highsmith and his ASD (Advanced Software Development). Selecting a process is, of

[5] http://www.sei.cmu.edu/cmm/cmm.html

course, dependent on the organization. From one extreme, adopting XP, it is expected that the process will be followed to the letter, (although the case study didn't and still experienced a successful project). To the other extreme, using RUP where the framework is so loosely defined many different processes seem possible. A word of caution here: since RUP is so versatile it is possible that what is ultimately adopted will be called agile but, in fact, will be a BDUF process that is agile in name only. Furthermore, all players, including developers, project managers, test and Q and A departments, customers, financials, and upper management must be in total agreement with using agile processes. Without this commitment, failure is a strong possibility.

While not a panacea, agile processes can help an organization to better compete in a volatile and rapidly changing market by helping to deliver high quality software at a competitive price and in a timely manner to a market moving at blinding speed.

References:

1. Software Engineering a Practitioner's Approach; Pressman, Roger S. McGraw Hill; 2001; p 20.

2. Mass hysteria and the delusion of crowds; Kenny, Michael; itopia Technoparkstr 1 8005 Zurich Switzerland; 2001; page 7.

3. The Manifesto for Agile Software Development; http://agilemanifesto.org/; last referenced Nov 1, 2002.

4. The New Methodology; Fowler, Martin; http://www.martinfowler.com/articles/newMethodology.html; June 2002; Last referenced Nov 1, 2002

5. Limitations of Formal Methods and An Approach to Improvement; Shaoying, Liu, Adams, Rolf; Dept. of Computer Science Faculty of Information Sciences Hiroshima City University; 1995; page 5.

6. An Essential Distinction of Agile Software Development Processes Based on Systems Thinking in Software Engineering Management; Wendorff, Peter; XP 2002 Extreme Programming; page 218.

7. Agile Software Development: The Business of Innovation; Highsmith, Jim, Cockburn, Alistar; IEEE Computer; Sept 2001; page 120.

8. Messy, Exciting, and Anxiety-Ridden: Adaptive Software Development; Highsmith, Jim; American Programmer, Vol 10 no. 1; Jan 1997

9. Embracing Change with Extreme Programming; Beck, Kent; IEEE Computer; October 1999; page72.

10. Agile Software Development Methods; Abrahamsson, Pekka, Salo, Outi; VTT Publications ISBN 951-38-6009-4; Sept 2002; page 39

11. The Unified Process; Jacobson, Ivar, Booch, Grady, Rumbaugh, James; IEEE Software; May/Jun 1999; page 97

12. The Process; Booch, Grady, Martin, C. Robert, Newkirk, James; Addison Wesley Longman Inc.; 1998; page 99

13. XP Transition Roadmap; Denver Systems Operations: TRW Incorporated; Eckman III, C. Emmet; 2002; page 222.

14. Introducing an Agile Process to an Organization; Cohn, Mike, Ford, Doris; Mountain Goat Software and Precision Projects; 2002.

15. Limitations of Agile Software Processes; Turk, Dan France, Robert Rumpe; Colorado State University, Fort Collins, Colorado; 2002; page 43

16. Using XP in a Big Process Company: A Report From the Field; Grenning, James; Object Mentor, Incorporated; 2000

17. The Decision is in: Agile Versus Heavy Methodologies; Charette, Robet; Cutter Consortium Vol. 2 No. 19 http://www.cutter.com/freestuff/epmu0119.html; last accessed Nov 23, 2002.

Bibliography

Extreme Programming Explained First Edition; Beck, Kent; Addison-Wesley Publishing Co.; 1999

Adaptive Software Development: A Collaborative Approach to Managing Complex Systems; Highsmith, James; Dorset House; January 2000

Webliography

Agile Alliance; http://www.agilealliance.org

Object Management Group; http://www.omg.org

Crystal Methodologies; http://crystalmethodologies.org/