Lean and Agile Software Development

A Case Study

By

Collin Murray

Submitted to the System Design and Management (SDM) department

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Management and Engineering

at the

Massachusetts Institute of Technology

February 2008

Signature of Author: .....

Collin Murray
System Design and Management

Certified by: .....

Thesis Supervisor: Prof. Michael Cusumano
Sloan Management Review Distinguished Professor of Management, MIT Sloan School of Management

Certified by: .....

Pat Hale
Director, System Design and Management

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
MAY 07 2008
LIBRARIES

ARCHIVES

Table of Contents

Lean and Agile Software Development

A Case Study

By

Collin Murray

Submitted to the System Design and Management (SDM) department
On January 18, 2008 in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Management and Engineering

Abstract

This paper looks at agile and lean development transitions for organizations that formerly used the waterfall style of development. There has been lots written about the positive aspects of agile software development and the anticipated benefits are widely touted. Through my research I became aware of significant obstacles that organizations can encounter when adopting an agile development method. The obstacles seem to be more applicable to organizations that use the waterfall development method and are compounded when legacy products exist. The intent of this thesis is to identify positive and challenging aspects for organizations that undertake a transition from waterfall development to agile development.

Thesis Supervisor: Michael Cusumano
Title: Sloan Management Review Distinguished Professor of Management

Dedication

This thesis is dedicated to my family.

My wife Kelly, whose love, patience, encouragement, and support enabled me to undertake this academic endeavor. Without her unfailing support I would not have been able to pursue my educational desires. Thanks Kelly – I love you!

My children, Jackie and Collin, who also provided endless love, support and encouragement. My hope is that they'll be inspired that 'it's never too late to follow your dreams' (and no, I'm not the oldest student here...).

My father, whose love and words of wisdom always guide and help me.

Acknowledgements

There are numerous people who have assisted me with this thesis by participating in the survey, responding to my inquiries, and allowing me to interview them. The names of all the people who lent assistance are too numerous to list, but I would be remiss if I did not mention the following individuals and teams who generously provided their time and valuable insights:

Professor Michael Cusumano, Russ Holden, Chip Carter, Daniel LeFebvre, Mike Barcomb, Margaret Rora, Nancy Van Schooenderwoert, Peter George, Charlie DeWitt, Al Eldridge, Sue McKinney, and Bill Krebs

In addition, I would like to acknowledge the SDM program and my fellow classmates. I enjoyed the time learning, and learned from both professors and students. Finally, I must acknowledge IBM and their sponsorship for the program, particularly: Kevin Cavanaugh, Norm Lord, Bill Hume, and John Woods.

1. Introduction

This paper looks at agile and lean development transitions for organizations that formerly used the waterfall style of development. There has been lots written about the positive aspects of agile software development and the anticipated benefits are widely touted. Through my research I became aware of significant obstacles that organizations can encounter when adopting an agile development method. The obstacles seem to be more applicable to organizations that use the waterfall development method and are compounded when legacy products exist. In order to present a complete picture for the reader, a brief history of waterfall, agile, and lean development methods is provided, followed by analysis of two organization transitions, one that has undergone a transition to agile, and another one that is currently in the process of transitioning. The analysis looks at a brief background of the organizations, their product(s), why the transition process was initiated, goals of the organization, status and results from the transition, and key findings. Finally, conclusions are drawn from these experiences and other readings (particularly Agile Conference proceedings) and are presented along with some questions that managers should consider prior to undertaking the transition.

1.1 Definitions and Background on development methods

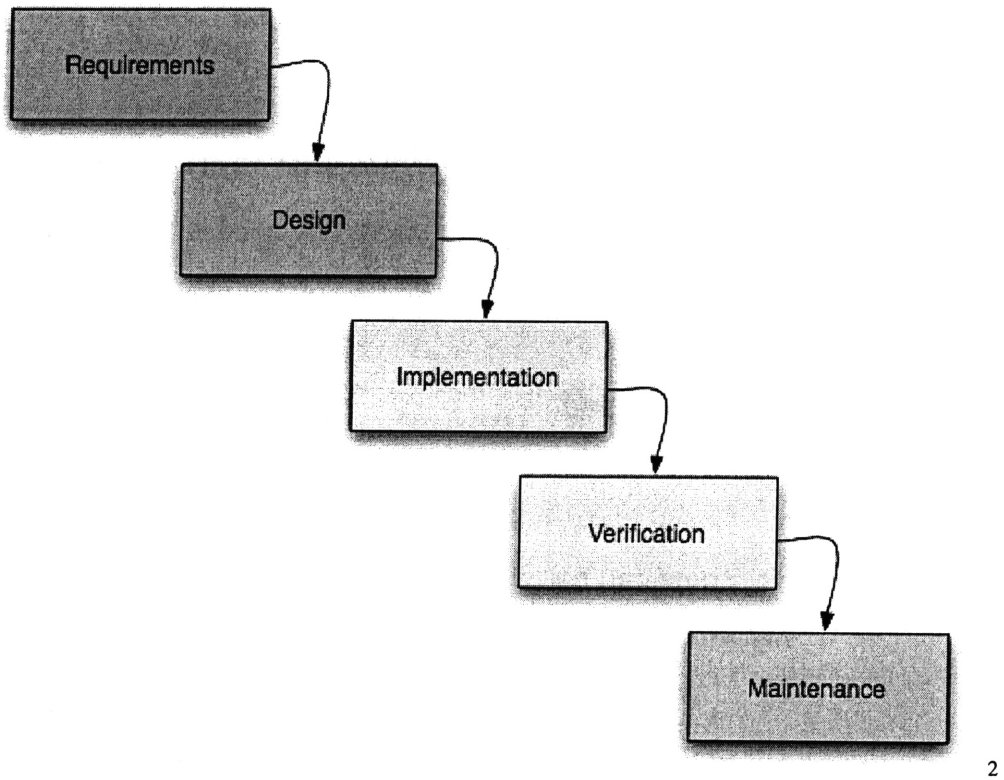
In order to understand the aspects of the development methods discussed in this paper, a brief definition and history of each method is provided.

1.1.1 Waterfall Development Method

Waterfall
Nothing can harm me at all
My worries seem so very small
With my waterfall
Jimi Hendrix, "May This Be Love", 1967

The waterfall method uses a sequential approach to developing software, where each step of the development process follows the prior one. This method typically has a large, up front investment to determine requirements for the software product. Following requirements determination, there are various phases including the coding (implementation) and testing (verification) phases. The testing phase of waterfall development methods typically involves a substantial amount of time once the coding has been completed.

The term waterfall is attributed to Winston Royce, through his paper "Managing the Development of Large Software Systems"[1], published in 1970. The misinterpretation of Royce's model led to the standard accepted practice of waterfall development that is used today. Royce was attempting to emphasize a more iterative method of software development, but the sequential approach was what 'stuck', not the aspects of iterating between stages. A depiction of the waterfall method showing the process is shown in Figure 1 below. The verification phase includes the large testing effort.



2

Figure 1

The waterfall development method garnered further adoption via DOD STD 2167. This standard helped establish the waterfall method as a standard not only in the defense community, but in other areas of software development as well. The proponent of STD 2167 (David Maibor) later discussed the reason for recommendation and expressed that with different information, he would have made a different recommendation:

> *"The US norm DoD-STD-2167 favours the waterfall model, other norms (CMMI, V-Model) followed. But David Maibor (author of DOD-STD-2167) wasn't familiar with timebox driven iterative development and evolutionary requirements and with hindsight he would have recommended it more explicitly as he does within STD-2167!"*[3]

1.1.2 Agile Development Method

The terms agile development and iterative development are often used interchangeably. While agile development practices in name are relatively recent, iterative and incremental development (IID) practices have a longer history. As will be mentioned later in the case studies, some of the development practices that were utilized may not have referred to a practice as agile or iterative and incremental by name, but were precursors to newer software development methods.

A brief history of the evolution of agile development practices is presented to allow the reader to gain a perspective on the dynamics of the software development industry and to underscore the need to remain current, or at least aware of current and competitor's practices. As new methods and systems are introduced it is important to determine which ones may help organizations be more effective and successful in the market. One of the earliest documented practices of incremental development was for Project Mercury, the United States of America's first manned spacecraft. The software development effort was undertaken by the IBM Federal Systems Division using incremental development in 1958. Gerald M. Weinberg, one of the workers on Project Mercury discussed their use of iterative development:

> *"... we had our own machine and the new Share Operating System, whose symbolic modification and assembly allowed us to build the system incrementally, which we did, with great success. Project Mercury was the seed bed out of which grew the IBM Federal Systems Division. Thus, that division started with a history and tradition of incremental development."*[4]

Although much could be written about the history and complete evolution of agile and iterative development, there were subsequent notable documents and attempts at using iterative development methods that warrant discussion. Chronologically, Royce's paper on development for large software systems in 1970 appeared first. As mentioned above, Royce's paper was misinterpreted as a purely sequential series of steps, leading to the belief that he was a proponent of the waterfall method. In reality, Royce was proposing a type of iterative development model. In 1975, Victor Basili and Albert Turner published a paper on IID entitled "Iterative Enhancement: A Practical Technique for Software Development". Another landmark paper concerning the evolution of software development was Barry Boehm's paper "A Spiral Model of Software Development and Enhancement published in 1988. Both of these papers touted the benefits of iterative development compared to waterfall development and are noteworthy for their contribution to pre-agile development methods.

The Basili and Turner paper proposes an iterative development model that starts out with a skeletal implementation of what the requirement is believed to be. This is because the problem or requirement may not be completely understood (this aspect is discussed later in this paper). As a result, enhancements and modifications can be made to the initial implementation during successive iterations. During each iteration cycle, analysis is performed to see how close the feature is to what the requirement really is. This process continues as additional changes are needed to complete the project, referred to in the paper as "iterative enhancement"[5].

Barry Boehm's paper on the spiral method of software development illustrated how software projects could succeed using risk evaluation during each iteration cycle. Similar to Basili and Turner's paper, a prototype can result after each cycle through the spiral. The spiral method also allows the flexibility of using various software development

methods within the project development cycle depending on the risk posed through each spiral loop. Lending credibility to the paper was the results achieved via the spiral method's implementation at TRW, including avoidance of risks that may have otherwise contributed to longer development cycles and higher costs, software reuse, and an improvement in productivity[6].

Agile software development entails breaking a software project into smaller modules that can be accomplished during an iteration cycle. The iterations run a fixed timeframe, and requirements are pulled into an iteration cycle by priority of the requirement. Unlike the waterfall method, agile development is receptive to requirement changes and doesn't perform a large, initial planning effort. In addition, organizations can find out early on whether or not the project is what the customers want. Finding out early provides development organizations the opportunity to address problems and get further feedback from customers. With the waterfall development method, organizations typically don't receive feedback until the product is close to completion, giving it little opportunity to improve the code, and also increasing the risk of making further changes.

The popularity of agile software development has risen over the past several years. There has been much written about agile software development in published books and online content. According to a recent Dr. Dobbs survey, around two thirds of software development companies have implemented one or more agile development methods: "agile development has crossed the chasm"[7]. There are various agile development methods employed by software development organizations, such as Extreme Programming (XP), Scrum, Feature Driven Development, Crystal, Test Driven Development, and DSDM. Each method has its own benefits, and some work well with other methods (such as Extreme Programming and Scrum).

Several early leaders involved with various (now known as agile) development methods convened in February of 2001 to discuss alternative development methods from the traditional big up front planning and heavily documented method. What arose out of the conference was 'agile development'. The principles of what agile development should be

were noted and are now known as 'The Agile Manifesto' (see Appendix 1 for the list of principles).

There have been an increasing number of articles written concerning agile software development. A search of the Engineering Village database for Compendex, Inspec, and NTIS shows the trend of agile software development through published articles with the result set of the query "agile software" (see Figure 2 below).

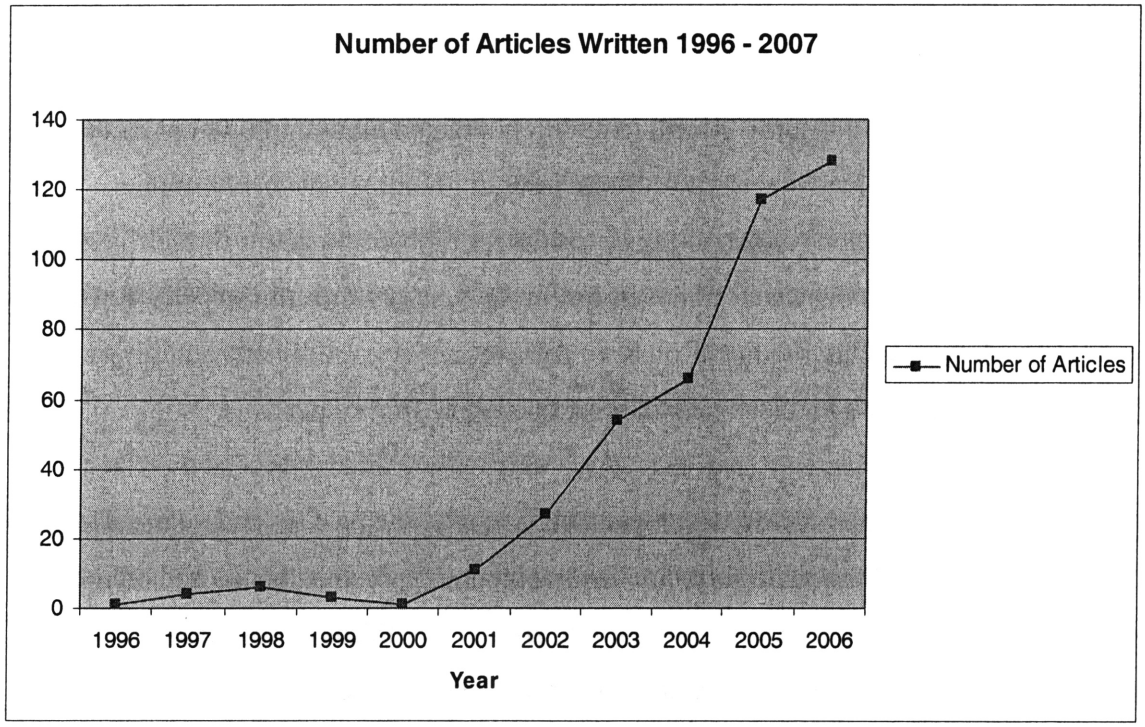


Figure 2

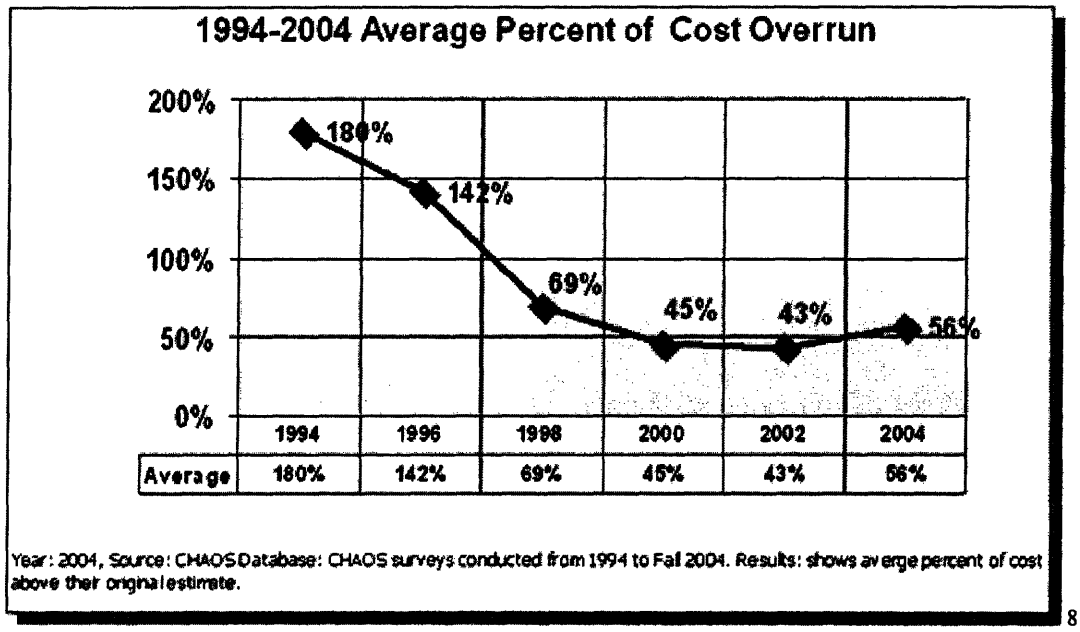
Historically, many software development projects have resulted in taking longer than expected, costing more than budgeted, and not meeting customer requirements. Through my observations of the organization case studies and some of the agile literature, it appears that agile development methods may be of particular help in addressing the following aspects of software projects (particularly for projects that are not fixed in scope and duration):

- software projects typically go over budget

- software projects typically run longer than planned
- software requirements churn
- many software features are unused
- high quality software

Over budget

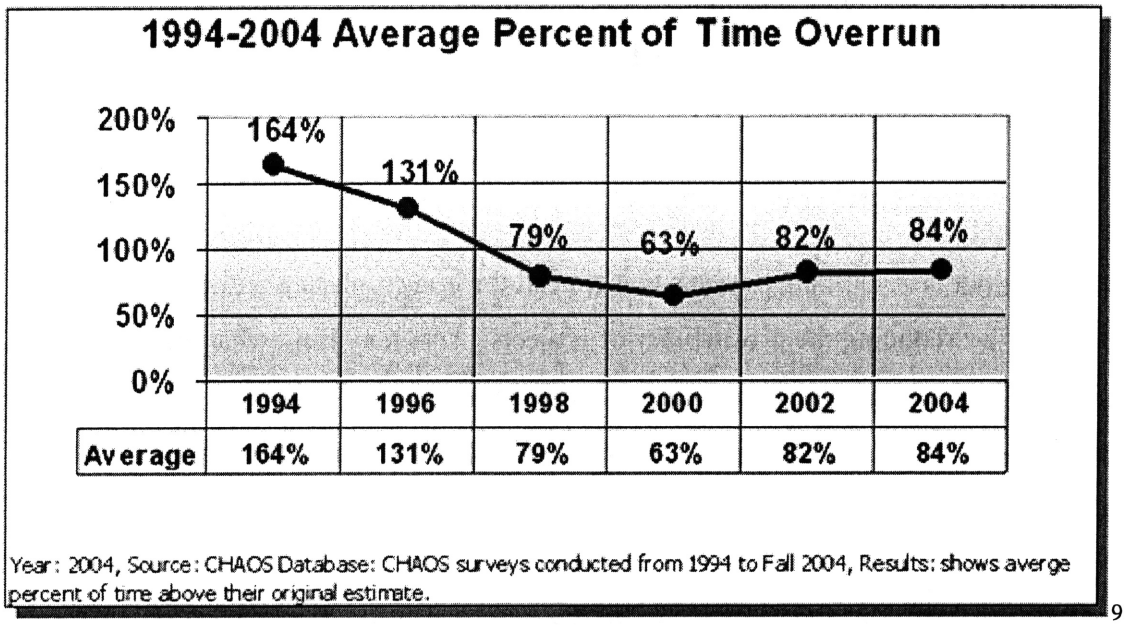
Software product development projects that exceed the budget place a burden on organizations by reducing the profitability of projects. This may limit what organizations can do in the future if development expenses become too large. A major reason for software projects going over budget is due to the next reason, that software projects typically take longer than planned.



8

Time overrun

As shown in the chart below, a significant portion of software projects fail to meet their planned completion dates. This results in additional incurred costs to the development organizations and may also present competitive disadvantages such as missing opportunities to compete with other products in a timely manner, and hence losing potential sales revenue.



9

Software requirements churn

Customer requirements change over time, and in some cases customers don't really know what they want. An important reason to get customers involved with the development process early on is due to "Humphrey's requirements uncertainty principle":

> *"For a new software system, the requirements will not be completely known until after the users have used it."*[10]

It is not uncommon that there is a significant churn in requirements for software projects. Microsoft has reported a 30 percent change of features within product specification documents, and an additional deletion of 20 to 25 percent of features originally included as part of the product specification[11]. Craig Larman, an agile practitioner and author of several software development books, mentions that very large software projects have churn rates of 35% or more[12]. In other cases, there are reports of over 50% of software requirements changing during the product development cycle[13].

Unused software features

It is estimated that only 20% of software features are used regularly, and 64% of software features are rarely or never used[14]. The reasons for this are due to:

- customer requirements change
- misunderstanding of customer requirements
- poor quality of implemented code

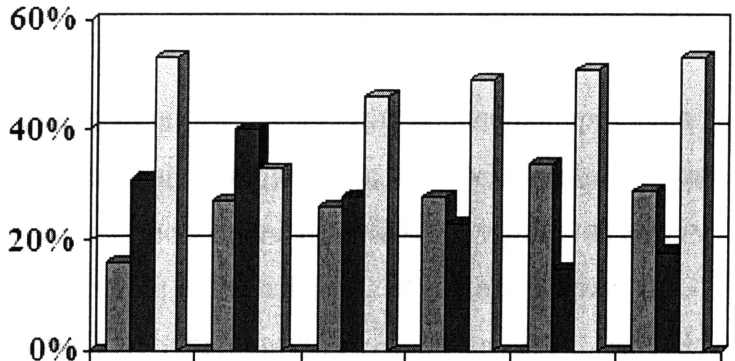
As mentioned above, customer requirements often change during product development. As a result, features that are implemented without adjusting to the updated needs of customers may find themselves not needed. Misunderstanding of customer requirements also contributes to unused software features, as features that are implemented may not meet the customer's actual needs. Poor quality of code written for new features will also affect the ability for customers to use the feature. If the code is too buggy, then customers will likely refrain from using the feature set rather than put up with the side effects that it brings.

High quality software

Software with poor quality affects the ability for companies to sell the software, thus limiting the revenue and profitability of companies. Further, poor quality software increases the costs for developing organizations. The costs increase due to the need to provide support for the product as well as issue updates to address defects within the product. This adds to infrastructure costs by having a larger support staff and the requirement of a distribution system to provide patches or Service Packs to remedy problems within the product.

The Standish Group's list of the 'Top Ten Reasons for Success' for software projects lists 'Agile Process' as the number five reason.[15] As seen in the chart below, the percentage of projects that fail has largely been on a downward trend, presumably due in part to the proliferation of agile development adoption. The chart results are based on over 9,000 IT projects surveyed, 58% within the United States, 27% within Europe, and the rest of the world accounting for 15% of the projects surveyed[16]. Failed projects are those that were

cancelled, or finished but not used. Challenged projects are projects that finished, but exceeded the planned budget and/or time allocated to the project. It is interesting to note however, that there are larger percentages of projects that fail or are challenged as the size of the project increases (measured in scope and time).



	1994	1996	1998	2000	2002	2004
Succeeded	16%	27%	26%	28%	34%	29%
Failed	31%	40%	28%	23%	15%	18%
Challenged	53%	33%	46%	49%	51%	53%

[17]

Source: The Standish Group, 2004

Agile development helps curb the '90% complete' aspects and the rework associated with projects that result in late completion. System dynamics models have shown that rework, particularly rework that is discovered late in the development process can have a significant impact on project completion times[18].

> *"Rework is the most important single factor driving schedule and budget overruns. Most management reporting systems overestimate real progress and discourage reporting of rework." - James M. Lyneis*[19]

Agile development principles focus on completion of work prior to moving on to the next requirement or feature, thus preventing late rework from being a contributor to project delays. While rework may still be required during iterations if problems are found during

testing, the compounded effect of having to address defects later, and possibly defects built on top of defects, is eliminated.

1.1.3 Lean Development Method

A relative newcomer to the agile development world, lean software development takes its roots from Toyota's Production System. Mary and Tom Poppendieck have written about lean software development and have adapted the principles of lean development to lean software development[20]. The seven theoretical principles for software development are a translation from lean manufacturing methods utilized by automotive companies, intended to provide value for customers by providing products and features that they want while reducing waste for the developing organization. These principles are:

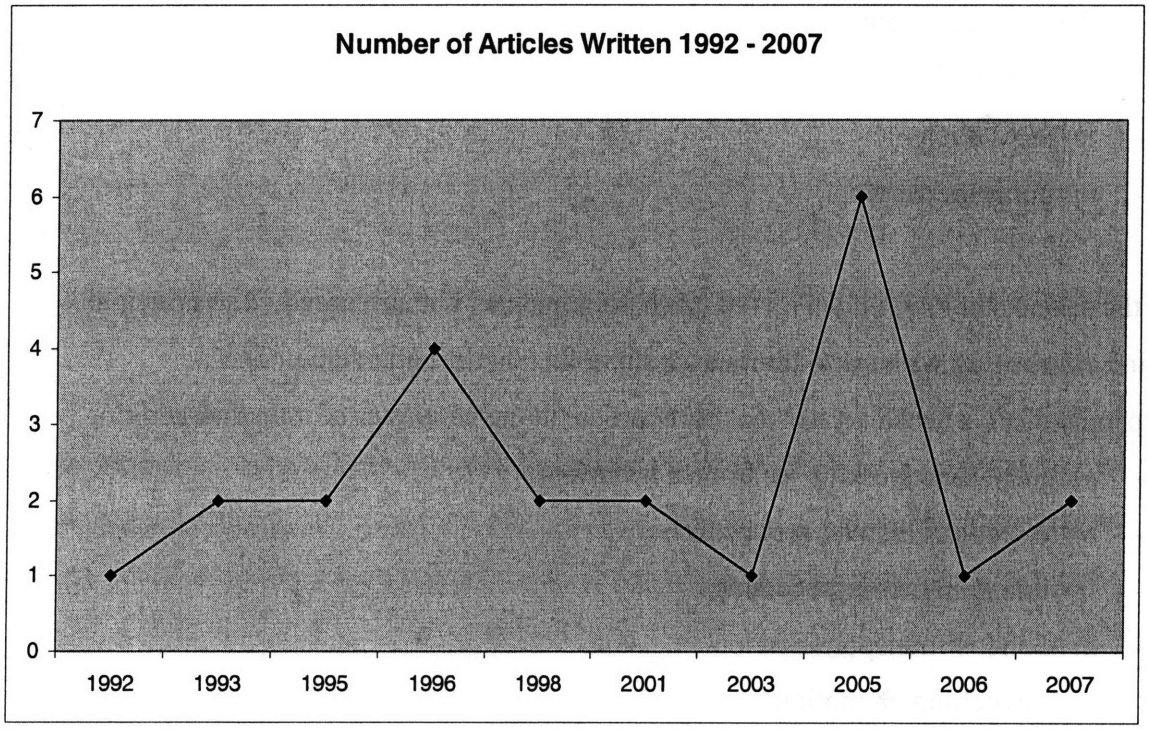
- eliminate waste
- build quality in
- create knowledge
- defer commitment
- deliver fast
- respect people
- optimize the whole

For the sake of brevity, only a few of these principles will be discussed. The principle behind eliminating waste is to remove anything that doesn't add value. The Poppendieck's list seven wastes that correlate to the seven wastes of manufacturing[21]:

- partially done work → in-process inventory
- extra features → over production
- relearning → extra processing
- handoffs → transportation
- task switching → motion
- delays → waiting
- defects → defects

Lean software development addresses these wastes by focusing on completing features during each iteration, developing what the customers want (and no more), and ensuring that testing is performed for each iteration, and high quality code results. Building quality in the product sets the goal of eliminating the need to track bugs. Deferring commitment focuses on the ability of the organization and individuals to make the best possible decisions, waiting until the last moment to get enough information to make the decision (particularly since requirements may change, and not all information is known up front).

Lean software development is a newer introduction to software development and has not received as much attention as other agile development methods such as XP and Scrum, but has gained some visibility recently. The following chart shows the number of articles on "lean software" from Engineering Village:



Of note, Niklaus Wirth, the designer of the Pascal language and winner of the ACM Turing Award has one of the earliest references to lean software development in a 1995 published paper, "A Plea for Lean Software"[22]. While this paper doesn't apply the

concepts of Toyota's lean principles, it covers some familiar ground such as unused features and code bloat, both sources of waste.

2. Case Studies

What drives an organization to undertake a fundamental change in development methods? For successful products it appears to be much easier for an organization to stick with the status quo rather than implement changes. For larger development organizations that have been entrenched in their development methods for many years, the cultural implications can be substantial. While there are benefits touted for those that successfully navigate the change (high product quality, features that customers value, and time to market), there are costs associated with the transition. Therefore, it is necessary to understand the costs before tackling this change. This section examines the transition from the waterfall development style to agile software development methods within two software organizations: IBM and Kronos.

2.1 IBM – The Lotus Domino development organization

2.1.1 Introduction

This case study looks at the early stages of the Lotus Domino organization's development transition from the traditional waterfall method to an agile method. The observations of the transition encompass positive aspects of an agile method and some drawbacks or challenges of switching to an agile method. Some of the challenges were anticipated while other challenges were unexpected, resulting in the need to constantly evaluate and adapt processes as needed.

Lotus and IBM have traditionally developed software using the waterfall based model. The notable differences between development practices within Iris Associates (the development team for Notes under the Lotus Development Corporation) and IBM were tools and scaling requirements. As an early start-up, Iris consisted of six individuals developing the code for what was to be Lotus Notes 1.0. The development model was waterfall based, consisting of setting requirements up front, then proceeding to designing, coding, and then testing the product[23].

As a start up organization, the tools used to develop Notes were largely developed internally, source control was maintained by one individual who handled discrepancies and merging, and the build process was slow. In the early stages of developing version 1.0, the build process would take over one day to complete for around 180,000 lines of code! A large part of the time spent building was due to the performance of the computer processors available at the time (this was around 1988). As a result of the time required to build the product, build weeks would turn into nightmares, sometimes requiring three weeks to declare a 'master build'. The master build could then be made available to those working on the product. During this time, developers would be doing their work on a side build and have to synchronize with the master build. As expected, there were source code conflicts that needed to be handled once the master build was ready to be built again. There was flexibility in what could be worked on however, as the project

was a new product and user needs could not be completely determined since there was no comparable product (and users don't always know what they want until they see it). Scaling changes were required due to growth of both the development (and eventually testing) organization and the size of the code base. More management and structure was needed as the number of people in the organization grew, a departure from the initial structure averse mentality that was a part of the Iris culture. In a sense, the lack of formal structure could make things a bit more agile, but the planning, coding, and testing efforts were very much in line with the waterfall model. Development methods also evolved as the organization grew, such as tighter source control and dedicated build resources. A dedicated build room was instituted around version 3 of Lotus Notes.

Another notable difference between Iris/Lotus and IBM development methods concerned the way testing was handled. For the first few release of Notes, testing was performed by a separate organization within Lotus. Eventually there was an integration of the development and testing teams. This was not due to the acquisition of Lotus by IBM (in 1995), but rather the recognition of the need to have better testing by having the two groups closer together. A time box approach to developing and releasing versions of Notes and Domino was attempted in the mid-1990's with the goal of releasing products at a pre-determined interval. If the deadline was approaching and certain features weren't complete, those features were to be put into the next release of the product to allow a quality product to go out the door. This time box approach didn't have quite the effect that was intended, as product release dates still slipped a bit to allow time for features to be completed prior to public release. In these cases, the quality and value that the product offered still outweighed the emphasis on a set delivery time.

Within the larger Software Group division at IBM, approximately 25% of software products use an agile development approach. Mirroring the results reflected in the Dr. Dobbs survey, around two thirds of software organizations within IBM are using one or more agile practices. Agile development was chosen as the right approach to move towards for software development projects to improve the effectiveness of the engineering efforts. It was decided that agile development adoption would not be made a

mandate, to make the transition a bit less formal and avoid the feeling that results of the transition would be "pass/fail". To assist in the migration, some projects had been utilizing agile development methods at the engineering level for several years and could be used as examples of how improvements in product quality, product costs, and employee satisfaction could benefit. During the first full year of encouraging groups to transition to agile development, the goal of having 30 agile development projects was far exceeded, with the result of 110 development projects adopting agile development. Additionally, the spread of agile development adoption exceeded expectations of the number of agile leaders and mentors that would be in place within the division.

The types of projects that have implemented or are implementing agile software development methods range from PC, UNIX, LINUX, and web based applications to internal applications, to mainframe applications and operating systems. Examples of PC based applications include Lotus SameTime and Tivoli Workload Scheduler. For internal applications, the IBM website (ibm.com) has adopted agile practices, notably Extreme Programming (XP). Larger hardware system and mainframes have also been participating in the agile development transition with applications such as CICS Transaction Server making the transition to agile development practices. The types of applications adopting agile development practices run the gamut from small to large in terms of both size of the applications and the hardware they reside on. As the agile development initiative is not a mandate, and is relatively recent in terms of the lifetime of some applications, some applications will likely take longer to adopt agile development practices. The larger system applications, while largely adopting agile practices after other groups have started, have been steadily picking up the pace of agile development adoption.

From a divisional perspective, the largest visible benefit to date has been the consistency and reliability of delivering quality software products. The peaks and valleys associated with some past software projects has been smoothened, avoiding the need to add more people to work on projects that are running late. This avoids encountering the situation described by Fred Brooks as "Adding manpower to a late software project makes it later."[24]

2.1.2 Background

The Lotus Domino development organization is one of several software organizations within IBM, and develops Lotus Domino, a successful enterprise software product that has been in the marketplace for over fifteen years. While some of the development team members have changed over time, there was a firmly established culture using the waterfall method. The typical process involved setting product requirements up front, some generated directly from customers, and others determined by employees (developers and product managers, for example) who attempted to gauge the customer's needs indirectly. Once requirements were determined, specification documents would be written with varying detail of the intended functionality. The specification documents would contain time estimates for completion, and it was not uncommon to have a team work on several features at once, with some features requiring many weeks of development time. Developers would proceed to code the features/requirements, and once complete would hand over to the test team (QE, for Quality Engineering). Testing efforts would involve unit testing the particular features using 'black-box' and 'white-box' methods. The last portion of the product development cycle, usually 3 ½ to 4 months, would be dedicated to final pass testing, including a regression testing phase.

Over time, the code base and complexity of the underlying code grew significantly. From the initial version of the product consisting of 500,000 lines of code, the size of the code has grown to over 22,000,000 lines of code for the current release (see Figure 3 below). The dedicated maintenance development team grew from 10 dedicated people in 1997 to 40 people in 2007. The code base also quadrupled during the same period, from 5 million lines of code to over 20 million lines of code. While the relationship appears to be linear, the number of additional people working on maintenance was actually higher than the 40 people dedicated for maintenance. On average, it was not uncommon to have feature developers spend 15% of their time working on maintenance related issues. The increase in people maintaining the existing code may reflect, in part, the increasing complexity of the code base over time.

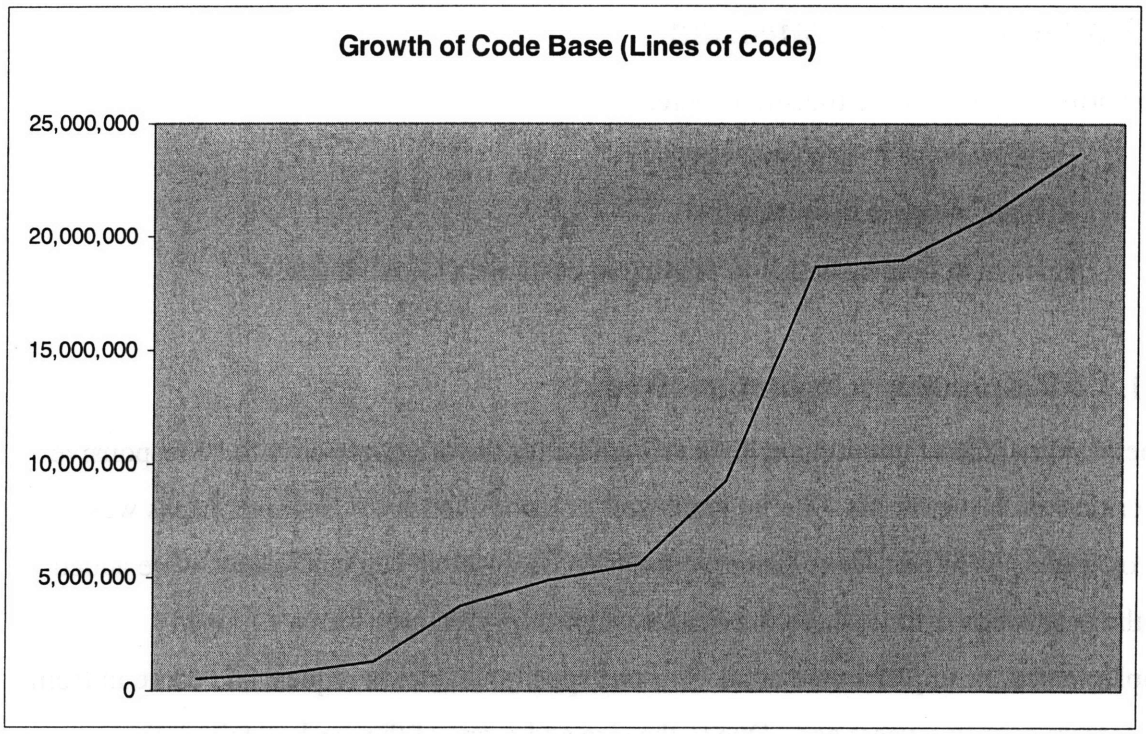


Figure 3

The average time to market for each major release was around 1.5 years, comparable to its nearest competitor. Overall, product quality was good, but there was concern that the organization wasn't responsive enough to customer needs, and that some of the features added to the software didn't meet customer needs.

In order to meet these concerns, the Domino architect proposed the adoption of agile development methods. After discussion with management and senior technical personnel, the organization decided to pursue an agile development approach. There were resources available within the company to provide guidance on how to get started, along with executive level support for implementing agile development practices. It became evident from discussions with team members that having available resources to learn about agile development was important, both for learning, and as a proof of concept. Executive level support was also important due to resistance encountered during the adoption phase.

2.1.3 Goals of the transition

The primary goals of the transition were:

- Be responsive to customer needs
- Reduce waste in the product
- Have high quality builds to provide to customers for feedback

2.1.3.1 Responsive to customer needs

The primary goal of adopting agile development for the organization is to be responsive to critical customer needs. The organization's responsiveness to customer needs was inconsistent. While many customer requests for features and enhancements were added to the product over time, there were cases where some features weren't properly implemented, or implemented at all. For one specific feature, a request had come in from customers over two years ago. Due to the size and scope of the work required, this requirement was put aside in favor of other projects that could be completed, largely due to resource constraints. With the prioritization process now being used, this feature is targeted to be included in the current project.

2.1.3.2 Reduce waste in the product

As shown above, the amount of code in the product grew significantly over time. As new code gets added to the product, it adds to the complexity of the product. This becomes more of an issue when the product is maintained, as changing the code later can be tricky for a couple of reasons. First, understanding the intent of the code can be difficult to decipher, and second, making changes to any code may result in an undesirable change of behavior (even if the original behavior wasn't intended). In order to minimize the complexity of the software, only the necessary code must be written. This means writing features that customers need and value. Jeff Sutherland, CTO of PatientKeeper refers to the long term costs of maintaining software with a (slightly) tongue-in-cheek perspective:

> *"The task then is to refine the code base to better meet customer need. If that is not clear, the programmers should not write a line of code. Every*

line of code costs money to write and more money to support. It is better for the developers to be surfing than writing code that won't be needed. If they write code that ultimately is not used, I will be paying for that code for the life of the system, which is typically longer than my professional life. If they went surfing, they would have fun, and I would have a less expensive system and fewer headaches to maintain.[25]"

Another source of waste that the agile transition hoped to address was that of incomplete features. The amount of time and money spent developing a feature that was not complete or didn't meet customer requirements were wasted resources, as customers were unable to use those features.

2.1.3.3 High quality builds

During prior development cycles, there would typically be two or three Beta releases of Lotus Domino provided to customers. Prior to releasing a Beta version of the product, there would be a change in focus for the development team. First, new feature development would slow down to avoid putting in new features that would not be completed by the deadline. Second, there would be a significant effort by both the QE and development teams to address outstanding bugs within the product. Successful adoption of agile development would allow the organization to focus on continuous development and have more frequent Beta quality builds available for customers.

2.1.4 Transition

This section looks at how the development organization undertook the transition, from the initial kickoff meeting to the present period.

2.1.4.1 Kickoff

After hearing about agile development and the benefits it offered, two senior individuals in the organization attended lean and agile development training presented by Mary and Tom Poppendieck at IBM headquarters in Armonk, New York. The intent was to better

understand agile and lean development and share the knowledge with the larger development team.

In order to communicate the agile and lean development message, all employees within the organization were required to attend a meeting to discuss the development method change, and the goals it expected to reach. The meeting was jointly chaired by an executive and a senior technical leader within the organization. A condensed version of the training received in Armonk was then provided to the organization's team members along with information on other resources available. A company-wide agile wiki was also available, and referenced further resources both within and external to the company. The IBM agile wiki was first created in 2003, and served as a reference for other software development organizations within IBM.

Executive support within the team and from higher levels of the division was made visible to encourage widespread agile development adoption. As the transition took place, periodic follow up meetings were held with the entire organization to address concerns and provide updates on the transition progress. It should be noted that there was no official Scrum training provided to the organization, but rather the leaders took on the task of understanding Scrum as a team. The terms 'agile' and 'lean' software development are used interchangeably within the organization.

2.1.4.2 Agile implementation

The agile plan that was implemented prioritized requirements and allowed for the top 25 to be worked for the next product version. Twenty teams were created to implement and test the new features, with each team having an iteration of three to five weeks. Each team had short daily meetings to determine the current status of the work, the plan for the day's activities, and what obstacles were present.

A list of the top 25 prioritized requirements is kept in a database that is visible to all team members. A team of senior individuals meets regularly to review the list of requirements. New requirements can be discussed at the meeting, but individuals need to understand the

importance of the current requirements before nominating one. Since resources are limited, adding a new requirement means removing an existing one from the queue. Requirements are not really incorporated via a true 'pull' system (however, the management team recognizes the need to move towards a pull system). There are several factors that are used to determine what requirements will be included in the product. The requirements are determined by a committee consisting of management and technical personnel. Part of the criteria used, in addition to priority of the requirements, is the resource availability on the development and test teams. If the available resources can't meet a higher priority requirement at the moment, then a lower priority requirement will be chosen. The individual teams do decide the content of the work performed during the iterations however.

An 18 month period was selected for completion of the new project. In order to secure agreement on the timeframe and new content of the product, it was necessary to use 'soft words' to avoid specifying exact content. This was done to allow the development team to develop the code for requirements that weren't all known up front. Although there was a list of customer requirements that already existed, there may be some more urgent requests that could come in and supersede some of those. Another reason why this was done, was to allow time for upper management to gain a high level picture of what development was doing while they mentally adjusted to agile development practices. A bottoms-up commitment couldn't be done on a feature by feature basis (after all, it contradicts the agile philosophy), but the timeframe was settled to meet business needs for both the organization and their customers. A new approach was used, where a 'ballpark' layout of the anticipated features and time required to implement them was generated to help ensure that the requirements fit within the 18 month window.

There were several impediments recognized that would limit the organization's ability to completely or successfully adopt agile development. These are discussed in greater detail below (see Transition challenges), but the management team recognized the need to approach and address the largest impediment first. Once that impediment was addressed, the next one could be tackled, etc.

2.1.4.3 Iteration schedule

The iteration schedules range from three to five weeks for the 20 teams. Since not all of the teams have the same schedule, this creates some coordination issues for the planned monthly 'code drops' (full builds of the product that were available for testing). The iteration schedules were met the majority of the time (77% of the time, according to the survey), while 17% of the time the iteration schedule was seldom or never met. While the expectation is that iteration schedules must be met, a lower adherence is deemed tolerable since the process is only 6 months old, and an adjustment period is expected. Another aspect for not having complete adherence is the thrashing, or context switching that occurs for team members largely due to customer issues that arise due to the product being a legacy product.

Lack of completion of a feature before moving to the next feature occurred on a regular basis less than 10% of the time. The majority of the time (67%) however, features were completed prior to moving on to the next iteration. Occasional cases where new features were taken on prior to completion of the current feature happened 22% of the time according to the survey.

2.1.4.4 Reflection

After each iteration period, a reflection process was used to determine how the iteration went, and what could be improved for the next time. The intent was to make changes as needed to continually improve the process. This process appears to be working well, as half of the survey respondents reported that the team meetings are 'Very effective' at achieving goals, and a third of the respondents reported that the team meetings are 'Somewhat effective'. 17% of the respondents reported that the team meetings were 'Not effective'.

2.1.4.5 Agile methods used

Scrum, Lean and test driven development were the main agile methods used in the organization. The key components of Scrum are: self-directed teams, daily meetings, set

iteration cycles, and adapting to customer needs after each iteration cycle. Within the Domino team, each team held daily meetings, following the format:

- What did you do yesterday?
- What will you do today?
- What is in the way of getting things done?

The meetings were typically short, with one half hour duration.

2.1.5 Transition challenges

There were several significant obstacles that presented a challenge going into the transition.

2.1.5.1 Organizational mindset

The first challenge was adjusting the mindset of the organization to working agile and understanding the benefits it offers. Direct participant's perceptions about the transition can vary depending on the role played. Agile development introduces a whole new dynamic of welcoming changes late in the game if they are of high enough priority to provide value to customers. These types of changes are unwelcome in the waterfall model due to the schedule and planning disruptions they cause. With an agile approach however, satisfying customers is built into the mode of operation. Lower priority tasks may be dropped to make room for new, higher priority tasks, conflicting with long term planning.

In addition, with lean software development a greater focus is put on team efforts. While a portion of individual's annual ratings incorporated teaming aspects, more teaming is required for agile and lean development projects to succeed.

2.1.5.2 Technical debt

Being a large legacy software product, there was a significant amount of technical debt (the backlog of bugs, or product defects in the product) that existed. Thrashing was also common, as both developers and QE individuals were intermittently pulled off of current

tasks to work on other tasks and then revert back to their prior tasks. A major cause of the thrashing was due to the technical debt of the product, as critical customer problems needed to be resolved immediately, forcing developers to switch tasks midstream.

Lack of automation testing is the largest obstacle for making a successful transition. In order for the development process to be fully agile, it is necessary to be able to perform frequent testing. The most efficient, effective, and consistent way to achieve this is through automated testing. For the legacy code, the amount of automation testing coverage was around 15%. This requires that extensive manual testing be performed to verify the quality of the product.

2.1.5.3 Existing waste

It is very difficult to gauge what features of a product are used, and for what reasons, as this application resides within a customer's environment (as opposed to being in a hosted environment which may allow for easier tracking of feature use). The sources of information that can be used to determine what existing features are used are: discussions with customers, internal experience with the product features (which ones do the organization use, which ones work) and reports of defects for features. If unused features were able to be clearly identified then it would allow for easy decisions to be made on what code could be removed from the product.

2.1.5.4 Dependencies on third party code

Dependencies on third party products, and code developed in other non-agile organizations of the company. Being agile means being able to respond to customer requirements as their needs arise. Having dependencies on non-agile organizations or software may impact how successfully the project will meet the customer requirements.

2.1.5.5 Lack of dedicated agile coaches

While there were ample resources that could be used within the larger division, and training was provided to the organization, there were no dedicated agile coaches to assist with the transition.

2.1.6 Positive transition factors

There were several positive factors that allowed the organization to have an easier adoption of the agile development process.

2.1.6.1 Capable technical leadership

The organization had strong technical personnel within existing teams and in leadership roles. The group of individuals had been a large part of the product's success by ensuring that product quality was high, and serving as a resource to new members of the development team.

2.1.6.2 Daily builds

Daily builds were a standard practice, and both the requirement of developers meeting daily deadlines as needed and having the necessary build team infrastructure were already in place. As part of the daily build process, a standard test analysis program was run to find certain types of bugs. If a bug was found, the build was 'broken', and the responsible developer was required to address the problem immediately.

2.1.6.3 Source control

Strong source control was also enforced prior to the agile transition. The control system eliminated many source version conflicts and also provided a way to track changes within the project. The ability to track the history of changes was very useful for finding the source of problems when a build broke. The program used for tracking source code changes is ClearCase. This program allows developers to have their own branches of source code for side build testing and then easily merge into the standard product branch when appropriate.

2.1.6.4 Code reviews

Code reviews for source submissions by peers and team leads were a common development practice. The submission and review process got tighter as the schedule progressed, and milestone/Beta build deadlines were approached.

2.1.6.5 Frequent builds

Providing frequent builds for internal deployment was also a common practice. In addition to the development team and supporting teams, builds were frequently provided to other organizations within the company. This allowed the development team to gather frequent feedback on quality and usability of features for the product. There was also a mechanism to provide Beta builds company wide, with a forum for feedback. In some cases, the development team was able to get 50,000 users to test an early version of a previous version of the software.

2.1.6.6 Proof of concepts

There were several 'proof of concept' products that used agile development that existed within the division, such as the recent development of SameTime 7.5 (an instant messaging product). There were also several examples of successful agile software development projects that were very large and also had team members in various locations within the United States, Europe, and Asia. This was important to show team members that agile development does work, and was not just a fad or development theory. It also helped dispel the belief that agile development doesn't work very well with very large projects. The experienced practitioners in the company were available for discussions and provided further presentations to the team.

2.1.6.7 Executive level support

Executive level support was visible throughout the larger division, making it easier to adopt an agile transition without taking excessive risk of implementing a new approach. The executive support was relayed to employees via emails and a company wide wiki that all employees had access to. Additionally, periodic organization meetings held by executives reinforced commitment of the development change.

2.1.6.8 Strong customer ties

The development organization had strong customer ties. Management, development, and quality engineering team members had various relations with customers either through

dealing with critical customer issues, trade show contacts, on-line discussion forums, and the "key-grip" program. The "key-grip" program paired development and test engineers with large customers that implemented Beta versions of the product. This provided the organization the ability to get feedback about product requirements. It also provided the opportunity to have additional testing of Beta builds for further feedback.

2.1.6.9 Commitment to learning

Employees within the organization were empowered to spend ten percent of their time each week on learning. The choice of what they could learn was up to the individual employee, but it was expected that the time would be spent to improve their skills, and contribute to the greater success of the company as a whole.

2.1.7 Results

The agile development transition is about half way into an eighteen month project. While it's too early to draw conclusions on the overall success of the transition, there have been some notable achievements during the transition.

2.1.7.1 Beta quality (Milestone) versions

To date there have been four Beta (or Milestone) quality versions of the product, with a fifth one currently being prepared. This compares favorably to the progress of the development cycle using the waterfall method, where the first or second Beta version would still be under development. Quality of the builds has also been higher compared to builds using the waterfall process. The higher quality level is unable to be quantified at this point due to lack of comparable metrics such as defect rates. Since there is more frequent testing, and the features are developed using test driven development there are less bugs found. It is also difficult to compare to the waterfall method as much of the testing was performed at a much later date, making bug rates hard to compare. There is anecdotal evidence, however, that quality is higher. This is based on feedback received from customers using the Beta quality builds and the stability of the daily build versions running internally ("eat your own dog food").

2.1.7.2 Stronger business understanding

Employees have a better grasp of the organization's business goals and customer goals. Using the waterfall method, product requirements were determined well in advance, and hence developers and testers 'merely' had to implement the requirements. With the agile development process, employees have more frequent customer interaction. The interaction provides feedback on features that have been implemented and the opportunity to provide and discuss new requests that customers need. Understanding the customer needs is important to make sure that the requirements are valid. Another aspect that is occurring is that employees have a greater understanding of how a particular customer's needs fits within the organization's business goals and how it prioritizes compared to other customer needs.

2.1.7.3 Stronger teaming

Stronger teaming has developed in the organization. Prior to the agile transition, there was a clear division between developers and testers. With the agile development method in place, there is much more cooperation among the team members. QE has had more of a say in the development process, and both development and QE realize that a team effort is necessary to achieve the goals. This isn't uniform across all teams, but significant progress has been made.

2.1.7.4 Increased automation testing

Automation testing has been improving. For newly implemented features, automation testing has been completely implemented for around 60% of these cases. While this hasn't met the goals of complete automation testing, even for newly added features, the amount of automation testing has contributed to higher quality and the ability to detect regression problems sooner, on subsequent builds. As discussed below, there are factors that have prevented the team from reaching its goal of 100% automation testing capabilities.

2.1.7.5 General acceptance of the transition

During the transition there was initial resistance and skepticism at all levels of the organization, from developers and testers to management. As time progressed, members of the organization began to see benefits of agile development, as the quality of builds increased and positive feedback was being received from customers and internal deployment.

2.1.7.6 Customer interaction

The survey showed that over 60% of respondents or their teams had regular customer interaction ranging from a one week period to the length of the iteration cycle. This is a key component of agile development, as customer feedback is vital to ensure that the proper requirements are being developed, both from the initial idea/proposal standpoint and the actual implementation. While there is no baseline to compare interaction to previous contact levels, discussions with individuals showed that customer interaction prior to agile development was sporadic, and largely occurred when urgent customer issues arose, or at tradeshow attendance.

2.1.7.7 Improved productivity

The survey showed that over two thirds of the respondents feel that agile development has either mildly or greatly improved productivity. One of the survey responses indicated that the biggest benefit of using agile development methods was "higher quality, more consistent productivity", and another response indicated "reduced overhead, more focused on actually delivering small increments of working functionality rather than [work] that may never be finished and may not meet the requirements".

2.1.7.8 Lower code submissions

The number of code submissions for this current product version is around one-third lower compared to a prior recent version for a comparable time period (the first six months of development effort). This partly reflects the prioritized approach to implementing features, and the higher quality of those features (due to less follow-up

fixes submitted). As previously mentioned, some code from another code stream is automatically being merged into this current version, so the actual amount of code planned to be put into the new version is even less than the current ratio of overall code submitted.

For code that is submitted directly into the product (i.e.: not automatically merged from a parallel code base), the survey reported that the code is always tested in a side build 56% of the time, and often tested in a side build 28% of the time. Side builds are builds that developers use to create changes in for testing without submitting the actual code changes into a daily build. Testing in a side build further improves the quality of code and avoids build breakage, which can slow down the process for other teams.

2.1.7.9 Clarity of existing debt

Agile has increased the clarity of the importance of removing technical debt. While prioritization of technical debt has always occurred within the organization, the agile development transition has led to an effort to review existing features within the product to consider possible features that should be removed. The team promises to be more aggressive about pulling code out to improve value for customers. As one senior team member put it, "a feature is guilty until proven innocent". This presents a tough challenge to the organization, as the product needs to maintain backward compatibility.

2.1.8 Major obstacles encountered

In addition to the positive aspects that the agile development transition has brought thus far, there were also some significant impediments encountered during the transition. This section discusses the major obstacles encountered during the transition.

2.1.8.1 Impact of lack of complete automation testing

This has been a trouble spot for the development and QE teams. In some cases, the development teams have throttled back on output and have switched to building automation tests for certain areas. This is more prevalent for areas where large new

features or substantial rewriting of existing code is happening. The net result is that the code that gets written should be quality code, but will impact the overall functionality of the feature sets by limiting how much of the required functionality can be included. In the meantime, some technical debt of the code has surely accumulated and will need to be discovered.

2.1.8.2 Impact of non-agile developed components

With an agile development process, teams commit to complete certain requirements during each iteration cycle. For stand-alone requirements (it doesn't depend on another piece of code, or the dependencies are on fixed, existing code) teams have complete control over the implementation. For requirements that depend on another group's functionality or requirements, the situation becomes more complicated. For requirements that are being developed within the same team, coordination can be achieved as the goals are the same – complete the given tasks during the iteration cycle. The real complexity comes in when an agile team is dependent on source/modules/components from non-agile teams. In this product's case there are dependencies on third party modules and modules developed by groups within the same organization but do not have an agile approach. Iteration and feature completion were impacted on at least two occasions due to the other internal group not completing their requirements on time. This is part of the culture clash of waterfall and agile development methods. The waterfall team asked for a list of requirements early on in the development cycle. The agile team couldn't offer an exact list of what requirements were needed because it wasn't known early on. An alternative is to state some requirements, but they will likely not be the real requirements, resulting in wasted development and testing time. Management intervention was required to assist in getting the modules and source code that was needed.

2.1.8.3 Thrashing

Although some aspects of thrashing (employees switching between tasks due to interruptions) were expected, it still posed a significant problem to the team. QE was impacted as there were several code releases that needed to be worked on at one time. Using the waterfall method, QE didn't get heavily involved until development work was

complete. At that point the developers would begin work on the next project. Since the next project used an agile process it also required QE's involvement from the beginning. This presented a large dilemma for management, as QE was expected to finish testing the prior (non-agile) product and begin work on the new product using test driven development (the test cases were written first, and development coded the requirements to them). Since it was important to get the prior release out on schedule, there was a conflict as to what work should be covered first. As a result, there was some limited testing performed on certain feature sets of other versions to allow for a smooth transition to the new product (using agile methods). While this wasn't the preferred option, limited resources dictated that something had to be cut somewhere, and having a successful agile implementation was deemed more important.

High priority customer issues for existing versions of the product also created serious interruptions as developers needed to focus on serious problems instead of the new product requirements. This resulted in missing the completion of some iteration cycles either by failing to get complete requirements coded within the iteration period, or getting full testing completed, or complete automation tests written. This behavior appears to be common for legacy products that transition from waterfall to agile development. The recent waterfall to agile transition at Harvard Business School experienced some delays in agile project completion due to the need to provide maintenance and support for existing products[26].

2.1.8.4 Legal requirements

As part of the product release process there are certain legal requirements that must be met and reviewed prior to release. This posed a notable clash with agile development, as the goal of providing monthly build versions of the product to customers requires that the same legal requirements apply. The goal of having a monthly Beta quality build was impacted by the legal requirements of signing off on the originality of the product. This process typically added another week to week and a half before the build was ready for distribution to customers.

2.1.8.5 Build breakage

Regular occurrences of build breakages slowed down the availability of daily builds. The current build process (compiling and linking the source code) takes around four hours, and the 'kitting' process (grouping of files to an installable format) adds another two hours. Once those steps are complete, a test program is run on the overall build to check for various regressions, and requires another 45 minutes to complete. The long build time means that only a maximum of one build can be completed per platform per day.

The main reasons for build breakage are:

- code submissions from prior releases (a non-agile concurrent code stream) getting automatically merged into the agile version
- the large number (greater than 10) of platforms (operating systems that the product is built for) results in the lack of ability for developers to consistently test compilation on all platforms

The times for building do not include the time required to determine what caused the break, how to address it, and merge in the appropriate changes. Two aspects that helped with this task are the strong source control and tracking mechanisms, and a competent build team. The large code base and time required to complete a full build restricts the frequency that builds can be completed.

2.1.9 Role changes of team members

The roles of managers, technical leads, and developers all changed significantly during the transition. This caused some employees to question whether or not the team and role was appropriate for them, along with requiring management to get involved with various personnel issues.

2.1.9.1 First line manager role change

There is a greater emphasis on trust in agile organizations, and as a result managers may end up losing some of their authority and ability to control the work that they were accustomed to. Instead of directing developers on what tasks to perform, the agile teams use a 'pull' system to bring in prioritized work to complete during each iteration cycle (there is also input from other parties such as the product owner, etc.). The loss of power was seen as a threat to some managers, as their role can be changed to solely removing impediments and handling personnel issues instead of directing employee activities. The overall policy for the team states that management is not allowed to check in for status of the project during an iteration cycle, but rather has to wait for the status during the scheduled time interval (every 3-5 weeks). This provides a greater level of trust between the developers and management team, as the developers can focus on their work, knowing what they have to deliver. The project results are transparent during each iteration phase, as the work is either completed, or it isn't. The team takes responsibility for its requirements, and management can verify compliance. According to one individual, at this point in the transition, "managers are disoriented to some extent. They used to be team leads, but it's now a supporting role, getting resources, helping the teams follow the process, clarifying user stories, solving staffing problems, and other supporting problems."

2.1.9.2 Technical lead role change

As the developers became more empowered, the technical leads were expected to perform some of the duties typically handled by managers (non-personnel related issues). Some technical leads wanted to remain 'purely technical', and required a role change to avoid having to lead team meetings, check status, determine what requirements should be present, etc.

2.1.9.3 Developer role change

Since test automation is a critical piece of an agile development process, developers have become involved with writing automation test code to ensure proper testing is performed. As mentioned earlier, this resulted in the lack of complete implementation of some

requirements. It also changed the role of some developers to temporary QE employees to get proper testing performed.

2.1.9.4 QE role changes

Once agile development is fully implemented, including complete test automation, there will be a reduced need for 'black box testers'. Black box testing requires no knowledge of the inner workings of the feature or system. Since the requirements are written according to the test plan, there may be diminished need for test engineers that experiment with various settings to test the features. Conversely, automation testing has shifted the emphasis to use test engineers with more development skills to write the necessary test programs and drivers for feature and system verification. This role behavior change was also seen at Misys Healthcare Systems, as presented at the Agile 2007 conference:

> *"The skill set for our QA Engineers would need to shift. Our QA Engineers were all very senior automation engineers using Mercury QTP and Test Director. They were all writing test scripts to test sophisticated data validation and workflows. Now, we were going to need our QA Engineers to start writing fixtures in C# to access the business logic directly.*[27]"

2.1.10 Conclusion

While the product is still under development, there are several factors that need to be taken into account when undertaking an agile software development transition. The first is that pilot or isolated projects can be difficult to successfully achieve in an organization with a mix of waterfall and agile teams. Pilot projects are susceptible for two main reasons. The first reason is that a new or foreign process will likely be met with resistance. Without the proper support system in place, the project is unlikely to gain traction and succeed. The second reason is that dependencies on non-agile components or applications may cause the pilot project to miss its goals. Non-agile teams have a different set of principles they adhere to, such as long-range planning of requirements.

Since agile teams may not know what their requirements are until just before starting the development process, providing sufficient lead time to non-agile teams for their needs is impossible. Attempts to change the schedule of waterfall teams with minimal notice will likely result in not receiving the necessary components during the desired iteration cycle.

Role changes will likely occur within the organization. Evaluating the developers, managers, and technical leads to understand their skill set and real desires can help minimize the job changes during an agile transition and minimize disruption.

Executive backing is critical. This came up during conversations with several individuals, including management. Without proper support, an agile transition is unlikely to succeed as the risks an individual undertakes may not be worth it if the project fails. Another reason is that executives looking for current project status may not see significant progress by relating it to the prior waterfall method. The result will be less code written, and hence less features. What won't be visible is the quality of the work completed and increased customer satisfaction during the iterations.

Legacy code can heighten possible business and organizational impact. The impact of having legacy code ranges from lack of full testing to the need to respond to urgent, existing customer issues, and the conflicts that can arise from QE testing the prior version while working on the new product version. From a product perspective, legacy products can also restrict what features can be implemented and removed from the new product. Existing customers expect subsequent versions of the product to be compatible with existing applications. Removing or changing existing functionality (even if it's 'better' functionality) can break customer applications, leading to product satisfaction issues. One possible way to mitigate the impact of legacy products is to determine how much work can be done by using a fraction of the team's available time (e.g.: 80%). This can account for meetings and the need to switch gears and work on urgent customer problems. This also helps prevent generate overly optimistic estimates about the work that can get accomplished during an iteration and project cycle.

Evaluating the organizational impact that an agile transition may bring will help minimize the disruptions of the change and contribute to a smoother product release. This may require making role changes ahead of the transition to get the proper team in place. Additionally, changes in other parts of the organization (such as legal and HR) may be required, resulting in a complete agile organization. From the HR perspective, the reward and compensation system may require modification. As one manager stated: "We're good at extrinsic (compensation, benefits, etc.). We're not *as good* at intrinsic – we extol an individual vs. the team."

Lean (and agile) development emphasizes the focus on team efforts. As a result, the rewards system should be structured to recognize the team effort. Otherwise, individual employees may act on what they perceive to be in their best interests for their annual employee evaluation. To date, there have been no changes to the rewards system, but management does recognize the need to tie employee goals and rewards together to focus on team productivity. There is however, a portion of the individual's performance rating that does encompass teaming efforts, and managers have the ability to fairly account for team activities.

Prescribing one set of development and organizational practices to apply to entire organizations is not appropriate. Rather, organizations can use principles for development practices that allow for flexibility between projects to suit the needs of particular projects and satisfy cultural concerns. When transitioning, the goal should be to constantly improve the processes, so it is not necessary to implement all changes at one time.

While not consistent across all members of the project, many individuals and teams had a long history of testing code prior to submitting. Daily builds were also a common practice. Although it is hard to quantify, some of the product's features weren't widely used. The reasons for non use were: incomplete feature implementation, incorrect feature implementation (based on customer requirements), defects within the feature, and lack of

demand for features implemented - largely based on an individual's 'feeling' of features being needed.

The development team and QE had a lengthy history of dealing with customer issues and maintaining legacy code. While this offers benefits by keeping in touch with customer needs and experiences, it has also been a burden on the organization. The burden results from the need to switch gears from new product development to putting out fires for a customer crisis.

It is very important to identify the outstanding obstacles or constraints within the organization and start addressing them in order of importance. Changes within the organization can be made incrementally and improved upon over time. In this case, the largest obstacle for the organization was the lack of complete automation testing. Automation testing is a critical aspect for lean and agile development teams as it provides an efficient means of testing every build, especially daily builds. Automation testing is cheaper, more thorough, and more consistent than having testers perform manual testing.

At this point, there is no *quantifiable* measurement of success or change within the organization (each team does have their own burn down charts for progress on complete feature implementation). However, there is the ability for management to determine the current status of the project. With an agile development process managers have the ability to determine whether or not a requirement is complete. There is no partial success, the feature is either complete, or is not complete. There are no '90% complete' statements, in this case the feature would be classified as incomplete. This transparency gives management the ability to see the real overall status of the project.

The organization has been able to reap benefits that would otherwise be unreachable using the conventional waterfall development method, such as increased Beta builds, and more responsiveness to customer needs. The management team and engineering team have made some adjustments along the way, and it is expected that some additional tuning will be made to help the organization achieve its goals. The knowledge learned is

being tracked, and eventually will be shared with other software organizations within IBM.

2.1.11 Survey Results

I created a survey to assess the current status of the agile transition, from both organizational and product perspectives. The survey was sent to 51 individuals, including management, developers, testers (QE), and technical leads of all the teams. 35% of those surveyed, or 18 of the individuals responded to the survey. The survey consisted of open and closed-end questions. While the results of the survey are limited by the small population of responses, there are some findings that provide both positive and troubling signs for the transition. The survey was presented about one third into the agile development transition.

The first part of the survey consisted of closed-end questions, with the last two questions providing the opportunity to choose a response that was not offered in the survey. Key findings from the survey are mentioned here, for full survey results, see **Appendix 2.**

Nearly half (44%) of the respondents were software developers, 22% were technical leads, 17% were QE members, and 17% were managers. The high proportion of developers responding to the survey may distort the results of the survey in areas such as time spent maintaining legacy code, as this work was typically performed by feature developers as opposed to QE members.

50% of the respondents stated that the daily team meetings were very effective at achieving team goals. One third responded that the meetings were somewhat effective, while 17% stated the meetings were not effective. In discussions with individuals, the length of the meetings varied since the original inception of daily meetings, with the meeting typically getting shorter from initial one hour meetings that some teams held. As the process is still relatively new, it is expected that further adjustments will be made as needed, although attempting to 'please' everyone will likely fall short.

Iteration schedules were largely met the vast majority of the time, with respondents reporting that schedules were always met 33% of the time, and 44% met most of the time. Poor iteration schedules adherence occurred less than 20% of the time, with respondents reporting schedules were seldom met 11% of the time, and never met 6% of the time.

The goals of the agile/lean transition were clearly stated, as 81% of the respondents stated that the goals were clear. 19% reported that the goals of the transition were not clear.

The majority of the time (66%), teams completed the current feature iteration prior to moving on to the next feature in line. 6% of the respondents reported that they frequently moved on to the next feature without completing the current feature iteration.

Testing of new code prior to submitting into the source code base occurred very frequently, with 55% of the respondents stating that the code was always tested prior to submitting, and 28% responded that the code was often tested prior to submitting into the general source code base. 11% of the respondents reported that the new code was never tested in a side build prior to submitting into the source code base.

Nearly two thirds of the respondents (62%) do not have a dedicated developer or test partner. These results may indicate that some developers are performing more QE work, or QE individuals may be covering more than one developer's work, or a combination of both.

The majority of respondents (72%) reported that they spend at least 10% of their time maintaining legacy code or customer problems. Of this number, 11% reported spending more than half of their time in this capacity, and another 33% spending over 20% of their time for the existing code.

Respondents reported that team members interact with customers on or before the iteration schedule intervals 44% of the time. This number is broken down with team members reporting contact weekly (6%), monthly (32%), and the length of the iteration

schedule (6%). 17% of the respondents reported having no contact with customers. Other responses for this question indicate that only certain team members interact with customers, and others interact only as needed, such as when questions are posed on electronic discussion forums.

The impact of the lean/agile transition has largely had a positive affect on productivity within the organization. 74% of the respondents stated that the transition has been positive, broken down by 19% stating that it has greatly improved productivity, and 55% stating that it has a mild improvement on productivity. 13% of the respondents reported that the transition has resulted in decreased productivity.

The open ended questions provided insights into the benefits and obstacles encountered thus far during the transition. Notably, respondents reported an increase in product quality, improved teamwork, and increased productivity. For obstacles, the largest type of response related to the existence of legacy code and the maintenance it requires. Some responses indicating resistance to the transition to agile development were also received. Following are some of the more notable and informative responses:

In your opinion, what is the biggest benefit of using agile development methods?
Of the 16 respondents to this question, the largest category of response related to an increase of the quality of the product:
"Building test driven development where quality is built in from the start"
"There seems to be fewer defects in development's product."
Other responses fell into the categories of:

Teamwork
"Increased personal and team accountability..."
"...much better understanding of the strength and weakness of my teammates."

Real work and productivity

"More time spent developing and less time spent writing a spec up front when you don't know all the issues yet".

"Higher quality, more consistent productivity."

What is/has been the most difficult issue to overcome for successfully adopting agile development?

Of the 15 responses to this question, the largest category belonged to the existence of prior versions of the product. In other words, the biggest issue was working on a legacy product. Some of the responses received were:

"managing technical debt"

"...getting pulled in multiple directions, not able to put dedicated focus on this project."

"dealing with legacy issues... and other things which are not part of a given iteration."

Other significant categories for responses were:

Transition adjustment

"Transition is very painful, feels bad"

"Preventing operation in a waterfall fashion"

Planning and measurement

"Perception that it is taking longer"

"Proper planning and measurement of progress towards end goal"

Please add any other comments that are relative to the agile development adoption process.

Eight responses were received for this question. Notable responses received for this open ended question were:

"Some tasks and design issues do not fit well into short iterations."

"The company has a process-oriented culture; it is difficult to convince people to give up wasteful processes, measurement, and other familiar 'waste'."

2.2 Kronos

2.2.1 Description

This successful software organization develops several human resource management software systems that are widely used throughout Fortune 1000 companies. Like IBM, they had a firmly entrenched culture of using the waterfall development method to develop their software products for over 20 years. The organization consists of nearly 600 engineers involved in developing their products operating in several different locations.

The Kronos development organization develops products such as Workforce Central Suite, consisting of nearly 20 applications including Workforce HR, Workforce Payroll, and Workforce Timekeeper[28]. Much of the product development uses the Java programming language, followed by .NET. Kronos undertook the agile development transition beginning in 2005 for the engineering team.

2.2.2 Why agile?

The first steps towards adopting an agile development process came from the senior executive level of the organization, the Chief Technology Officer (CTO). From this perspective the current development process "didn't feel right". While the organization wasn't broken going into the agile development transition (the organization was predictably delivering quality products), the CTO's driving concern was to ensure that as the organization grew, they would not fall into the trap that some other large organizations encounter: slowing productivity as opposed to the nimbleness that small teams possess. Factors contributing to the feeling were the high number of problem reports, time spent debugging the code was increasing at the cost of implementing less features, the target release date for recent versions of the product slipped, and employee burnout was being experienced. The twelve month product development cycle plan consisted of roughly six months of coding and six months of testing. The plan typically resulted in five months of coding and seven months of testing.

The goals of adopting agile development practices were: lower overhead associated with development, change responsiveness to customer needs, and improve the quality of the products. Due to employee burnout, fun was also a reason for adopting agile software development practices. The management team understood that developers wanted to write code. They wanted to create new technology and solve problems. Fixing bugs and regression testing was considered less fun, but was a significant piece of the development effort. For a twelve month cycle, developers were spending five months doing the fun stuff (writing code), and seven months doing the not-so-fun stuff. The last seven months were high pressure times, due to the unpredictability of testing and debugging. Long hours spent finding and addressing the bugs during the last seven months was a common practice. A more sustainable pace was needed, not 'bug bashes' where people would stay late to fix bugs within the products. Management wanted to change the cycle from 5/7 to 11/1, where eleven months were spent doing the fun stuff. This coincides with statements made by Pete Deemer, chief product officer at Yahoo! (India Research and Development):

> *"...**Waterfall**- it's not that much fun to work within. In fact, we'd go a step further and say that the **Waterfall** is a cause of great misery for the people who build products, and the resulting products fall well short of expressing the creativity, skill, and passion of their creators. People aren't robots, and a process that expects them to act like robots often results in unhappy people."*[29]

2.2.3 Details about the agile implementation

The first steps for adopting agile development methods were initiated by the CTO by having consultants (Cutter Consortium) come in and evaluate the organization. After the consultants observed the organization's development practices the CTO then presented the results to the organization and laid out how the development transition would be made. The presentation also showed the engineering team why the change was necessary and how things would improve. This executive level support enabled easy buy-in by the engineering team, as it came from the CTO himself (it could be viewed as essentially a

mandate but the positive aspects were highlighted). A rallying cry of "11/1" was used to motivate employees to see the positive aspects of agile development. "11/1" indicated that the engineering organization was going to focus much more on the actual development work (11 months), and less time on the integration and final testing phase (1 month). Not only would the change help the company and direct stakeholders, but it would also help the employees by focusing on what engineers want to do, as opposed to the more mundane aspects that were occupying increasing amounts of time. Five full time consultants were brought on board, and a team of five to ten internal full time agile coaches was assembled to provide training to the organization. It took about three months to roll out agile development practices to the organization. After a six month period, the consultants left, leaving the internal coaches to finish carrying out the agile transition. The method transitions took place in two phases. The first phase, instituted in 2005, focused on adjusting the development practices used. These practices include the incremental approach to developing the products in smaller, "bite-size chunks" and retrospectives. The second phase, instituted in 2007, involved the adoption of Scrum practices within the organization. This phase introduced role changes for managers and provided greater empowerment to the development teams.

Although Kronos was a traditional waterfall development based company, there were some practices that the organization had used that were agile based (without using the agile name) that made the choice of choosing agile development easier. These practices included using iterations of four to six weeks to develop web based versions of products, and bringing in sales people to co-locate with the engineering team. The co-location of engineering and sales allowed the organization to be more in touch with what customers were actually looking for.

It was decided that the entire suite of products would make the transition to agile development together. This was due to the overhead penalty that was forecast by transitioning one or a few groups and then having to apply an agile transition process to the remaining groups later. Further, it just 'made sense' to transition all teams at once, since there were several products that in essence became integrated into one large suite of

applications. In the words of one director of development, they were entering un-chartered territory, transitioning the whole development organization at once, contradicting recommendations in agile development articles and consultant's suggestions.

The iteration length for most of the projects was two weeks. Every six weeks to two months, a release candidate was produced. This consisted of integrating all of the various products being developed. The product release cycle was set at eighteen months. An eighteen month product release cycle was chosen because customers are not always able to upgrade too frequently. The chosen duration does allow for customers to receive demonstrations of the new version while it is in development, however, so they are able to see the progress and provide feedback on the features.

The new feature development goal was to implement features for twelve user stories. A user story is essentially a surrogate for a high level specification of a product requirement. Initially, the engineering team felt that they could accomplish only six stories (or headlines) during the given time period. This caused some frustration within the management ranks, but the team was told to "do the best you can".

2.2.4 Assessment of the transition

In order to gauge the progress of the agile transition coaches are involved to check the pulse of the teams. They meet every 2 weeks to discuss the status of the transition. The subsequent addition of Scrum practices made it easier to determine an agile health assessment and will show if and where additional focus is needed. The organization is looking at the possibility of using quarterly audits to assess the state of agile development.

The CTO of Kronos holds a monthly lunch for twelve randomly selected employees. The intent of the lunch is to discuss the organization and allow the CTO to stay in touch with what is going on within the engineering teams. This provides an opportunity for individuals to discuss how the agile development process is working, and to ensure that employees understand the intent behind agile development.

2.2.5 Results

The agile transition has had some positive results for the organization. Over the past two and one half years, "a lot has changed, but only a little has changed each time".

2.2.5.1 New features

Although the team initially thought they could accomplish six user stories during the development cycle, they were able to complete fourteen stories after the first year cycle. This far exceeded the initial expectations, and is in large part due to the empowerment that the employees gained during the agile development adoption.

2.2.5.2 Mix of agile and waterfall

The planning phase of the development process is still performed in the waterfall method. A commitment is made early on about the headline details of features that will be included in the products, but there is flexibility that allows for feature changes to be made during the development cycle as customer needs dictate. This was partly due to the waterfall mentality that some members of upper management possessed.

"The tension of fixed scope/fixed budget/fixed schedule pushed management into old habits." At some levels of the organization, it was noticeable that management strayed from agile development principles. The toughest adjustment was that scope should be flexible. Implementing Scrum helped management "start living agile", and adjust their muscle memory.

2.2.5.3 Quicker responsiveness to customer needs

The organization was able to respond to customer changes during the development phase due to feedback received from customers. In their words, they "have implemented several features not in the original plan in response to competitive pressure or customer needs". The development change has started to become visible at the marketing level, as the urgency and responsiveness of development to customer needs has been demonstrated.

This contrasts with the perception of a non-engineering employee where the response from engineering wasn't as flexible as was hoped when proposing a change. The response could be the result of the new change not being of high enough priority compared to other features being developed, but the reasoning is unclear.

2.2.5.4 Improved cycle time

Kronos was able to significantly reduce the amount of time spent getting the product ready for public release. There has been a steady progression of an increased percentage of time spent doing the actual development work compared to testing and integrating. Instead of spending five to six months for final testing for the most recent release of the product, only four months were required to ensure the product was ready for customers. For the next release, the organization is on track to exceed the 8/4 timeframe.

Another positive aspect is that during the agile product development cycle there are between six and nine 'Release Candidates' (Beta quality versions) available during the cycle. This allows the organization greater opportunities to provide current versions of the product to customers to provide feedback on the feature implementations.

2.2.5.5 Higher quality

There have been fewer defects in versions of the products that have been released. A large part of the higher quality was due to the frequent integrations of the various components of the suite of products. Instead of waiting until the end game of the development cycle, the development team pieced the entire project together every six weeks to two months. While each individual piece may have been tested during the development cycle, each integration build allowed the opportunity to test the entire suite of products to look for possible interaction issues. It also prevented the situation where defects would be built on top of defects that remained in the product until final testing was performed.

2.2.5.6 Disruption minimization

In order to minimize disruptions for existing (legacy) products and focus on developing the new products, the organization used a dedicated team for new feature development and a dedicated team for customer problems. The rapid response team for handling customer problems consisted of five to seven people with the goal of getting customers up and running quickly. They also adopted a Service Pack (SP) schedule to provide solutions for urgent customer problems. The Service Packs were released once a month, providing customers with timely and quality solutions to their problems. Interruptions would come in to the feature development teams if a new high priority bug came in. The workload is based on team capacity, via a pull system. The Service Pack schedule interval was reduced from four months to one month to allow customers to get critical fixes sooner. Every month a final regression test on the Service Pack is performed and made available to customers (for those customers that can't wait, one-off fixes may be applied). This change was made to allow proper focus on what bugs to fix, and incur less interruptions and thrashing for the development team.

2.2.5.7 Employee satisfaction/attrition

Employee attrition initially increased during the agile adoption phase. This encompasses the role change for Project Managers and other personnel that left during the transition. Since the agile process has been implemented however, attrition has decreased.

2.2.6 Role changes

As was the case with IBM, Kronos encountered role changes for some of the positions within the organization.

2.2.6.1 Project Managers

During the agile transition, Project Manager's positions were eliminated. As part of the agile process, product owner positions were used. Most of the Project Managers became Scrum Masters, but those that didn't want to change left the company. The Scrum Master serves as the servant leader to the team by making sure that they follow the Scrum

process and that it is working appropriately. Essentially, the role became one of removing impediments, as the teams managed themselves.

2.2.6.2 Developers

The role of the developers changed from writing code solely for the product to spending half of their time writing test code. The position also required a change in their mindset from a 'throwing it over the wall' approach, where the assumption was that QA would test it and find any problems associated with the code.

2.2.6.3 QA (testers)

Testers had the initial notion of "we'll test it once, not more than once". This mindset required a change, as agile development requires frequent (constant) testing, including code that has already been tested many times. This also underscores the importance of automation testing, as in the words of one director of development: "you'll be running the test a thousand times".

2.2.6.4 Managers

Middle managers got squeezed the most from the agile development adoption. The lower level teams make the process work, and senior managers make proclamations about what needs to be done. Prior to adopting agile development they thought about the product, and not much time on how to make engineering better. After adoption of agile development, the focus has switched to topics such as 'how to automate the build process' and 'test automation'. The middle management group is 'only' accountable for productivity.

2.2.7 Conclusion

While Kronos obtained some portion of the benefits of agile development, there were other aspects and costs of the transition that were encountered, both anticipated and unanticipated.

2.2.7.1 Difficulty to overcome established method

The waterfall mentality was hard to overcome. As mentioned earlier, some of the planning phase of the development cycle still uses the waterfall method. Adjustment of possible scope changes was a new concept, as in the past every requirement was stated up front.

2.2.7.2 Insufficient management training

The transition wasn't initially thought of as a mind shift for the entire organization, but rather as a shift for development and their methods. As the transition progressed, however, it became apparent that other aspects of the organization needed to understand and account for an agile development organization. Upper level management was still expecting product requirement commitments via the waterfall mentality. As a result, Scrum was rolled out as a management practice in 2007 (over a period of five to six months).

2.2.7.3 Realization of benefits

It took about six months to see any benefit from the agile transition. Full benefits weren't realized until the project was completed, and expectations were exceeded. Empowerment of the employees allowed the team to exceed their expectations of what they could deliver within the product development cycle.

2.2.7.4 Do it again

The agile process worked well for the initial projects, but the difficulty came when the next round of projects were starting. Daily routines were remembered, but the kickoff procedure was forgotten as it was only performed once, one year earlier. Only one agile coach was retained during the initial transition process. This made it tough to kick off the next agile project as the organization required a review of past processes.

2.2.7.5 Clarity of goals

The clarity of the reasons for transition to agile development was made clear to the engineering team, as the plan, expected benefits, and reasoning were presented by the CTO to the organization. While all members of the organization understood that an agile transition was taking place, there were some misconceptions about what agile development would provide. One non-engineering employee I spoke with mentioned that there was anticipation of a more positive impact, but so far the impact hasn't been visible at the product level.

2.2.7.6 High expenses

The costs (in both monetary terms and initial confusion) should be carefully considered prior to undertaking the transition. It was clear that after careful evaluation by the CTO and the consultants, that the rewards for transitioning would be worth the cost, but excessive expenses may limit the executive backing and slow or stop the transition. In this case, additional costs were incurred for external and internal coaches, along with eventual Scrum training.

2.2.7.7 Lack of experience curve

By rolling out all teams at once, there was no experience curve that could be utilized to ease the pain of the transition. In other words, if one team experienced something that didn't work in the Kronos environment, then other teams may experience the same problem. Teams couldn't benefit from other internal organizations agile adoption practices.

2.2.7.8 Agile inconsistency

Each team made adjustments to the agile development methods that suited their own team. When individuals moved to other projects, they had different interpretations of agile development compare with their new team's agile methods. The entire team needed to be brought back to base agile, using fulltime agile coaches.

2.2.7.9 The right choice

At this point, two and a half years since the initial agile transition for Kronos took place, there are no regrets about the development change. The development change has improved in areas such as quality and responsiveness to customer needs. Prior to the introduction of agile development, engineers had stated that a particular monumental character set code change would be 'impossible' to complete. Using the waterfall development method, the nature of the work would be too difficult to manage. This is due to up-front requirements, and necessary changes that would come during the development phase. With the introduction of agile development methods, the change was able to be implemented, and in a relatively short period of time. This character set change is a critical piece of functionality that will help better position Kronos' products.

3. Conclusions

Implementing lean and agile practices can enable companies and their products to win in the market and gain a competitive advantage over their competitors by:

- being quicker to market
- reducing cost for product development
- providing higher quality software that equates to lower maintenance costs (i.e.: customer support, dedicated development debugging teams, less interruptions for developers to switch gears and work on bugs)

There are many cases where agile development methods have been shown to work in organizations. The level of success (and how it is measured), may vary from organization to organization, but there are some aspects of agile development that appear to be needed to have any chance at succeeding. This section provides detailed observations of factors that need to be present and decisions that need to be made when adopting agile development methods. To summarize the observations and requirements, a guideline of questions that organizations should ask prior to and during agile development adoption is presented to help determine the fit of agile development within an organization.

3.1 Basic requirements

The most fundamental pieces needed for a successful transition to agile development are proper executive level support, and clarity of why the transition is taking place, and what the goals are for the transition.

3.1.1 Executive level support

Executive level support is required to get 'buy-in' from members of the team and to ensure that executives understand the process when communicating the current status of projects under development. Since more than just the development method changes (e.g.: other role changes), it is critical to get all members of the organization involved in

understanding what agile development is, and that the transition is a full team effort. This type of change requires executive sponsorship to show that the change is for real, and that there are no penalties for adhering to the new development method. Perhaps this is why most of the agile transitions that I researched were initiated at the executive level ranks within an organization. For IBM's Domino product, the initial effort started at the product architect level (a technical executive), but was driven within higher levels of the company. At Kronos, the agile development transition started at the CTO level. KeyCorp, one of the papers referenced at the Agile Conference, listed the initiation level as their CIO. This level of initiation appears to contradict some of the literature that states that agile development practices arise from the engineering level. What may be possible is that certain, limited practices can be performed at the engineering level but overall agile development practices need reinforcement from above to have the maximum impact.

As part of the executive level sponsorship, there should be a proper support system in place to provide assistance to the organization during and after the transition. The support system may consist of coaches, consultants, training, wikis and other discussion forums. Sharing information on experiences about what worked and what didn't is essential. As a result, learning from agile transitions should be documented for future reference.

3.1.2 Clear communication

Communication of the reasons and goals of the transition is important to ensure that all members of the organization understand what is expected of them. Since individual empowerment increases with agile development methods, it is important for all team members to have a common understanding. The goals of the transition must be made clear to get all necessary parties on board with the changes that will come. Periodic meetings to provide updates and clarify further questions that will surely arise will help make sure that the organization has a clear understanding of the change.

3.2 Other significant requirements

While many components are needed to successfully adopt agile development methods, there are a couple of capabilities that can help an organization quickly navigate the transition: strong build capabilities, and full automation testing.

3.2.1 Strong build capabilities

The amount of time required to build and merge code changes can affect how quickly builds can be provided, both for internal deployment and for customer feedback. In cases where the build and merge process take substantial time (more than ½ a day), the benefits of having a daily build can be impacted if there is a problem with the build, requiring additional work to fix. This in effect can limit the availability of builds, as an error requiring a full rebuild for a 4 hour process can mean that the daily build won't be available. This is also compounded for products that build on more than one platform, if several platforms need to be rebuilt. The latter case can happen even in cases where a side build is made and tested on only one platform. As part of the build process, a source control system must be used to properly track changes.

3.2.2 Full automation testing

Automation testing is a key requirement for agile development. Automation testing is quicker, more comprehensive and consistent than manual testing. Having the ability to run complete automation testing allows organizations to quickly verify the functionality of the product and provide high quality releases for customers to test and validate that their requirements are met.

3.2.3 Understand the existing state of the organization

Assuming the proper executive level support is in place, and the goals of the agile transition are well understood, there are several key factors that affect an organization's ability to successfully adopt agile development methods:

- established development methods
- legacy products
- dependencies on other organizations for code/components

Agile development transitions are easiest for companies that do not have well established development methods, and aren't based off of legacy products. This is due to both the lack of resistance to change, as well as the ability for individuals and organizations to quickly change when needed. The existence of legacy products further complicates the picture. Legacy products force organizations to constantly monitor and attend to existing customer needs. This disruption for engineers makes it difficult to focus on the new development effort, possibly limiting progress towards completion. Management may want to consider calculating work capability at 80% of the value[30] to encompass disruptions and meeting time. Dependencies on other organizations for components or source code during product development may limit how agile a development team may be. When organizations are dependent on outside teams, they lose the ability to control what can be completed within an iteration cycle. If the outside team uses agile development methods, there is still a chance that the iteration cycle completion target may be missed, but it shouldn't be as substantial as when dealing with waterfall development teams.

This conclusion hints that product development advantages may lie with new organizations, or small organizations that have no existing products to maintain, and can be flexible in their development approach. As with any other product being developed, time to market may be the critical factor however, so speed should be considered along with scope and cost.

3.2.4 Don't bite off more than you can chew

The deployment method needs to be evaluated to determine if a small scale transition (e.g.: pilot project) should be undertaken, or a full company transition. In the two cases described above, different approaches were taken, each with its advantages and disadvantages. Salesforce.com undertook a simultaneous transition for all thirty of their development teams in the organization. Their considerations covered both small scale and large scale transitions:

"At this point, most literature recommended an incremental approach using pilot projects and a slow roll out. We also considered changing every team at the same time. There were people in both camps and it was a difficult decision. The key factor driving us toward a big-bang rollout was to avoid organizational dissonance and a desire for decisive action. Everyone would be doing the same thing at the same time. One of the key arguments against the big-bang rollout was that we would make the same mistakes with several teams rather than learning with a few starter teams and that we would not have enough coaches to assist teams every day. One team in the organization had already successfully run a high visibility project using Scrum. This meant that there was at least one team that had been successful with an agile process before we rolled out to all the other teams. We made a key decision to move to a "big-bang rollout" moving all teams to the new process rather than just a few.[31]"

From their perspective, Salesforce.com was successful in turning around their organization in three months with a reduction in product defects and quality code builds every month. Determining the choice of small scale or large scale adoption needs to consider the organizational capabilities and ability to change. Recall the transition pains that several teams within the Kronos organization would have to endure if a particular practice or tool didn't work when transitioning all teams to agile development practices at the same time. Take into account the current agile practices that the organization presently uses. Both IBM and Kronos used certain practices that were effective prior to the transition that were essentially agile practices. Regardless of the choice chosen, a pilot project or complete agile adoption, be aware of the culture clash between waterfall and agile development methods.

3.2.5 Choose the right product release interval

Agile development methods may provide greater benefits to organizations that can generate frequent customer upgrades and deployment. This can occur if customers can get the features they need quickly, have a limited scope (and hence limited risk of

regression problems), and are able to deploy the product quickly and with low cost. This model appears suitable for Software As A Service models (SAAS) products, and organizations that provide products that can be deployed on a uniform platform (such as PatientKeeper). There are other concerns however, such as sensitivity to changes and government and industry regulations that may prevent customers from being able to use upgraded releases as quickly as developing organizations would like.

Regardless, just because an organization has ship-quality code doesn't mean they have to ship it right then. Knowing when to release the next version of the product to customers is important. Of course if the costs of providing frequent versions to customers is minimal, and customers aren't upset if they 'can't keep up' with the versions, as some customers have stated, then being able to have frequent releases may be a good thing for the company.

3.2.6 The long and winding road to true agile development

"Beware of agile noises". In the organizations I met with, there were concerns raised at various levels about 'working waterfall while talking agile'. There are several reasons that this appeared: existing culture of waterfall development methods, lack of proper training, lack of belief in the method, and resistance to change. Stating that an organization is agile is once thing, but the true test comes when the results are shown by providing high quality products that succeed in the marketplace. For the organizations studied, there have been some signs that product quality has improved and customer-desired features have been implemented due to being agile. One aspect that is missing however is how well the products have done or will do in the marketplace by using agile development methods. Further, it is difficult to compare agile market results with prior market results due to the changing business environment (e.g.: product sales vs. service sales, SAAS models, etc.). Assuming that product sales are strong, and customer satisfaction is high, a useful metric may be the developing organization's costs associated with the development effort. This too becomes harder to measure, as initial transition costs and reuse of existing code or components can yield conflicting results.

Attaining pure agile development can take a long time to achieve. Within IBM, approximately one fourth of software development projects use some aspect of agile development[32]. This shows that there is still a ways to go before agile development is completely integrated in the company. Some development projects use complete agile development methods, some use agile development methods but are not exclusively agile, and still others don't use any agile development method. This pattern appears to reflect the transition period necessary for complete adoption, especially when related to the Dr. Dobbs survey information results mentioned previously.

At Kronos, some of their products have integrated agile development methods, while others may use other methods that work well for that product or group's situation. This is particularly true due to development groups that may have been incorporated into the organization via an acquisition, as is the case at both IBM and Kronos.

3.2.7 Timing the benefits

Benefits may not be visible early on. It is important to give the project sufficient time to show an improvement from the agile methods. In both of the organizations I looked at for this paper, both were seeing initial improvements around the six month timeframe, but some of the Agile Conference papers mention an earlier demonstration. It's important to persevere during the project, particularly at the early stages of implementation.

3.2.8 More than an engineering change

More than just developers and testers may need to change during an agile development transition. This includes management, sales, human resources (HR), and Project Managers.

As shown earlier, the roles of many individuals may need to change to adapt to agile development processes. While developer and tester role changes may be anticipated, there are other roles that will require change as well. Management role changes were

visible in the two organizations discussed earlier. Sales personnel may need to change to establish closer relationships with the engineering organization and possibly help establish relationships between customers and engineers (something that some sales personnel have been loathe to do in some cases). Human resources personnel may need to adjust to the team aspects of agile development by instituting different evaluation and reward systems. Project manager's roles and responsibilities may also change with agile development methods. In one discussion with a senior project manager, there was a feeling that more project managers could be used for agile development projects, as they are essentially made up of many small projects.

KeyCorp, a large corporation, encountered project management role changes during the waterfall to agile development transition:

> *"First and foremost, all the project managers were returned to their positions in the development organization, reporting directly to development managers. There the project managers (PMs) became accountable for project success – no longer were they simply facilitators and enforcers of the process. While they had always been accountable for truth in reporting, now they were accountable for results – their bonus was on the line.*[33]*"*

Human Resource changes made explicitly for agile development practices weren't visible at the organizations that were researched, although management understood the need to have updated HR policies in place. In the organizations that I looked at however, portions of the review process do incorporate team accomplishments, and managers have discretion on how to fairly evaluate employees. For large organizations however, changing evaluation and reward policies can be a monumental task.

3.2.9 Patience is a virtue

Adjust as needed, but give the process appropriate time to succeed. This can be tricky, as determining the right amount of time to allow a method to diffuse and be accepted in an

organization may vary. Experienced coaches and practitioners can help guide the team and transition for these cases. Evaluating the transition is imperative to make sure that problems and resistance is found and addressed early on.

3.2.10 Developing agile developers

Agile and Lean software development methods aren't explicitly taught in schools. In discussions with Computer Science students, this is likely due to the short duration of projects, as a typical semester is around three months. Since iterations are typically four weeks long, this doesn't provide much time for useful project completion. Arming the new wave of software developers with the philosophy and tools of agile development may ease the overall transition from waterfall to agile software development. This also reduces the acclimation period for developers when they become employed by software organizations.

3.3 Potential benefits of agile development

As discussed in the prior two organization cases, the anticipated benefits of agile development are: developing products that customers want and will consequently purchase, higher quality products, organizational cost savings, and timeliness to market. There are also some further potential benefits of using an agile development approach that merit mention.

3.3.1 Employee retention

There has been anecdotal evidence of software developers leaving companies that follow the waterfall development process for the lure of agile practices[34]. Since empowerment of employees is one of the principles of agile development, workers may feel more appreciated. They also get to have a greater understanding of the business goals of the organization and their customers. For Kronos, one of the goals of implementing agile development methods was to bring fun back into the office. Agile practices may help avoid the "brain drain" situation.

Conversely, one negative aspect that was mentioned relates to engineers that are entrenched in the traditional development model and have work assigned to them by management. This appeared to be more of an issue for more senior engineers, and less of an issue for those with less experience. Another issue that arose was the lack of completion of agile development. With the waterfall development method, there was a slow period, or 'lull' during the planning phase that gave the engineering team a break from the normal engineering routine. With agile development, a new iteration starts once the prior one completes. For some engineers, this change in the work mode was unsettling, as no 'breaks' were readily visible.

3.3.2 Reduction in TCO for partners and customers

Many customers perform their own testing of products (even final versions) before deploying at their companies. This results in additional costs for customers (increasing the Total Cost of Ownership, or TCO) and for the company developing the software, as companies will deploy later, slowing the revenue stream for upgrades. In addition, companies may adopt fewer upgrades due to the time it takes them to test before deploying. Fewer upgrades means less revenue for the development firm, and also leaves them susceptible to competitors that may be able to produce higher quality software (due to less testing required at the customer's site).

Higher quality software also decreases the cost of using the software for customers, as each problem encountered may contribute to downtime. This problem can be further compounded when patches or fix packs are applied to systems to address the problem. Each time a fix is applied to a system, there is downtime for the system that results in additional costs for the group applying the fix.

Software development firms should build trust with customers to alleviate the burden of testing software. Communication of process changes is a step in that direction, along with input from customers about their specific environments and tests that they perform before deploying. According to Redmonk analyst James Governor, marketing agile practices may be forthcoming:

"We've seen vendors using agile. Now we may be seeing the second stage, where they're marketing their use of it.[35]*"*

3.3.3 Improved product perception

Communicating that organizations use agile software development methods may also bolster the standing of the organization and their products. Organizations may benefit from the perception that they're leaders in the industry by using new development technologies. Customers also feel that they have much more impact on the product (due to their input and more frequent dialogues). As a result, customers may be more likely to become advocates of the products that they 'helped' to develop. Both IBM and Kronos have been communicating the agile development method to their customers. Aside from customers that participate directly in the process (via Beta versions, and providing feedback and having discussions with the engineering teams), the discussions have typically taken place at higher levels within the organizations. It's not clear if sales and marketing teams have been able to clearly and effectively communicate the advantages that agile development brings. At some point in the future however, agile development methods may simply allow development organizations to establish an even playing field with other development groups as agile development becomes more widely adopted.

3.3.4 Attract capital investment

For startup software companies, adopting agile software development methods may help attract the interest of venture capital funding. OpenView Venture Partners, a venture capital fund requires that software companies it invests in use agile software development. Jeff Sutherland, co-creator of SCRUM serves as a senior advisor to the venture[36].

3.4 Questions to ask before transitioning

Having seen changes and impacts that an agile development transition can have on an organization, the following questions present a guide to determine how fit an organization is to adapt to agile development. Prior to undertaking the transition to agile software development, managers should consider the following (incomplete) list of

questions to determine areas that may need additional focus during the transition and prepare for otherwise unanticipated changes and challenges. These questions are based on observations of actual changes and practices at the organizations previously discussed, including the referenced literature.

Should the entire organization undertake the transition to agile?

It is possible to have pilot agile adoption successes, but as was shown, there are considerations that need to be made. Pilot projects need to be chosen carefully, as failure can stop future transitions for the entire organization. Likewise, transitioning an entire organization also requires careful consideration.

Does lean/agile work for very large products being developed?

Can the requirements/needs be broken down into small, manageable modules without losing sight of the whole feature? Depending on the length of iterations, larger requirements may need to be broken down into sizes that are both meaningful and can fit within the length of an iteration cycle.

Is there a legacy product that this product is based on?

As shown through the discussions with the two organizations above, legacy products may impose additional considerations for organizations such as: dividing time between maintaining the existing product and developing new products, what existing features within the product need to change to meet new feature requirements, and what existing features 'cost' too much to maintain.

Are third party products or other organization's code incorporated into the product being developed?

If the organization is dependent on non-agile organizations for components or pieces of the code, additional planning may be needed to ensure that the necessary components are received when required.

Are those companies using agile development methods? How will you handle dependencies on these organizations?

Backup plans may be needed in cases where there is a dependency on an organization that does not use agile development. For waterfall organizations, the requirements are typically stated well in advance and are difficult to change quickly.

Do I have executive level support within the organization?

As many of the companies I've looked at had their agile transition initiated at the executive level, it's important to have the proper backing. This is to ensure that teams will commit to the transition, and also that upper management is aware of what teams are producing, and what to expect when looking for project status.

What resources are available for the transition? (How long will those resources be available?)

Some examples include coaches, both internal and external, agile practitioners within the organization, and consultants. It is important to make the resources available as needed, considering the effects of having the resources on other parts of the organization.

Should I make Beta/Milestone/early versions available for testing?

Part of the agile development process is to provide customer's with value by creating the features to meet their needs. Since customer needs may change, or needs may be misinterpreted, it is important to provide versions for customers to 'preview' the features and provide feedback on them. Each build should meet quality requirements and identify the new features and status of the features.

How do I gather the feedback provided by those testing the product?

Customer feedback should be gathered to determine how successful the new features and quality of the product is. For large partners where relationships have been formed, one on one discussions around the iteration cycles seem appropriate. For larger audiences,

web-based feedback postings are appropriate, but the information needs to be analyzed after gathering, and not just allowed to be stored in a 'black hole'.

Are there identifiable customers that can and are willing to participate in an early product release program?

There needs to be a benefit for both parties. Identifying customers that can benefit from the new features, particularly existing customers that will understand the value proposition are likely to participate if they sense that the value the product brings will increase.

How often should I release a version of the product?

Factors that need to be considered are: Are sufficient features included in the product to enable it to sell? How often will my customers be able to upgrade? Are the support and maintenance teams prepared for several releases in the marketplace at one time? Is the sales team able to handle frequent releases?

Do I have the right agile team members in place to account for role changes?

This covers managers, developers, technical leads, and testers. As shown earlier, roles for individuals can change. The transition can be made easier if the right people are in the right roles with the flexibility to adapt to the changes.

Are there other aspects of the organization that need to be changed to be successful with agile development (e.g.: HR – rewards, marketing, legal, management)?

The rewards system should be aligned with the goals of the organization. If the goals are aligned to reward team achievements, this may provide greater motivation to change. As seen at both IBM and Kronos, a portion of the goal system is focused on team efforts.

For legacy products, how can you minimize interruptions for urgent customer issues?

It is important to keep current customers happy while maintaining existing software products. As a result, consideration of allocated time or team members to work on critical customer issues may help minimize interruptions and allow management to plan for available resources (e.g.: 80% time allocation).

3.5 Questions to ask during the transition

While the agile transition is taking place, organizations should of course monitor the progress to make sure that the change is on the proper path. Periodic organizational meetings to provide status updates on the progress help reinforce the transition message. This is especially important, as improvements from the transition in the cases observed took around six months to appear. The following questions serve as a guide during the transition.

Is code being produced code too quickly?

Does the pace of the code that is being produced meet the business goals for shipping products? If the pace is too fast, then there will be excess effort and costs spent to develop software that won't be shipped yet. If the pace is too slow, then the business may fail getting the product to the market when needed, and potentially lose sales to competitor offerings.

Are teams taking on too much work?

Managers need to ask "Is the pace sustainable?" Teams can push through more work for a short period of time, but it is difficult to sustain an extended effort for a long period of time. Requirements planning must be based on realistic production capabilities that an organization has. If the teams are producing too fast, it may not be sustainable by the organization and provide false expectations for future performance which at some point will fail to be met.

Are the right people in the right roles?

As previously discussed, role changes for some members of the organization should be anticipated. Management (various levels) need to ensure that team members are in the proper positions, as some individuals may not like the new role requirements once the transition is under way.

Is the proper message being sent (and received)?

Team based effectiveness may require a change in the rewards system to reflect the agile development goals. Since the focus is on the customer, teams must work together to satisfy their needs. This may entail employees in one role helping another member of the team in a different role, to achieve the goal of properly developing and testing the new features. This behavior was exhibited in the Domino organization by developers who assisted the test organization with automation testing.

Are the methods working, or do they require modification?

An organization may find that some practices don't fit well in their organization. This may require further analysis to determine if there are barriers that need to be removed, or if there is another, more suitable practice to fit the organization and development goals. As seen at Kronos, it is important to make sure there is a 'baseline' of agile development methods so all parties have a common understanding. This makes it easier for subsequent organizational changes such as employees transferring into a new team or group within the agile organization.

Appendix 1 – The Agile Manifesto

Principles behind the Agile Manifesto[37]

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development.
The sponsors, developers, and users should be able
to maintain a constant pace indefinitely.

Continuous attention to technical excellence
and good design enhances agility.

Simplicity--the art of maximizing the amount
of work not done--is essential.

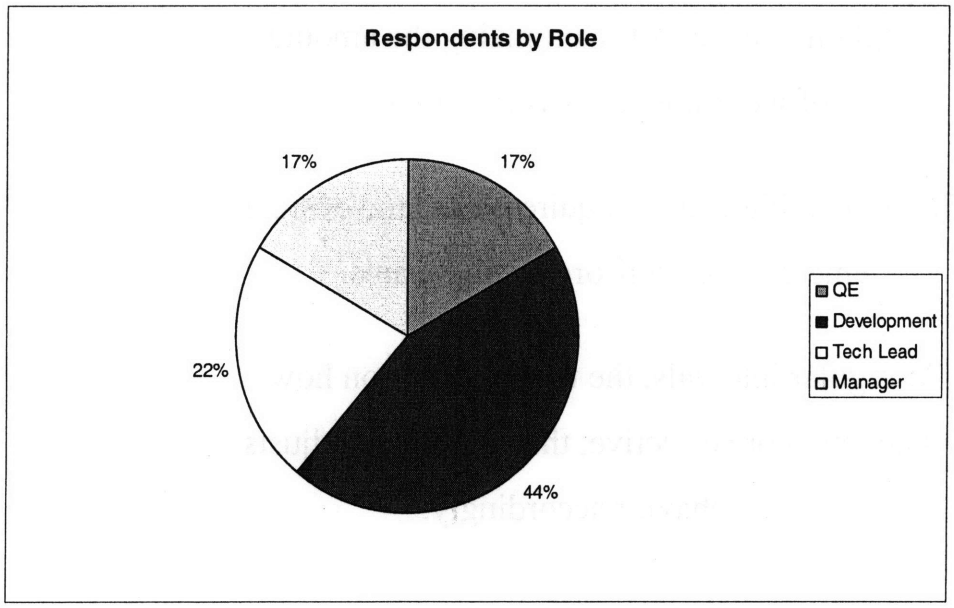
The best architectures, requirements, and designs
emerge from self-organizing teams.

At regular intervals, the team reflects on how
to become more effective, then tunes and adjusts
its behavior accordingly.

Appendix 2 – IBM survey results

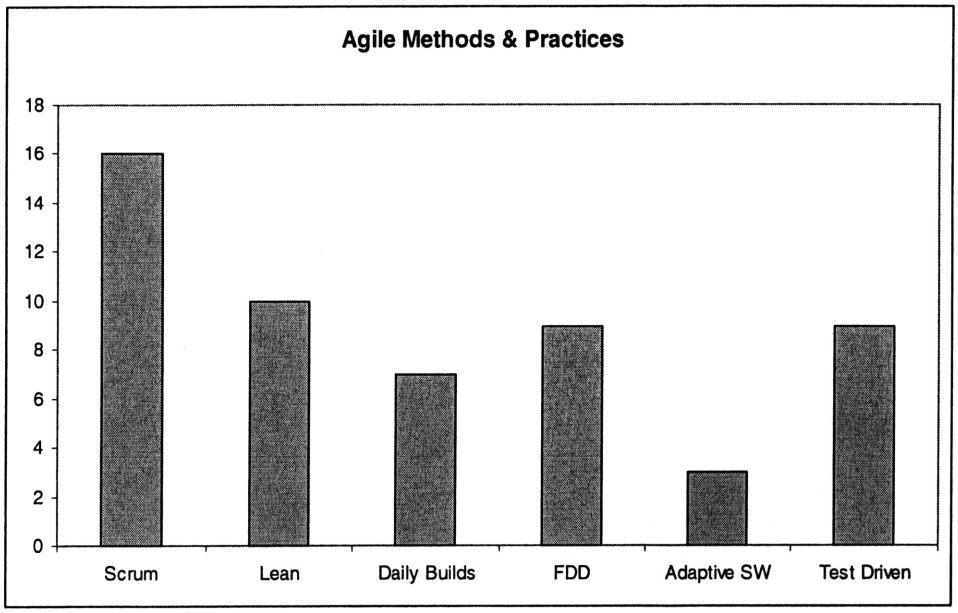
What is your role in the organization?

QE (17%), Development (44%), Tech Lead (22%), Manager (17%), Other (please specify) (0%)



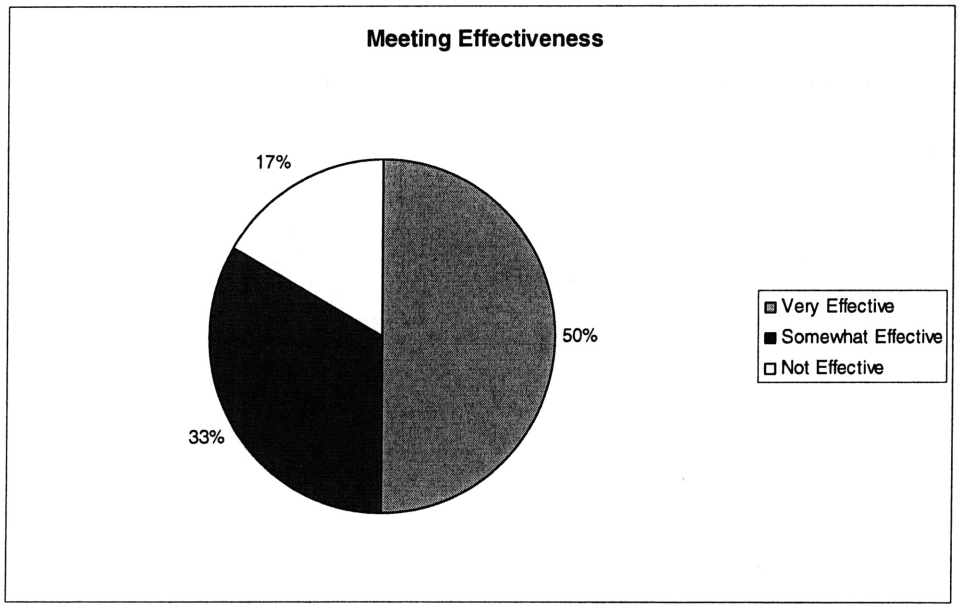
What agile methods and practices are used in your team? (Select all that apply)

Scrum (94%), Lean (53%), XP (0%), Paired programming (0%), Daily builds (35%), Feature Driven Development (47%), Crystal (0%), DSDM (0%), Adaptive Software Development (18%), Test Driven Development (47%), Other (please specify)



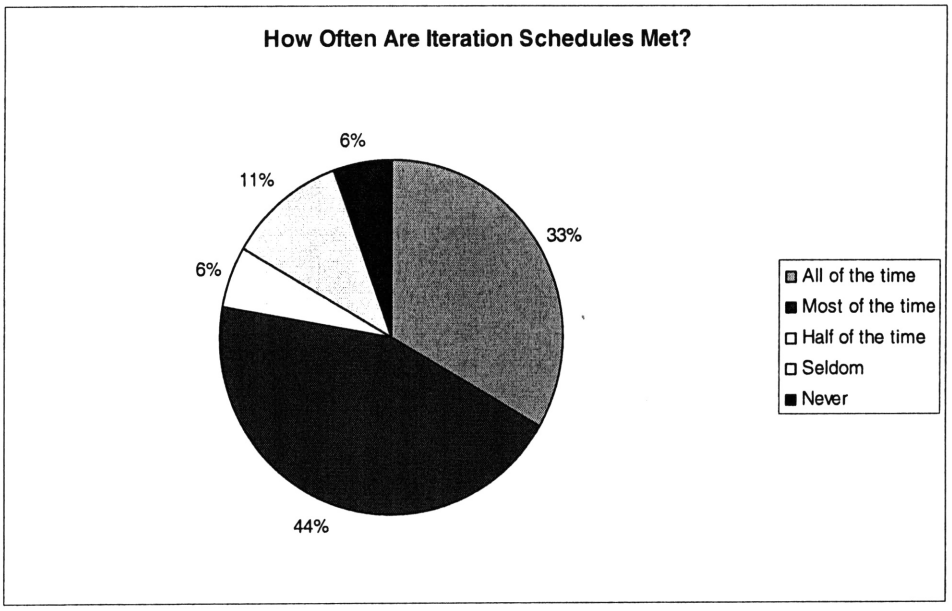
How effective are the team meetings at achieving goals?

Very effective (50%), Somewhat effective (33%), Not effective (17%)



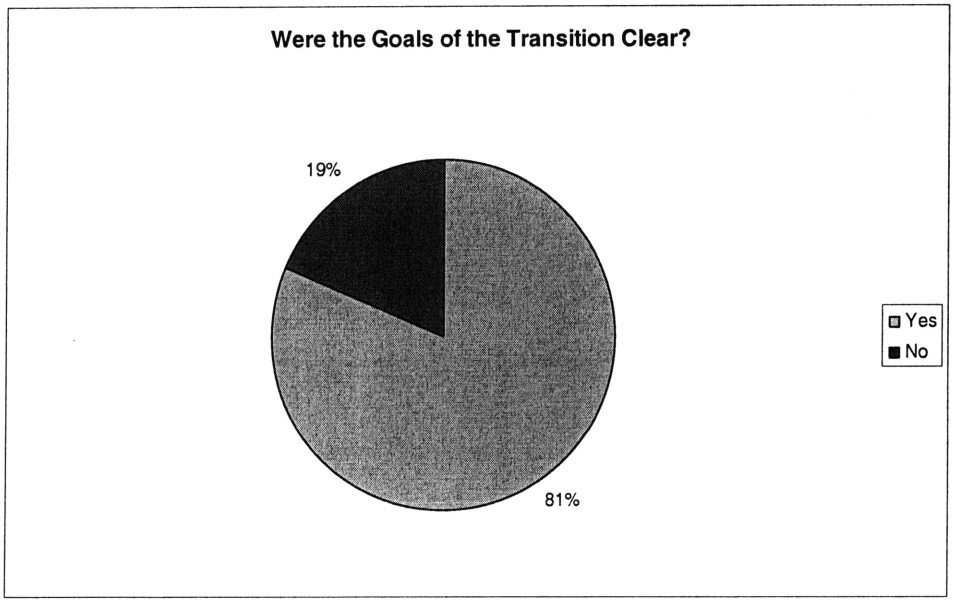
How often are the iteration schedules met?

All of the time (33%), Most of the time (44%), Half of the time (6%), Seldom (11%), Never (6%)



Were the goals of the agile/lean transition clearly stated?

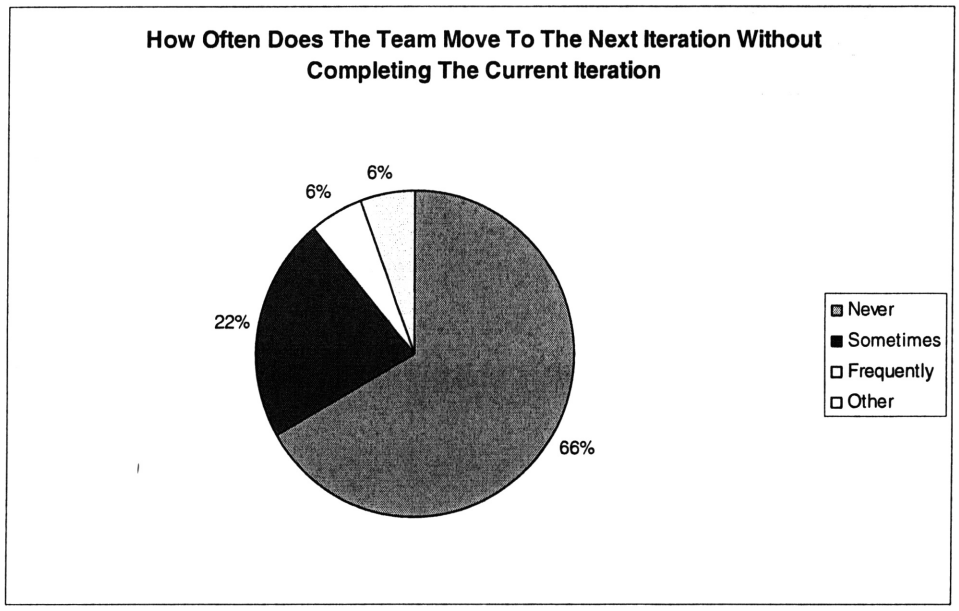
Yes (81%), No (19%)



How often do you or the team move on to the next feature/project without completing the current feature iteration?

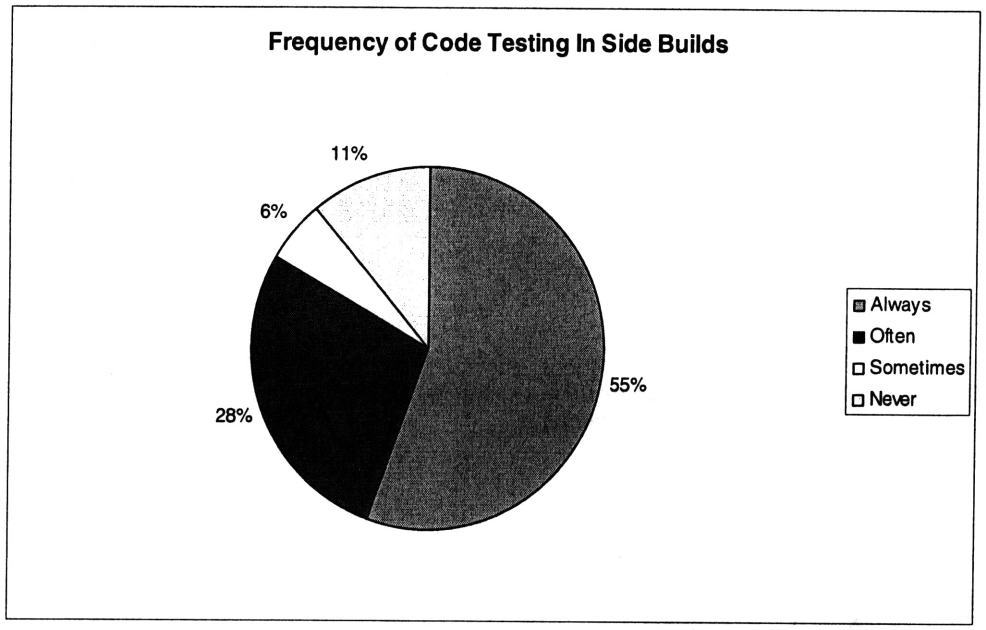
Never (66%), Sometimes (22%), Frequently (6%), Other (please specify) (6%)

The Other comment stated that if the requirement wasn't completed in the current iteration, it was completed in the next iteration.



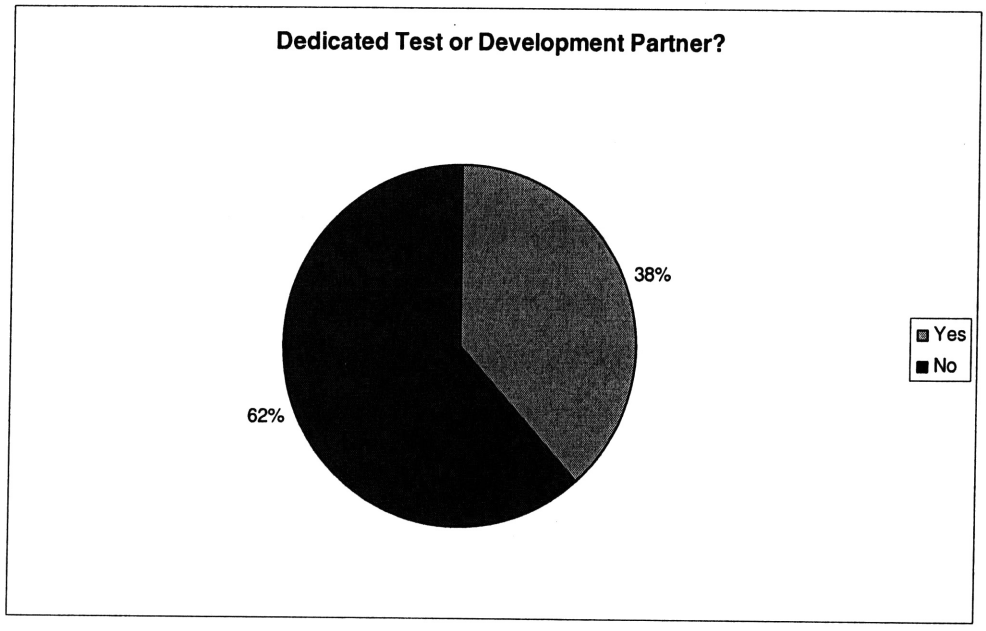
How often is your code tested in a side-build prior to submitting into the main build stream?

Always (55%), Often (28%), Sometimes (6%), Never (11%)



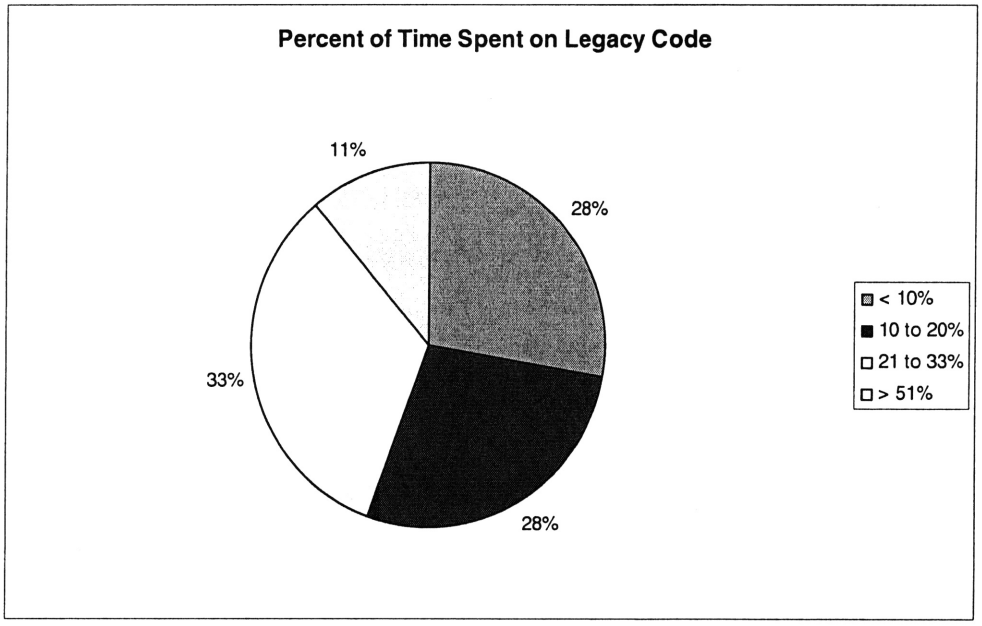
Do you have a dedicated developer or test partner (For Developers and QE only)?

Yes (38%), No (62%)



How much of your time (on average) is occupied by maintaining legacy code or customer problems?

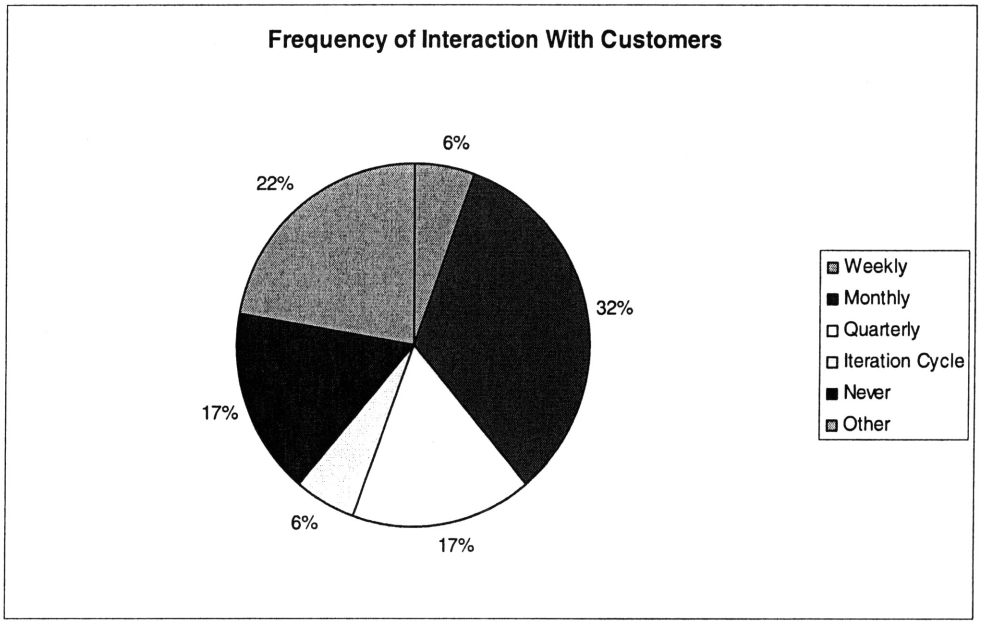
0 to 9% (28%), 10 to 20% (28%), 21 to 33% (33%), 34 to 50% (0%), 51%+ (11%)



How often do team members interact with customers about the feature/project?

Weekly (6%), Monthly (32%), Quarterly (17%), Iteration Schedule (6%), Never (17%), Other (please specify) (22%)

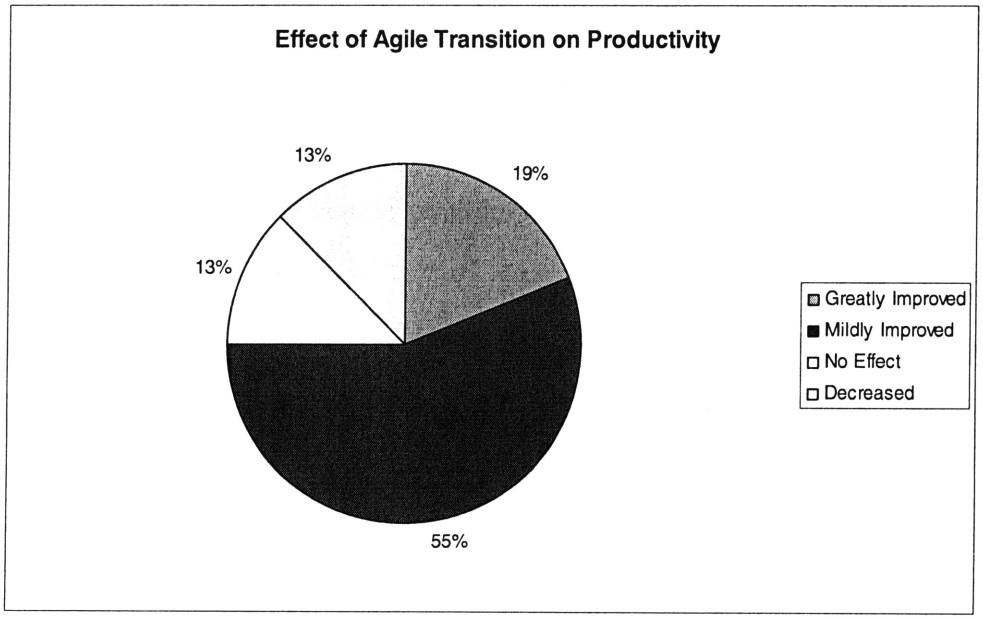
The 'Other' responses indicated that interaction occurred as needed via electronic forums, and that only specific individuals on the team directly interacted with customers.



How has the lean/agile development initiative affected productivity?

Greatly improved productivity (19%), Mildly improved productivity (55%), No effect on productivity (12%), Decreased productivity (13%), Other (please specify) (13%)

The 'Other' responses indicated that fewer defects are being written, and that the lean/agile adoption was unproven as the product was still under development.



There were three open ended questions contained in the survey:

In your opinion, what is the biggest benefit of using agile development methods?

The responses were:

"More time spent developing and less time spent writing a spec up front when you don't know all the issues yet".

"Daily scrums. Teaming."

"Early and continuous feedback."

"Focus on writing code and automated tests is good."

"None seen yet."

"Being able to focus and see clear progress on a goal."

"Reduced overhead, more tightly focused on actually delivering small increments of working functionality rather than code that may never be finished and may not meet the requirements even if they are finished."

"Better quality with mandated testing."

"Higher quality, more consistent productivity."

"Test driven development, when it is really practiced. Getting early customer feedback is a big benefit as well."

"Building test driven development where quality is built in from the start"

"There seems to be fewer defects in development's product."

"Increased personal and team accountability..."

"...much better understanding of the strength and weakness of my teammates."

What is/has been the most difficult issue to overcome for successfully adopting agile development?

The responses received were:

"managing technical debt"

"...getting pulled in multiple directions, not able to put dedicated focus on this project."

"dealing with legacy issues... and other things which are not part of a given iteration."

"Transition is very painful, feels bad"

"Load balancing between iteration and other work."

"Merging code into the main build stream can be time consuming."

"Preventing operation in a waterfall fashion"

"Perception that it is taking longer"

"Proper planning and measurement of progress towards end goal"

Please add any other comments that are relative to the agile development adoption process.

"Some tasks and design issues do not fit well into short iterations."

"The company has a process-oriented culture; it is difficult to convince people to give up wasteful processes, measurement, and other familiar 'waste'."

"It feels like we are always going full speed, watch out for burnout."

Bibliography & Further Readings

Larman, Craig - Agile & Iterative Development – A Manager's Guide

Poppendieck, Mary and Tom - Implementing Lean Software Development – From Concept to Cash

Cohn, Mike - Agile Estimating and Planning

Poppendieck, Mary and Poppendieck, Tom - Lean Software Development – An Agile Toolkit

Cusumano, Michael A.- The Business of Software

Cusumano, Michael A. and Selby, Richard W. - Microsoft Secrets

Lean Enterprise Institute - http://www.lean.org/

Agile Journal - http://www.agilejournal.com/

Windholtz, Mark - Case Study of Lean Software Development - http://www.objectwind.com/papers/LeanCaseStudy.htm

Windholtz, Mark - Lean Software Development - http://www.objectwind.com/papers/LeanSoftwareDevelopment.html

Agile Community Queue - http://www.infoq.com/agile

Agile Advice - http://www.agileadvice.com/

References

[1] http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
[2] http://en.wikipedia.org/wiki/Waterfall_model#CITEREFRoyce1970
[3] Agile & iterative development: A Manager's Guide, Craig Larman, Addison Wesley, 2004
[4] Iterative and Incremental Development: A Brief History, Larman & Basili, IEEE 2003
[5] Iterative Enhancement: A Practical Technique for Software Development, Basili & Turner, IEEE Transactions on Software Engineering, December 1975
[6] A Spiral Model of Software Development and Enhancement, Barry W. Boehm, May, 1988
[7] Dr. Dobbs Journal, August, 2007, "Survey Says... Agile Has Crossed the Chasm" http://www.ddj.com/architect/200001986?pgno=1
[8] Standish Group, 2004 Chaos report
[9] ibid
[10] Watts S. Humphrey (founder of the Software Process Program of the Software Engineering Institute at Carnegie Mellon University)
[11] Microsoft Secrets, , Michael A. Cusumano & Richard W. Selby, Touchstone, 1995
[12] Agile & Iterative Development, A Manager's Guide, Craig Larman, Addison Wesley, 2004
[13] Jeff Sutherland and Mary Poppendieck, Deep Lean Seminar, November 2007
[14] Standish Group Study, 2002
[15] InfoQ Interview: Jim Johnson of the Standish Group, 2006 http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS
[16] The Standish Group International, 2004 Third Quarter Research Report
[17] ibid
[18] Lessons Learned from Modeling the Dynamics of Software Development, Abdel-Hamid and Madnick, Communications of the ACM, December 1989
[19] MIT course notes for ESD.74 System Dynamics, 2007, James M. Lyneis
[20] "Implementing Lean Software Development", Mary and Tom Poppendieck, 2007
[21] ibid
[22] Niklaus Wirth, "A Plea for Lean Software", 1995 http://cr.yp.to/bib/1995/wirth.pdf
[23] Interview with Al Eldridge, employee #5 at Iris Associates
[24] Frederick P. Brooks, Jr. "The Mythical Man-Month", Addison-Wesley, 1975
[25] Jeff Sutherland, http://jeffsutherland.com/2003/03/scrum-get-your-requirements-straight.html
[26] Agile: Adopting a New Methodology at Harvard Business School, Borges, Gilmore, & Oliveira – Agile 2007
[27] From Waterfall to Agile – How does a QA Team Transition? Megan Sumrell, Agile 2007
[28] http://www.kronos.com/
[29] "Empowering Product Development" by Pete Deemer, 2007 http://www.cxotoday.com/India/Future_Technology/Empowering_Product_Development/551-83194-907.html?topic=306111
[30] "Implementing Lean Software Development", Mary and Tom Poppendieck, 2007
[31] Large Scale Agile Transformation in an On-Demand World, Fry & Greene, Agile 2007
[32] IBM Making Agile Moves, Chris Kanaracus, November 19, 2007, http://www.infoworld.com/article/07/11/19/IBM-making-agile-moves_1.html
[33] Enabling Agile in a Large Organization Our Journey Down the Yellow Brick Road, Thomas R. Seffernick, Agile 2007
[34] Discussions with Jeff Sutherland and Nancy Van Schooenderwoert
[35] IBM Making Agile Moves, Chris Kanaracus, November 19, 2007, http://www.infoworld.com/article/07/11/19/IBM-making-agile-moves_1.html
[36] Jeff Sutherland, Deep Lean Seminar, November, 2007
[37] http://www.agilemanifesto.org/principles.html