July 26, 2011

Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today

by Dave West

for Application Development & Delivery Professionals

July 26, 2011

Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today

Manage The Water-Scrum And Scrum-Fall Boundaries To Increase Agility

by Dave West
with Mike Gilpin, Tom Grant, Ph.D., and Alissa Anderson

EXECUTIVE SUMMARY

Organizations are adopting Agile through a combination of bottom-up adoption and top-down change. But the reality of Agile adoption has diverged from the original ideas described in the Agile Manifesto, with many adoptions resembling what Forrester labels water-Scrum-fall. This model is not necessarily bad, but if application development professionals do not carefully consider and make the right decisions about where the lines fall between water-Scrum and Scrum-fall, they are unlikely to realize Agile's benefits.

TABLE OF CONTENTS

2 **Agile Is Popular But Not Pure**

Teams Embrace The Ideas Behind Agile But Fall Short Of The Vision

Traditional Planning, Budgeting, Architecture, And Requirements Processes Create Inertia

Release Organizations Resist Releasing Software

9 **Water-Scrum-Fall Is The Reality**

Water-Scrum-Fall: *Water* Defines The Upfront Work

Water-Scrum-Fall: Teams Use *Scrum* To Develop Software In The Middle Of The Process

Water-Scrum-Fall: *Fall* Means Establishing Gates To Limit Software Release Frequency

RECOMMENDATIONS

13 **Understand The Limitations Of Water-Scrum-Fall, And Push The Boundaries**

WHAT IT MEANS

13 **Making Explicit Decisions Is Crucial For Agility**

14 **Supplemental Material**

NOTES & RESOURCES

Over the past two years, Forrester has interviewed more than 300 companies and 100 vendors that are driving the adoption of Agile either through its use or with their tools. Forrester also conducted four Agile-related surveys in 2009 to 2010, all of which influenced this document.

Related Research Documents

"Compliance Is A Hurdle, Not A Barrier, To Agile"
July 26, 2011

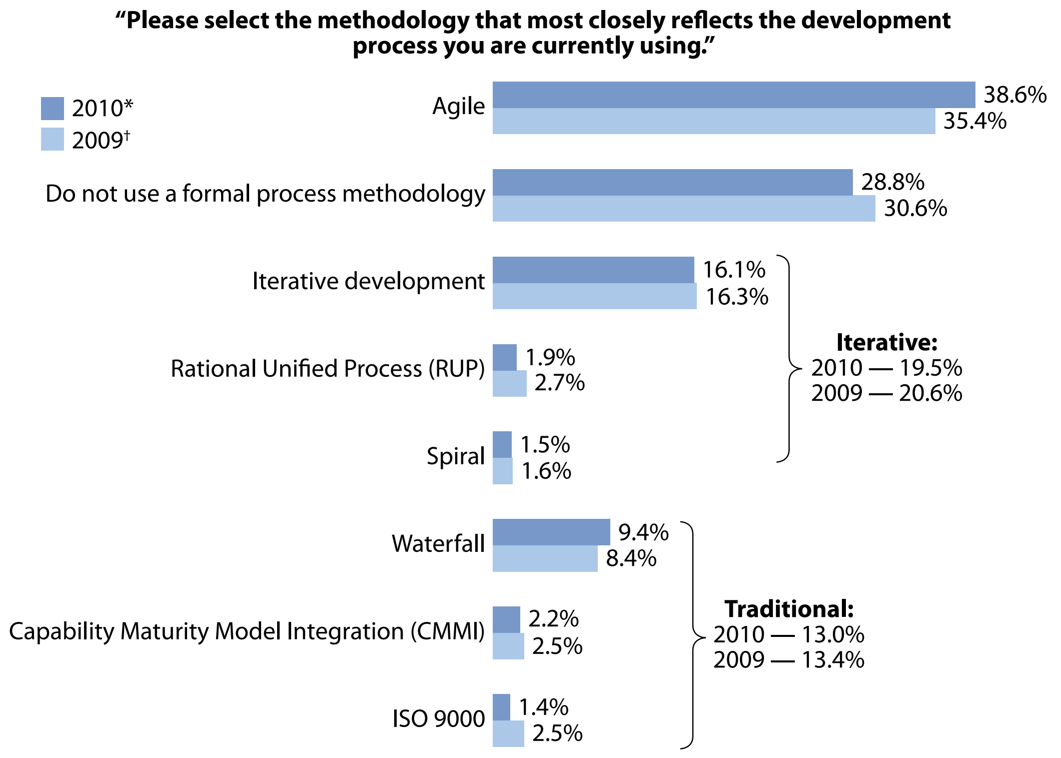
"It's Time To Take Agile To The Next Level"
March 25, 2011

"Best Practices: Building High-Performance Application Development Teams"
November 3, 2010

AGILE IS POPULAR BUT NOT PURE

In February 2001, a group of 17 people met in Utah to explore why some projects are successful and others are not. The Agile Manifesto was the result of that meeting.[1] During the past 10 years, Forrester has observed Agile changing from an approach practiced only by Agilists to one that many organizations employ. We continue to see Agile methods gaining popularity; in a Q3 2010 survey fielded by Forrester and *Dr. Dobb's Journal*, about 39% of 1,023 IT professionals said that they follow an Agile method (see Figure 1). But the reality is that the approach many organizations follow, though inspired by the Agile Manifesto, is constrained by both organizational culture and intuitive governance. The result is Agile adoption that is both challenging for the Agile team and that fails to realize Agile's business benefits, such as faster time-to-market, increased business value, and improved flexibility and responsiveness.

Figure 1 Agile Adoption Continues To Rise



*Base: 1,023 application development professionals
†Base: 1,298 application development professionals
("other" responses not included; percentages may not total 100 because of rounding)

*Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2010
†Source: Forrester/Dr. Dobb's Global Developer Technographics Survey, Q3 2009

60109 Source: Forrester Research, Inc.

Teams Embrace The Ideas Behind Agile But Fall Short Of The Vision

Scrum in particular has become popular; many teams are adopting its basic principles, such as daily meetings, the roles of product owner and Scrum master, and Scrum planning and retrospectives (see Figure 2). Scrum's success can be associated with many things, but in particular a strong focus on teams and team dynamics has attracted many people who feel that traditional approaches lack a real people focus. As Grady Booch often states, "Software is by its very nature a team approach"; Scrum provides the glue to enable that team approach.[2] But even though many organizations have tried to adopt Scrum, the reality is that:

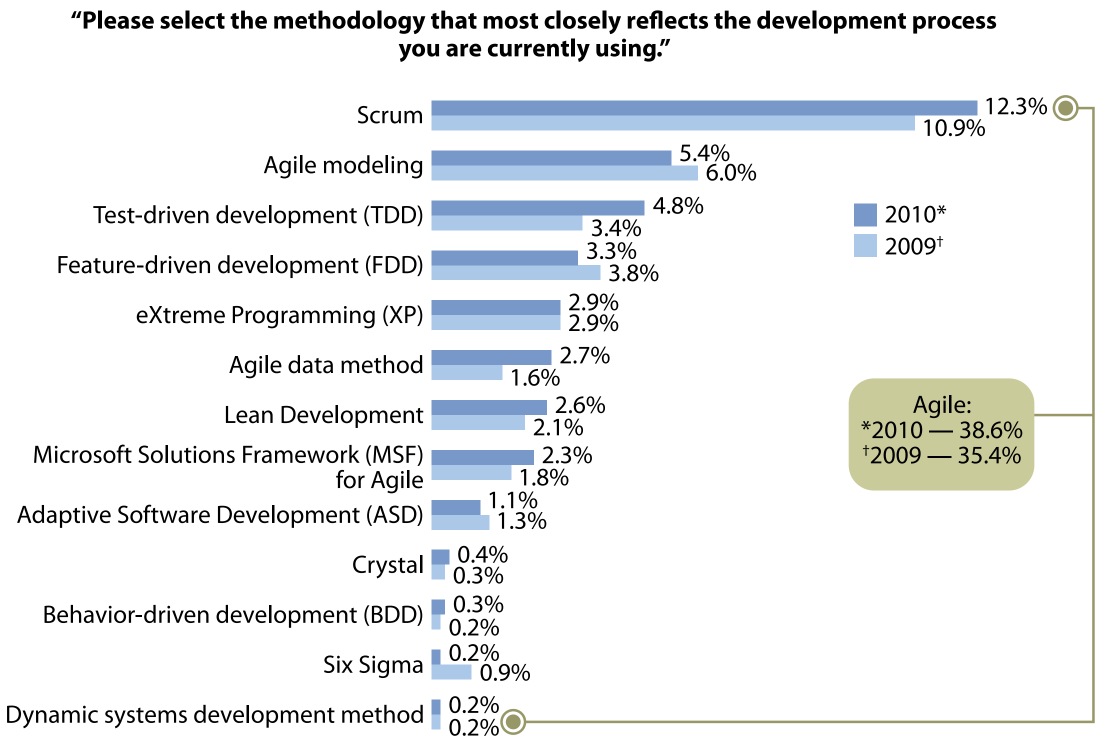
- **Business analysts often become product owners.** At first glance, making business analysts (BAs) product owners makes sense. However, unless they can make real decisions regarding the application or product direction, giving them this role becomes a source of many problems, slowing teams down as they wait for the real business owner to make a decision. The top responsibility of the product owner is to own the product, which can be very difficult for business analysts, who traditionally have focused on communicating a customer's intent to the development team. They often lack the credibility and authority to drive business and technical decisions.

- **Each team's members work on more than one project.** The ultimate end game of resource management is to maximize utilization of resources such as testers and developers. This often means that individuals work on multiple projects, slicing their time between different teams, problems, and even organizations. Agile development is a collaborative approach to delivering software that requires team members to work together on problems. Composing teams from people working on many projects makes it hard for team members to find time to collaborate. Time slicing also requires more context switching, which is both an overhead and a risk. Ideas and information are lost as team members switch between projects.

- **Not everything is done in the sprint.** As teams wrestle with the reality of doing Agile in a non-Agile organization, many activities that should be included in the sprint are moved outside of the development team because of constraints, tradition, or process. Testing is a classic example of this situation; in many organizations, separate testing teams do functional, performance, and security validation. This often results in missed tasks, slower bug resolution, and loss of the rapid feedback Agile teams normally enjoy.

- **A project culture causes teams to lose momentum.** Many delivery organizations have a project culture, which means that teams formed for one project do the majority of the work. Agile methods encourage teams to stay together across projects to reduce the overhead of learning to work together. In a project culture, learning suffers and teams lose momentum and flow when they have to socialize working practices and gain agreement on engagement models every time a new project team is formed.[3]

Figure 2 Scrum Has Become Very Popular



*Base: 1,023 application development professionals
†Base: 1,298 application development professionals
(percentages do not total 100 because only responses for Agile methodologies are shown)

*Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2010
†Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009

60109 Source: Forrester Research, Inc.

Traditional Planning, Budgeting, Architecture, And Requirements Processes Create Inertia

Delivery organizations have built up decades of traditional management tools — budgets, plans, architecture road maps, and requirements specifications — that make it challenging to move Agile beyond a small experiment to the way most work gets done. Describing the issue, the vice president of development at a large retailer said: "We cannot fund projects unless we know what we want, and we do not know what we want unless we have a project. It is a chicken and egg problem!" Many organizations adopting Agile have difficulty knowing when to start. Many Agile methods encourage teams to do work outside of the iteration, but it's difficult to decide how much and which work to do.[4] In these cases, organizations fall back on tradition because:

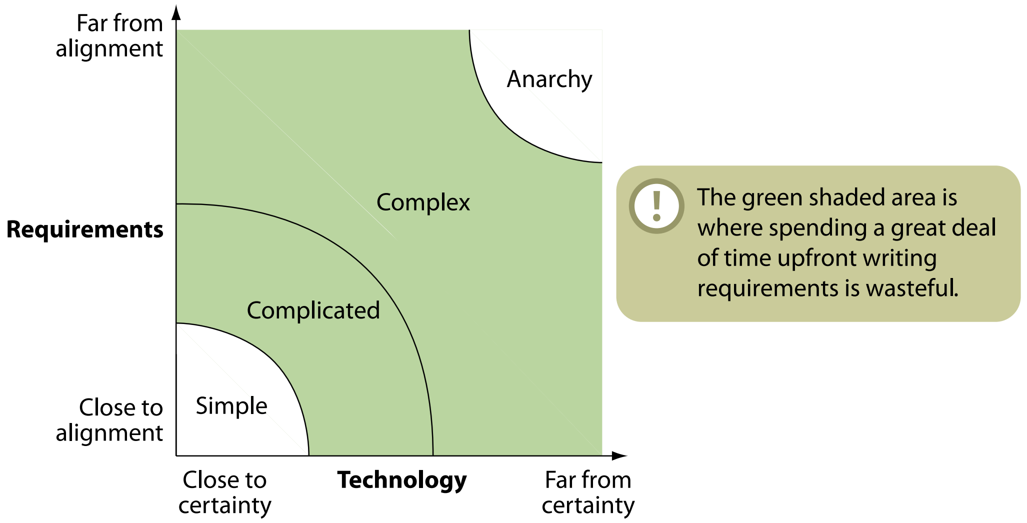
- **Plans drive funding.** The plan defines the project. It includes a detailed description of the tasks, resources, and time the project requires, translated into cost and time estimates. The project's justification is the difference between the benefits described in the business case and the initial estimates and plan. Though Agile projects do encourage having an upfront plan, that plan often includes far fewer details than traditional approaches require. Estimates are at the backlog rather than the task level.[5]

- **Different people do architecture and design.** Specialist departments such as enterprise architecture (EA) and data governance make decisions. Project work moves through these groups. There are clear responsibilities within each group during that phase of the life cycle, but overall ownership of the project tends to fall to the project manager, the one person who plays a consistent role through each phase. Agile encourages a cross-functional team to work together on the problem, seeking help from support organizations as necessary. This enables faster turnaround and development of more synergy between the disciplines.

- **The SDLC and governance enforce the creation of a set of documents.** Within the traditional software development life cycle (SDLC), projects use documents to mitigate risk. Documents describe the problem and solution, allowing stakeholders from outside the team to sign off on them. Agile approaches the problem of risk in a different way, focusing the team on delivering software in an order that exposes risk and allows the appropriate parties to review the risk in the context of working software. For example, if the team is concerned about performance, it develops code for performance testing as early in the process as is feasible.

- **Development is not involved in the requirements process.** Adding to the planning requirement traditional approaches often impose, BA groups are often working outside of development to engage with the business and create documents that describe the business problem. These documents form the contract between the business and development. For easily described problems (e.g., familiar problems with familiar solutions) or problems that will not benefit from input from the development team, this approach works well. However, many situations are not as cut-and-dried and have many unknowns. And where technology has a large impact on requirements (think mobile), it becomes even harder to define the requirements prior to understanding more of the solution (see Figure 3).

Figure 3 Complex Requirements And Technology Influence Process



Source: Adapted from Ralph D. Stacey, *Strategic Management And Organisational Dynamics: The Challenge Of Complexity,* Prentice Hall, 2000.

60109 Source: Forrester Research, Inc.

Release Organizations Resist Releasing Software

In the majority of large IT organizations, software releases — except for bug fixes — are rare; most have only three or four releases a year. The cost and risk of releasing software is often cited as the primary reason for this policy. When releasing software is costly, teams tend to increase the amount of software in each release, increasing the release's risk and complexity, which in turn increases the effort required to manage risk. This vicious circle makes release processes increasingly expensive and time-consuming, reducing overall satisfaction (see Figure 4). Constraints to releasing software frequently include:

- **Legacy systems are too complex.** I still wake in a cold sweat in the middle of the night thinking about a piece of code I worked on called RP150. RP150 had grown so complex that it was almost impossible to know if your change was going to break something. This resulted in very long test processes and change by experimentation, making planning and estimating difficult. It also meant that four times a year I slept with a beeper!

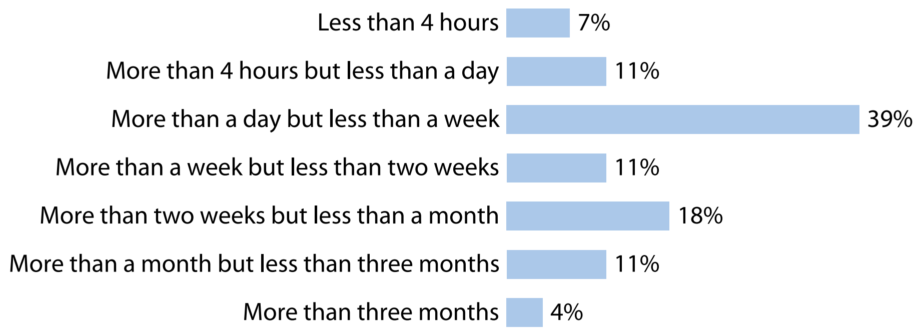
- **Operations and release groups are a bottleneck.** Because production systems must be governed, audited, and visible, the fewer people who work on those systems the better. This leads to these resources becoming bottlenecks — and to these teams finding it difficult to work with Agile teams developing releasable software. Forrester Analyst Jeffrey Hammond describes this as the release bus problem, with the bus being the bottleneck.[6]

- **Infrequent releases are the cultural norm.** The separation of operations and development has bred a culture of handoff and specialization, which in turn has shaped the processes that are an integral part of the organization's culture. Challenging this culture is often very difficult and requires careful attention to human factors such as power, position, and autonomy.

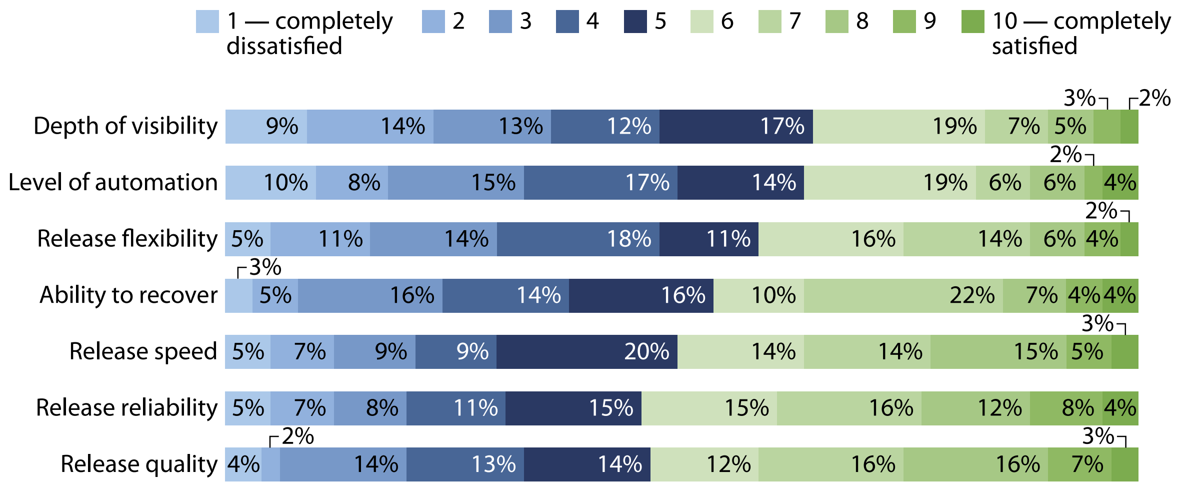
- **The business does not like change.** Highlighting the business as the principle roadblock for releasing software, the project management office (PMO) leader of a large entertainment company described the company's release process, saying, "We would release software more frequently, but our business does not like change." Business users may feel this way in part because changes are often so painful that they don't want to go through that again; after all, they have a business to run. Modern architecture and technology makes it possible to change software without much impact to the users — witness the software-as-a-service (SaaS) industry. But teams incented to build new features and capabilities often undervalue building software that is easy to adopt.

Figure 4 There Are Many Challenges Associated With Releasing Software

4-1 **"If you were to change one line of code on your project, how long would it typically take your organization to push the resulting change into production?"**



4-2 **"How satisfied are you with the following aspects of your release management process?"**



Base: 101 IT professionals involved in or aware of their company's release management processes (percentages may not total 100 because of rounding)

Source: Q4 2010 Global Release Management Online Survey

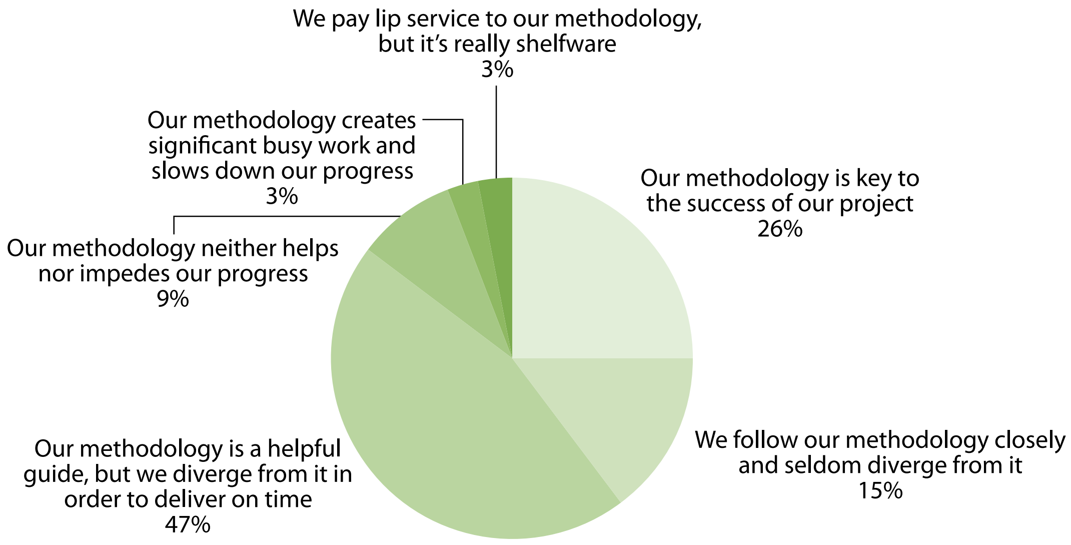
60109 Source: Forrester Research, Inc.

WATER-SCRUM-FALL IS THE REALITY

Hybrid Agile methods are a reality in most Agile implementations (see Figure 5). This happens in part because Agile adoption has been practitioner-led, leading teams to focus on domains they can influence, mainly the team itself. Areas outside of their control, such as business analysis and release management, continue to follow more-traditional approaches, meaning that Scrum adoption is limited to the development-team level (see Figure 6). Compliance requirements are another factor driving hybrid approaches, as they call for strong governance processes before and after development.[7]

Figure 5 Hybrid Processes Are The Reality Of Agile

"Which statement best describes how you feel about the development methodology you are using?"

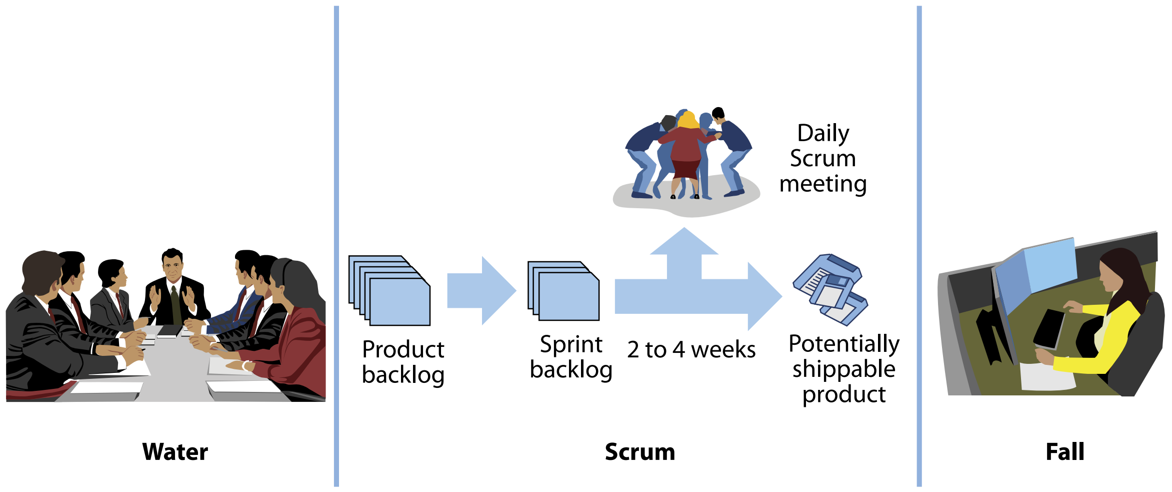


Base: 575 enterprise (1,000 or more employees) application development professionals using Agile (percentages do not total 100 because of rounding)

Source: Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009

60109 Source: Forrester Research, Inc.

Figure 6 Water-Scrum-Fall Is The Reality



60109 Source: Forrester Research, Inc.

Water-Scrum-Fall: *Water* Defines The Upfront Work

Governance rules require many organizations to define requirements and plans before starting work. In some companies, these plans form the basis of a contract between the business and IT that defines project direction, timeline, and budget. One vice president of development framed this, saying: "We have to spend time upfront with the business building the requirements and plans to ensure that we know what they want and they know how long it is going to take and how much it is going to cost. The problems start when the business does not know what they want or the architecture is new to us." Be careful spending too much time upfront because:

- **Too many early requirements are too many wrong requirements.** Never implementing a requirement or getting a requirement wrong is waste. This waste not only stops the team from spending time on more-important requirements but also increases project complexity.

- **Users don't always communicate effectively what they want.** How many times have end users only understood the problem after they have seen a solution that was not quite right? The problem with spending a great deal of time writing down detailed requirements is that users get to see the whole solution after some delay and only then can provide feedback. Feedback late in the cycle tends to be less welcome, as the team has to balance the feedback with the significant cost of changing the application.

- **The team takes less ownership of the outcome.** The handoff from requirements to development may encourage teams to focus less on customer value and more on fulfilling their contract. The resulting process looks more like a game in which the business tries to get all its requirements in and IT tries to avoid including any requirements that might increase risk to the project. This cat-and-mouse process does not add value because with this approach, teams take ownership of tasks they control, such as requirements, design, and development, but not for the completed product or associated business value. Thus, documents may be correct and may have received sign-off but actually add very little to overall business value.

- **The development team is less cross-functional.** Having a separate phase focused on planning and requirements often splits the team between functions. One of Agile processes' ultimate strengths is the idea of a cross-functional team in which development, quality assurance (QA), and analysis work together in a dynamic, often fluid, manner.

Water-Scrum-Fall: Teams Use *Scrum* To Develop Software In The Middle Of The Process

Scrum is the most popular Agile method and continues to gain popularity. It provides a simple set of principles, working practices, and roles for teams to execute. Developers particularly like Scrum's guidance on team organization and transparency. Developers like delivering software, so the practice of frequently releasing software makes intuitive sense.[8] However, teams should guard against embracing Scrum principles but missing some of its most-important characteristics. When adopting Scrum principals, be sure to:

- **Build a cross-functional team.** Applying Scrum to only the developers is a recipe for disaster. A proper Scrum team must comprise all the people necessary to deliver working software; typically, this means developers, testers, and business analysts working toward a common goal.

- **Include testing within the sprint.** Scrum-fall may encourage the separation of testing from development so that testing becomes part of the release process. Rapid feedback and course correction require teams to test within each sprint. Leaving testing to subsequent phases increases the cost and impact of defects as the number of dependencies grows.

- **Engage with the business.** Continue to engage with the business after the requirements phase. This allows sharing of additional context and ideas and encourages shorter and higher-level initial requirements, as business owners no longer feel that requirements specifications have to be their last word on the subject. Scrum teams should therefore include on the team business leaders who can make decisions regarding the project's direction.

- **Accept that change can happen.** The reality of water-Scrum-fall is that change will continue. The water stage defines the overall direction of the project, but the team will have many insights during the project that challenge initial ideas. By supporting change while at the same time ensuring that the team understands the impact of that change, the team will not only build better applications but will also learn more about its process for future implementations. One way to drive the point home is to ask the project's management a simple question: "Do you want the technology to help us adapt quickly to threats and opportunities?" If the answer is yes, then tune the approach to favor discovery and action over planning and execution.

Water-Scrum-Fall: *Fall* Means Establishing Gates To Limit Software Release Frequency

Frequent software releases to the customer enable rapid feedback and ensure that valuable software is being used as early as possible. However, most organizations do not have the architecture required to support dynamic, flexible releases; instead, they do infrequent releases backed by heavy process and governance. Adopting Agile processes will not by itself change the firm's underlying enterprise architecture; therefore, teams have to make the best of the situation. Agile teams should push the boundaries of the release process by:

- **Bringing operations and development into a closer working partnership.** The separation between operations and development has grown wider over the years. Process models such as the IT Infrastructure Library (ITIL) and Capability Maturity Model Integration (CMMI) encourage handoffs and demarcation of responsibility and ownership rather than collaboration, teaming, and shared goals. This may lead each party to think that the other is trying to pull a fast one, leading to confrontational situations and heavy reliance on paperwork and formal documentation. By sharing process and building a cross-functional team inclusive of operations, IT organizations can remove the disconnects.

- **Examining and removing release bottlenecks.** Once you have built a cross-functional team that includes operations, task that team with clear objectives: improving the process of releasing software and challenging the status quo. At the end of each sprint, ask: Can we go into production with this? If the answer is no, then evaluate what would need to be done to make that possible in the next sprint.

- **Adding more release activities to sprints.** Incrementally add to sprints release process activities such as preproduction testing, data migration, and security and performance testing. Including these tests within each sprint not only gathers more rapid feedback for the team but also encourages the team to automate these processes, increasing the overall fidelity of their results while also automating governance. Consistent automation of the complete promotion model also allows release managers to better understand software status.

- **Building shared objectives driven by the business.** Traditional approaches measure release teams based not on the amount they change but on minimizing the impact of that change. They measure development teams on the amount they deliver and how well they stay on schedule. The tension between these two conflicting measures incents different behaviors, increasing friction between the two groups. Giving everyone the same business-oriented goals helps remove this friction, allowing the entire team to balance quality and functionality in the context of a change.

RECOMMENDATIONS

UNDERSTAND THE LIMITATIONS OF WATER-SCRUM-FALL, AND PUSH THE BOUNDARIES

Unless you are a software-as-a-service (SaaS) vendor, it is highly likely that the Agile process you are following resembles water-Scrum-fall. Most firms do requirements and planning before forming a Scrum team. Once management is happy with the plans, development follows a backlog-driven Scrum model. The development team delivers software frequently, but the production release process runs at a different cadence, picking up the software at the intervals the release plan defines. This model is not inherently bad, and it does at least afford some level of agility at the development-team level, but if application development professionals want to maximize the value of agility, they should:

- **Push back on the water-Scrum side of the model.** Spending too much time upfront will not increase the quality of the release; on the contrary, it is wasteful. Documents are a poor proxy for working software, and thus any documents created should be just enough to introduce the problem area and allow high-level planning and development work to commence.[9]
- **Ensure they have built a truly Agile team for the middle of the process.** Just calling a team Agile is not enough; Agile teams should include all the people necessary to deliver working software, coupled with clear measures that allow the team to focus on delivering the right software that maximizes business value.
- **Increase the frequency of the releases to production.** Application development professionals should challenge the status quo of infrequent releases and push to better integrate release processes into the development team.

WHAT IT MEANS

MAKING EXPLICIT DECISIONS IS CRUCIAL FOR AGILITY

Perhaps one of the biggest problems with traditional software delivery approaches is that processes that evolved in response to certain situations have become so common that organizations rarely challenge them. Waterfall evolved in response to the business' growing demand for IT to deliver more and more process automation. Many waterfall processes assume that the problem domain and the solution space are consistent, with the development team having solved similar problems before. Dealing with the unknown is not a characteristic of a waterfall process that relies on interim documents to describe the problem and its solution.

Agile evolved in response to uncertainty, helping teams deliver value while working in a world of changing understanding and priorities. Most IT organizations don't follow a strictly traditional or strictly Agile approach but fall somewhere in the middle. Some problem domains are very new, and they and their associated technologies are unknown to the organization. For example, consider the application of mobile technology to a new business process or channel. Other domains have some areas of unknown but include vast chunks of functionality that teams understand and have done before.

In the future, Agile processes will become less about a particular method such as Scrum or eXtreme Programming (XP) and more about applying the right mix of practices and techniques to the situation and problem. Application development professionals will apply the most appropriate hybrid process to each problem. This hybrid could be a combination of waterfall, Agile, and other techniques such as Kanban. The result will be a more robust, flexible process that evolves in response to the situation rather than a well-documented, inflexible process that assumes that all problems are the same. These processes will be about more than just development; they will also address operations concerns and, more importantly, business change. And of course they will be agile!

SUPPLEMENTAL MATERIAL

Methodology

Forrester fielded its Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2010 to 1,036 developer professionals. Forrester fielded the survey September 2010 to October 2010. Respondent incentives included a summary of the survey results.

The Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2009, was fielded to 1,298 application development and program management professionals who are readers of *Dr. Dobb's* magazine. For quality assurance, respondents are required to provide contact information and answer basic questions about themselves. Forrester fielded the survey from July 2009 to August 2009. Respondent incentives included a summary of the survey results and a chance to win one of five $50 gift certificates.

Forrester fielded its Q4 2010 Global Release Management Online Survey to 101 development professionals; however, only a portion of survey results are illustrated in this document. The respondents consist of volunteers who join on the basis of interest and familiarity with specific topics. For quality assurance, panelists are required to provide contact information and answer basic questions about their firms' revenue and budgets. Forrester fielded the survey from October to November 2010.

Exact sample sizes for the surveys used in this report are provided on a question-by-question basis. Panels are not guaranteed to be representative of the population. Unless otherwise noted, statistical data is intended to be used for descriptive and not inferential purposes.

ENDNOTES

1 Source: Manifesto for Agile Software Development (http://agilemanifesto.org/).

2 Grady Booch is one of the creators of UML and the inspiration of many modern development methods. His thoughts on software development can be found at developerWorks (http://www.ibm.com/developerworks/library/i-booch/).

3 Flow is the time a developer or team is productively engaged in doing work as opposed to waiting on some other resource to deliver an artifact on which the team depends or performing some overhead activity (waste) that does not bring progress on a work task. Agile teams try to maximize flow time and minimize waste. Atlassian is an interesting example; it maintains flow by implementing bottom-up development processes that maximize flow and improve team autonomy. See the January 14, 2011, "Case Study: Atlassian Creates An Innovation Culture That Produces Results" report.

4 An example of work being done outside of the iteration or sprint is the use of Sprint0 in Scrum. Sprint0 is described as setting up the project and the foundations necessary to start a sprint and undertaking activities such as defining the backlog, planning the overall project, and forming the team.

5 Yet as one app delivery leader put it, "We've really been creating fiction all these years when we produced those detailed plans, because nobody really knows what we'll have to do or how much it will cost until after we get some way along in the project!"

6 For a great description of what Forrester thinks about the release practice, see the February 7, 2011, "Five Ways To Streamline Release Management" report.

7 Compliance does have an effect on application development and delivery — but not in the way that most people assume. In some aspects of software development, such as choice of methodology, compliance has little or no impact. This is true for Agile, which on the surface appears to be completely incompatible with heavyweight process and documentation requirements. In other areas, such as mobile development, compliance does have the expected effect, significantly limiting a team's options. Compliance also has some unanticipated effects on developers themselves, shaping their attitudes toward their work. Finally, and even more surprisingly, compliance can be a boon to the team in some ways, providing a compelling reason to improve team productivity. See the July 26, 2011, "App Dev Teams Dispel The Compliance Boogeyman" report.

8 Continuous Integration describes a development practice of bringing the code of a project together frequently to ensure that it builds, integrates, and tests. Perhaps the best description of this practice was by Martin Fowler. Source: Martin Fowler, "Continuous Integration," ThoughtWorks, May 1, 2006 (http://martinfowler.com/articles/continuousIntegration.html).

9 In traditional app dev shops, business analysts (BAs) create standardized, lengthy, text-heavy Microsoft Word documents to represent software requirements, but creating and consuming this documentation takes time that most BAs and stakeholders just don't have. On the other hand, Agile methods advocate that teams create "just enough" requirements documentation to meet the need. They use less-formal documentation, more pictures, and less text. Traditional shops have heard the "just enough" message, but they don't know how to apply it in their environments. See the December 29, 2010, "Thinking Lean: How Much Requirements Documentation Is 'Just Enough'?" report.