

IBM Global Business Services
AMS

Executive Report
October 2013

Applications Management

Distributed Agile

Accelerating Applications Delivery in a Geographically Distributed Environment
Using Agile Methods

A 3D rendered robot character, primarily white with blue accents, standing in a stylized, low-poly landscape of green and yellow hills. The robot has a humanoid form with a head, torso, and limbs, and is positioned on the right side of the image.

The
SmarterADM
Series

Mobile devices, collaboration tools, and cloud solutions are a few of the many emerging technologies which are rapidly changing the way organizations compete in virtually every industry. Effective software delivery represents a significant opportunity to improve a company's ability to compete.¹ And now, more than ever, the speed at which software can be delivered - developed, enhanced and maintained - is of vital importance to the business. Agile, and more specifically, agile in a distributed environment where team members are not collocated, is receiving increased attention as one of the important ways in which organizations can accelerate software delivery.

Agile software development - frequently positioned as the contemporary replacement for traditional plan-driven (waterfall) development methodologies - approaches software development from a social perspective and advocates close collaboration over strict processes. The goal is nothing less than the rapid creation of applications with superior code quality and at a lower overall cost.²

Past studies suggest that agile projects work best when the size of the project team is relatively small, the entire team is collocated (including the customer), and the need to respond to change is more important than predictability and stability. Nevertheless, large organizations tend to execute projects that fall outside of this agile sweet spot; large, complex, and geographically dispersed project teams appear to be the norm. This reality makes agile adoption difficult: organizations cannot meet, or do not wish to meet, the collocation requirement.

Speed... is of vital importance to the business.

Globalization in general, and the economic advantages of labor arbitrage, in particular, have further tended to create widely distributed technology teams. Indeed, the inability (or lack of desire) to collocate staff sets up an internal barrier to the adoption of agile practices that can stop the movement in its early stages. This is exacerbated by the knowledge that, regardless of methodology, large teams have difficulty cooperating and working as a single unit—particularly when they do not work in the same location. Additionally, as team size increases and the team becomes more geographically dispersed, communication overhead goes up nonlinearly while the ability of the team to deliver only goes up linearly.³

Despite the real and perceived challenges to implementing agile with geographically distributed project teams, it has been successful and has created significant value. Several current case study examples cited in this paper suggest that distributed agile organizations can approach a 50% improvement in cycle times and a substantial (up to 40%) reduction in defects over traditional methods, depending on the practices implemented.

***50% Faster
40% Less Defects***

Although more subjective and not as uniformly measured, improved client satisfaction was also reported. While the implementation characteristics were different, each relied on a combination of traditional human interaction and social collaboration tools to make up for the absence of sustained in-person face-to-face collaboration. This suggests that large organizations can benefit from agile principles and practices even with the difficulties introduced by a team's geographic dispersion.

The Agile Evolution

In software development, the term *agile* applies to those development methodologies that are lightweight; involve customers to create, prioritize, and verify requirements; and depend on the team's tacit knowledge over heavy documentation. They embrace change, frequent delivery, simple design, test-driven development, and frequent feedback as core practices. While agile software development methodologies are frequently positioned as the contemporary replacement for more traditional approaches, many of the practices—developing iteratively, creating incremental releases, and continuing to evolve a product at each step—have their roots in theories developed in the 1930s.

Despite this history and success, the industry began to favor a more rigorous, systematic approach in the late 1960s and 1970s. This is due, in part, to the inability of the then-dominant ad hoc approach to scale to large, complex systems. The resulting plan-driven methods sought to combine product development discipline with process structure to improve the creation of software.

This era also gave rise to the *waterfall methodology* (so-called because the visual depiction of its phases resembles a waterfall; see Figure 1) which set out to improve ad hoc development efforts, but actually did not set out to replace iterative development with the conceptually simple model of Figure 1.⁴ Instead, the original proposal was for an integrated process that has iterative and potentially overlapping stages with the possibility for early release while retaining close customer involvement (see Figure 2).

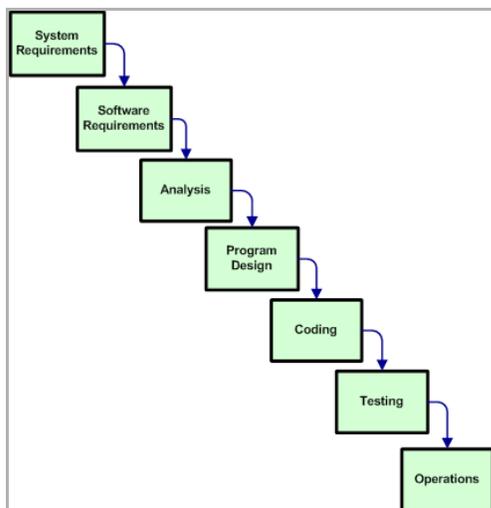


Figure 1

Nonetheless, the more simplified version became the prevailing methodology—this may be due to its conceptual straightforwardness. A focus on engineering brought with it the application of project management techniques that use defined processes, validation steps, and statistical control to manage software development.

By the mid-1980s, it appeared that software development was still in crisis. Schedules, cost, and overall quality were still unpredictable in many organizations. In response, the U. S. federal government began to look for a way to

assess the capability and maturity of its software contractors. This resulted in the Capability Maturity Model for Software (SW-CMM) and, much later, the Capability Maturity Model Integrated (CMMI).

The methodology proposed by the SW-CMM and CMMI to improve an organization's development practices is to identify a series of maturity levels that an organization can attain through the implementation of best practices. The release of CMMI firmly placed software development into the same category as engineered products. Indeed, the focus for the decade prior to the rise of agile was on process improvement - making software development less costly through improving its quality.⁵

While successful in many organizations, the engineering approach had not lived up to its full potential. Perhaps this is due to the nature of software development itself; unlike other engineering disciplines, one cannot build scale models or demonstrate the key processes involved with its creation. Furthermore, once produced, software is constantly subject to changes in order to adapt it to different circumstances and environments.

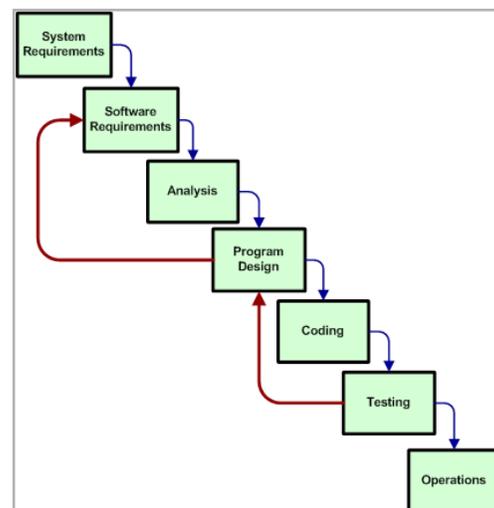


Figure 2

In the late 1980s and throughout the 1990s, several methodologies began to gain attention as an alternative to these strict product-based approaches.

Ironically, these alternative approaches represented a return to thinking that was

prevalent during the early and mid-1960s when large and complex software was in its infancy.

Fred Brooks, recounting his experiences with the early 1960s development of OS/360 software in his “No Silver Bullet” essay, identified the need to discard the building metaphor because he found that teams could incrementally create much more complex software than they would be able to build using a strict engineering approach.⁶

Others began to notice these same phenomena and proposed methodologies that advocated piecemeal or incremental growth. Along with the recognition that incremental approaches may be more natural and that they could produce more complex systems, there was also a renewed appreciation that there is a significant interpersonal and social aspect to creating software. That is, people and how they interact can determine whether a software project is successful or not, regardless of the process or technologies being used.⁷

The people issues shine through...

Some began to draw parallels to work in the field of architecture identifying patterns that made a complex project effort successful or unsuccessful. Not surprisingly, many of the resulting patterns focused on a team’s social interactions.

At the same time, the technical underpinnings of software development were beginning to change; the structured programming paradigm was rapidly giving way to a more object-oriented focus - particularly in browser-based applications accessible from the Internet or a corporate intranet - facilitating a more partitioned, incremental, agile approach.

Agile software development has garnered increasing attention of late due to the confluence of these various trends and *agile* has become something of a buzzword in the industry today.

Nonetheless, it has as its base the social principles that were articulated years ago. Perhaps, because of the rapid evolution of social software and collaborative technologies, these seminal agile principles have become significantly more implementable of late. The Agile Manifesto’s declaration of “individuals

and interactions over processes and tools” with a goal of working software, is seen as the antithesis to the rigid procedures of a strict engineering approach.⁸

Looking deeper, the agile movement can be seen as a correction to an overreliance on engineering procedures in a field where social aspects are an important part of the overall development process: the engineering paradigm is not enough.

Agile Adoption Focus Areas

While agile software development shares the goal of working programs with its more engineering-centric predecessors, there are also some notable differences.

Some of these differences have drawn criticism; such as a lack of up-front design, an unawareness of the system-wide perspective, a reliance on tacit knowledge, and pair programming. Indeed, despite several studies that appear to empirically prove that using these methodologies will yield predictable results and provide value under the proper circumstances,⁹ there can still be resistance within organizations due to the significant change from traditional methods that it represents.

The focus on managing the inevitable change that seems to accompany software development appears logical on the surface. Nevertheless, when confronted by existing process inertia and general resistance to behavioral change, adoption can be a challenge.

With this background in mind, our experience adopting agile extensively for IBM's internal portfolio of applications and our experience with a number of clients around the world, discussed in the next section, leads us to the following observations and conclusions regarding the key elements of agile methods. Although we touch on several elements of agile, we concentrate on distributed (i.e. non-located) teams, which is the focus of this paper.

1. Team Size

Agile software development relies on the tacit knowledge of team members, which *could* limit its ability to scale to larger teams. While team size is a factor in many of the agile approaches, the underlying perception that agile is *always not suitable* for large teams is not correct.

Research suggests that using agile software development is possible and beneficial for larger teams, but it requires that the project be subdivided and that it implement several plan-driven techniques to coordinate the multiple sub-teams. These accommodations have helped larger teams retain many of the benefits of agile.

2. Criticality

Many in the software delivery industry assume that agile software development is usually not a consideration when a project is producing a safety-critical product. The lack of extensive documentation and the use of simple designs *may* be a poor fit for such systems. The reality is that agile had been largely untried in these areas. There have been some success stories, although the assumption is still widespread, and it is currently unclear whether this consideration has any merit.

3. Skill Level

Agile software development demands team members with more experience than a more traditional, plan-driven approach. Perhaps this is because the tacit nature of information flow demands a higher level of expertise.

The implication is that experienced personnel are more difficult to find, train, and retain. And, they are also often more costly than those with less experience. Nevertheless, organizations should seed agile teams with the best and brightest. Use an agile coach - but each agile team will benefit greatly from the experience and expertise of the experienced member(s). Given the time-to-value and other economic benefits we are seeing, the return on this investment is generally clear.

4. Requirements Stability (Dynamism)

Agile is well suited to projects that have changing requirements, as they are designed to allow a change in course on a regular basis. If a project has a relatively rigid set of requirements and little possibility of change, the cost of optimizing the project for such change can be burdensome. For highly stable products, using an agile methodology can be a source of expensive rework. Agile methodologies freely acknowledge that their very design biases them toward accommodating change.

Given the accelerating emergence of mobile, cloud and related technologies and their potential impact on applications requirements, it is likely that greater and greater portions of the enterprise portfolio will need to accommodate changing requirements, making agile all the more compelling.

5. Culture

Culture may be the one true limitation of all methodologies—that is, it is not specific to agile. If an organization thrives on empowerment and many degrees of freedom, agile would seem to be a better fit. On the other hand, a culture thriving on rigidity may not feel comfortable with an agile methodology. Some resistance is inevitable if using agile is a change from a prior methodology. Moving from collocated agile to a more distributed version of agile may introduce similar resistance.

The social aspects of software development in general - as noted above - and specifically for agile adoption are greatly affected by the accelerated use of collaboration tools and techniques.

The profound societal changes we are experiencing from social collaboration on an unprecedented scale represent culture change at a global level. Channeling this cultural phenomenon into an enterprise culture change (adopting social collaboration as a fundamental part of the software delivery process) is still challenging, but should be viewed through the lens of broader systemic change - not simply as part of a methodology adoption.

6. Distributed Teams

Our introduction identified lack of collocation as a frequently seen barrier to implementing agile software practices. It is not surprising that this is one of the perceived limitations of the methodology. This perception is held in part due to one of agile's underlying concepts: that in order to remove the communication and social barriers that can cause a project to fail, team members (customer/users to programmers) should be physically collocated.

Distribution of the team (non-collocation) does, in fact, introduce new challenges, but it is not only possible to use agile in this environment, it

is being done globally on a large scale. It can be quite successful and provide substantial benefits in speed, quality and cost reduction.

6.1 Team Communications

Splitting software development tasks across the members of a distributed team is not always straightforward. Teams often need to share information both synchronously and asynchronously. This can lead to problems with how the team views the information across distance and time, including undesirable conditions such as message latency, unexposed team communications, and misunderstandings caused by the number and type of communications channels a message needs to pass through.

Nevertheless, the majority of these communications deficiencies can be (and are) accommodated through tooling. Information radiators and low latency, for example, can be accommodated through collaborative tools such as wikis or instant messaging while sound and visuals can be made available through teleconferences, shared web conferences, videoconferences, and even shared white boards.

The explosive growth of collaborative environments is fueling the growth and increasing the capabilities of collaborative technologies. What is important is to deploy these tools consistently across the teams and make their adoption and thorough usage part of the agile adoption plan.

6.2 Work Coordination

The ability to coordinate work is a key variable for determining whether a team is successful. This can be difficult for distributed teams because communications complexity and distance can reduce the effectiveness of coordination efforts. On the other hand, distributed software development teams exist in most large organizations regardless of the methodology employed. Those that are successful rely on trust and a shared communication system to overcome coordination issues.

Here again the technology platform chosen plays an important role in work coordination. And great technology is key here. Email, spreadsheets

and the work-coordination patterns ingrained over the past few decades will not enable agile teams. True work item level collaboration is required. Further, the element added here is the element of trust. Creating a trusting relationship between all the participants - aided by the work-coordination tooling - not only accelerates the process in total, but specifically serves to substantially reduce communications hurdles. (More on trust below.)

6.3 Group Awareness

Group awareness refers to an individual's knowledge as to who is on the project, where in the code they are working and what they are planning to work on next. Putting people in the same room can have a significant positive impact on their productivity simply because they have an awareness of other's activities; they also have a real-time understanding of the group's interactions.

An important element of a well formed collaborative environment for agile incorporates pervasive transparency as a core practice (as described below) and enables, in a meaningful way, this notion of group awareness. Existing, state of the art, collaborative development tools can be used to successfully overcome many of the deficiencies in awareness. Deployment of these tools requires a coordinated, dedicated program for adoption across the distributed agile teams. Blogs, wikis and forums, combined with an active policy of pervasive transparency will help create an agile-aware team, even when its members are globally dispersed.

6.4 Creating and Maintaining Trust

Trust between team members increases with the number of positive interactions that they have. The basis of team trust is shared values and common goals. Its absence can increase cost and impact the quality of information exchange.

Research suggests that teams and individuals with a low level of trust are prone to exhibiting negative behaviors which further increase distrust. While more difficult to master in a distributed environment than with collocated teams, distributed teams are also able to build trust between individuals: they must prevent their geographic distance from leading to psychological distance.

Communities... socially connected...are quantifiably higher performing

In IBM's model for agile adoption, extensive use is made of collaborative communities built around a technology platform that enables deep social interaction and pervasive transparency. It is this latter element of complete transparency which builds trust. Communities with higher levels of trust among members, and whose members are socially connected to one another, have quantifiably higher performing teams.¹⁰

6.5 Culture and Multinational Teams

Geographic distribution introduces the likelihood that a team will cross country and cultural boundaries. The way members of different cultures think about their relationships with a supervisor or teammates will affect the team's overall performance—particularly if the cultural norms between team members clash.

While it takes a longer time for distributed teams to develop trust, cultural issues remain less of an issue, ironically, because all of the modalities of communication are not available, which may reduce the number of cultural clashes which otherwise might occur. On the other hand, cultural issues may become hidden beneath the tools that distributed teams use to communicate.

It is currently unclear whether this dampening of culture is positive or negative in the long term. But what is clear is that some form of open, transparent collaboration platform which connects the community/team member, facilitates communication and work coordination, increases awareness and engagement by all parties and creates a virtual work space for all, is an important bridge for cultural differences in multinational teams.¹¹

Case examples

Distributed agile software development is not an all or nothing proposition. The distributed agile challenges that have been discussed can be addressed in different ways depending on the project, team and tooling alternatives.

We have identified four types of distributed agile development that we use today both internally (for IBM's own portfolio of enterprise applications) and with clients around the world:

- Full distribution
- Congruent time zones
- Periodic collocation
- Simulated collocation

The interesting phenomenon is that all have proven to be successful. Which technique is used appears to be driven by a variety of inputs including economic considerations, corporate culture, and cultures of the regions where development teams are located.

The common thread is that for those companies that really want to move toward agile, the "whole team" figures out a way to get there despite the distributed nature of their development teams.

Full Distribution

Full Distribution is perhaps what most people have in mind when they think of distributed teams. All of the project resources can potentially be located anywhere in the world, in any time zone.

Within IBM, this is the approach used most often. When we began our aggressive deployment of agile several years ago, our corporate initiative to be a true, Globally Integrated Enterprise¹² had already organized large portions of our professional staff in India, Brazil, China, and many other countries, at multiple locations within some of those countries, covering multiple time zones. Collocation was simply not an option and our teams had to figure out how to do agile in a Globally Integrated Enterprise.

IBM mitigated the impacts of this highly distributed environment by enabling social development tools such as Rational Team Concert (RTC) and IBM Connections. These tools allow the team members to collaborate asynchronously both in the context of the development work being done (RTC) and for more informal team collaboration using IBM Connections. Within IBM, we have seen greater than 50% improvement in cycle time (as measured by function points delivered), 40% fewer defects (with Test Driven Development and Static Analysis techniques) and improved

satisfaction ratings from the business - in addition to substantially reduced aggregate cost.

A full description of IBM's "Smarter Application Development and Maintenance", including this community-based agile deployment can be found on IBM.com.¹³

Congruent Time Zones

Similar time zone distribution has arisen as a way to mitigate some of the risks and difficulties inherent with Full Distribution. Full Distribution often leads to very small periods of time where all the team members might be working at the same time. We have worked with many clients who have chosen to mitigate this possible risk by ensuring that their teams lie in regions that have at least 4-5 hours of overlap in working hours. An example of this is the overlap in time between India and Europe.

Collaboration tools such as RTC and Connections are still important in this environment - to create communities, effective collaboration, awareness, etc., as discussed above - but during the overlap hours, instant messaging and conference calls can be used as well.

...major telecommunications company in Europe... 50% improvement in cycle time

Utilizing this technique, our client, a major telecommunications company in Europe, was able to deliver significant new capabilities with as much as a 50% improvement in cycle time. The client also reported greater satisfaction at improved visibility on project progress.

Periodic Collocation

Periodic or continual collocation is another approach that is often used. In one variant of this approach, we worked with a client on projects that collocated a significant portion of the team at the beginning of the project for planning purposes and then returned those team members to their respective work countries.

This temporary collocation affords the team the opportunity to bond in ways that will carry throughout the project. We have also partnered with clients on projects where the collocation is

done in smaller sub-groups, but throughout the life of the project. One example: A client had business resources located onsite continually with their development team in China. They rotated the resources, but there was always someone on-site to facilitate communication throughout the life of the project.

The development team may collocate resources with the client's business teams to again ensure streamlined communication. One client, a medical equipment manufacturer in the United States, successfully used this technique. They not only improved cycle time, but also greatly improved quality through the use of techniques like Test Driven Development and Continuous Integration.

medical equipment manufacturer... greatly improved quality

Simulated Collocation

One of the most interesting distributed team approaches we have used relies on the use of technology to simulate collocation. This model extends the concept of a collaborative community to simulate true, physical collocation.

Our client, a major Australian financial institution, felt strongly about the benefits of collocation for agile, despite the distributed nature of their teams along with IBM professionals in China. Committed to agile and the return on their investment which agile would yield, they were willing to invest in advanced technologies to simulate collocation. They are extremely satisfied with the results to-date and we continue to pursue new technologies with them to further strengthen the real-time experience of their teams. The tools range from enhanced video conferencing to fully immersive virtual environments. They have credited simulated collocation for improving cycle time and project visibility.

Conclusions

- Distributed agile is being successfully done on a global scale
- The benefits in speed, quality and cost are compelling
- The social, collaborative aspects of software development are key
- Modern, state-of-the-art social and life-cycle work-item level tools are required
- Agile is a cultural phenomenon - not just a methodology

Software - development, enhancement and maintenance of enterprise applications - is rapidly becoming a key competitive differentiator for many businesses across most industries. An organization's ability to deliver applications - new applications, new features, improved performance - has increasingly significant business impact as this new, software-centric, mobile, cloud-based economy rapidly grows.

Agile methods for software delivery are getting a lot of renewed attention - and for good reason. The need for speed is here now, and for many segments of our various portfolios across a wide range of business applications, agile can deliver that speed - even with highly distributed teams.

Agile Development processes and techniques have had to evolve to support the changing landscape in the IT Industry. Global distribution of resources has driven the need for advancements in collaborative technologies to support *whole team* communication, which is a foundation for agile development.

There are a number of ways to approach agile in a distributed environment, as we have outlined in this paper. As technology changes, these approaches will likely continue to evolve and perhaps new approaches will emerge to take advantage of those technologies.

The important point is that distributed agile is already being done successfully and on a large scale. It requires discipline and perseverance along with adoption of some form of the tools and techniques described - but the results are compelling.

And the need for speed will likely increase for larger and larger portions of our applications

portfolios - as evidenced by the emerging trend for continuous delivery and Devops.

Many agile teams have matured to the point where they are able to deliver production-ready code at the end of each iteration, yet there are delays as they work through deployment processes to get the functionality into production. The goal of Devops is to have a completely integrated process between the Business, Development and Operations that allows code to be deployed to production within minutes at the end of an iteration if the business decides it is ready - and begin to immediately receive (and integrate) user feedback so that the end-to-end processes can be repeated.

Continuous Delivery and Devops are the expansion of agile processes and techniques into the world of IT operations with continuing feedback from the users so the full cycle becomes iterative and continuous.

A solid foundation with agile is a prerequisite to subsequent Devops. And, this will require automated testing, test driven development and continuous integration techniques into operations, automated deployment capabilities, and new development patterns that support continuous delivery.

Our experience within IBM and with many of our clients around the world demonstrates that agile methods can be highly successful in accelerating time-to-value, reducing cycle time, defects and cost while paving the way for the next generation of software delivery.



About the authors

James F. Kile

Jim is a Delivery Project Executive in IBM's Global Business Services supporting IBM's internal application suite. He is an advocate for Agile Software Development, having received a doctorate based upon research into using agile with highly distributed teams in 2007.

James W. Penney

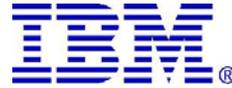
James Penney is a Senior Certified Executive IT Architect in IBM Global Business Services with over 29 years of experience in design and implementation of high performance, complex, open standards based, e-Business applications. He is a subject matter expert in various application development methodologies including Agile and Extreme Programming techniques.

Alex Kramer

Alex is a Partner in IBM's Global Business Services and leads the *Smarter Applications Development & Maintenance* initiative for IBM's Applications Management Services, North America.

Additional contributions by:

James Moffitt, Aslam Hirani, Devi S Nair, Alka Arora, Richa Sharma, Savitha Venukrishnan, Hai Liang Oo, Usha Srikanth and the many IBM professionals who deliver agile results with their clients every day.



© Copyright IBM Corporation 2013
IBM Global Services
Route 100
Somers, NY 10589
U.S.A.
Produced in the United States of America
March 2013
All Rights Reserved

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

Other company, product and service names may be trademarks or service marks of others.

References in this publication to IBM products and services do not imply that IBM intends to make them available in all countries in which IBM operates.

For more information see: <http://www-935.ibm.com/services/us/gbs/application-management/>

References

-
- ¹ M. Albrecht, E. Lesser and L. Ban, “The Software Edge: How Effective Software Development and Delivery Drives Competitive Advantage”, Executive Report, IBM Institute for Business Value, March 1, 2013.
- ² K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*. 2nd ed., Boston, Massachusetts: Addison-Wesley, 2005.
- ³ J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*. Upper Saddle River, New Jersey: Prentice Hall, 2005.
- ⁴ Adapted from: W. W. Royce, “Managing the Development of Large Software Systems: Concepts and Techniques”, *IEEE WESTCON*, pp. 1-9, Los Angeles, CA, IEEE Computer Society Press, 1970.
- ⁵ M. B. Chrissis, M. Konrad and S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*. SEI Series in Software Engineering, Boston: Addison-Wesley, 2003.
- ⁶ F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Anniversary ed., Boston, Massachusetts: Addison-Wesley, 1995.
- ⁷ A. Cockburn, *Agile Software Development*. The Agile Software Development Series, Cockburn, A. and Highsmith, J., eds., 1st ed., Boston: Addison-Wesley, 2002.
- ⁸ K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, *et al.*, “Manifesto for Agile Software Development”, <http://www.agilemanifesto.org>, 2001.
- ⁹ P. Abrahamsson, “Extreme Programming: First Results from a Controlled Case Study”, Proceedings of the 29th Euromicro Conference (EUROMICRO'03), pp. 259-267, Belek-Antalya, Turkey, IEEE Computer Society Press, 2003; P. Abrahamsson and J. Koskela, “Extreme Programming: A Survey of Empirical Data from a Controlled Case Study”, Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), pp. 78-82, Redondo Beach and Marina, California, United States, IEEE Computer Society Press, 2004; J. Drobka, D. Noftz and R. Raghu, “Piloting XP on Four Mission-Critical Projects”, *IEEE Software*, vol. 21, no. 6, pp. 70-75, November/December 2004; I. Gat, “How BMC is Scaling Agile Development”, Proceedings of the Agile 2006 Conference (AGILE'06), pp. 315-320, Minneapolis, Minnesota, United States, IEEE Computer Society Press, 2006.
- ¹⁰ “Small worlds: The social approach to software delivery”, Executive Report, IBM Institute for Business Value, June 1, 2012. <http://www-935.ibm.com/services/us/gbs/thoughtleadership/ibv-social-software-delivery.html>
- ¹¹ *ibid*
- ¹² <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/globalbiz/transform/>
- ¹³ <http://www-935.ibm.com/services/us/gbs/application-management/application-development-management/>