
# The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments

*Scott W. Ambler*
*Chief Methodologist for Agile, IBM Rational*

**Contents**



## Executive summary

The Agile Scaling Model (ASM) defines a roadmap for effective adoption and tailoring of agile strategies to meet the unique challenges faced by a system delivery team. The first step to scaling agile strategies is to adopt a disciplined agile delivery lifecycle which scales mainstream agile construction techniques to address the full delivery process, from project initiation to deployment into production. The second step is to recognize which scaling factors, if any, are applicable to a project team and then tailor your adopted strategies accordingly to address the range of complexities that the team faces.

The scaling factors are:

1. Team size
2. Geographical distribution
3. Regulatory compliance
4. Domain complexity
5. Organizational distribution
6. Technical complexity
7. Organizational complexity
8. Enterprise discipline

This paper begins with an overview of the fundamentals of agile software engineering and of common agile methodologies. It then argues for the need to scale agile development strategies to address the full delivery lifecycle, showing how the Scrum method can be extended to do exactly that. In fact, our experience is that the first "scaling factor" that organizations face with agile development is lifecycle scope. It then explores the eight agile scaling factors and their implications for successfully scaling agile software delivery to meet the real-world needs of your organization.

## Introduction

***Agile software development is an evolutionary, highly collaborative, disciplined, quality-focused approach to software development.***

Agile software development is an evolutionary, highly collaborative, disciplined, quality-focused approach to software development, whereby potentially shippable working software is produced at regular intervals for review and course correction. Agile software development processes[1] include Scrum, Extreme Programming (XP), Open Unified Process (OpenUP), and Agile Modeling (AM), to name a few. At IBM we've used agile techniques internally for many years, and both the IBM Global Services and IBM Rational organizations have been working with many

### Highlights

***Agile development is becoming widespread because it works well - organizations are finding that agile project teams, when compared to traditional project teams, enjoy higher success rates, deliver higher quality, have greater levels of stakeholder satisfaction, provide better return on investment (ROI), and deliver systems to market sooner.***

***Agile approaches are being used in a wide range of situations, not just the small, co-located team environments that dominate the early agile literature. Agile strategies are being applied throughout the entire software delivery lifecycle.***

of our customers to help them apply agile techniques within their own environments, often under complex conditions at scale. Agile techniques held such promise that beginning in mid-2006 an explicit program was put in place to adopt agile processes on a wide-scale basis throughout IBM Software Group, an organization with over 25,000 developers.

Agile software development techniques have taken the industry by storm, with 76% of organizations reporting in 2009 that they had one or more agile projects underway [1]. Agile development is becoming widespread because it works well – organizations are finding that agile project teams, when compared to traditional project teams, enjoy higher success rates, deliver higher quality, have greater levels of stakeholder satisfaction, provide better return on investment (ROI), and deliver systems to market sooner [2]. But, just because the average agile team is more successful than the average traditional team, that doesn't mean that all agile teams are successful nor does it mean that all organizations are achieving the potential benefits of agile to the same extent.

As you may know, agile approaches support software construction by small, co-located teams. What you may not have heard is that agile approaches are being used for the development of a wide range of systems, including but not limited to web-based applications, mobile applications, fat-client applications, business intelligence (BI) systems, embedded software, life-critical systems, and even mainframe applications. Furthermore, agile approaches are being applied by a range of organizations, including financial companies, manufacturers, retailers, online/e-commerce companies, healthcare organizations, and government agencies. Some organizations, including IBM, are applying agile techniques on large project teams – hundreds of people – and on distributed teams, in regulatory environments, in legacy environments, and in high-complexity environments.

The point is that agile approaches are being used in a wide range of situations, not just the small, co-located team environments that dominate the early agile literature[2]. Agile strategies are being applied throughout the entire software delivery lifecycle, not just construction, and very often in very complex environments that require far more than a small, co-located team armed with a stack of index cards. Every project team finds itself in a unique situation, with its own goals, its abilities, and challenges to overcome. What they have in common is the need to adopt and then tailor agile methods, practices, and tools to address those unique situations. But how? We know this can be very hard to do well.

**Highlights**

The goal of this paper is to share our experiences learned in applying agile strategies and techniques in organizations around the world, often at a scale far larger than the techniques were pioneered for. I begin with an overview of agile software development concepts and several agile methodologies which reflect those concepts. I then describe the Agile Scaling Model (ASM), a contextual framework for scaling the plethora of agile methodologies and practices out there today.

## Agile software development

If you already understand the values and principles of the Agile Manifesto, you can potentially skip that section below. What will likely be new to you is a definition for agile software development is also proposed, the implications of which this paper explores in detail.

### The Agile Manifesto

***Seventeen methodologists crafted a manifesto, and a collection of supporting principles, for encouraging better ways of developing software. The manifesto defines four values and twelve principles which form the foundation of the agile movement.***

To address the challenges faced by software developers an initial group of seventeen methodologists formed the Agile Software Development Alliance (www.agilealliance.com), often referred to simply as the Agile Alliance, in February of 2001. An interesting thing about this group is that they all came from different backgrounds, yet were able to come to an agreement on issues that methodologists typically don't agree upon. They crafted a manifesto, and a collection of supporting principles, for encouraging better ways of developing software. The manifesto defines four values and twelve principles which form the foundation of the agile movement.

The Agile Manifesto[3] states:
*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

1. *Individuals and interactions over processes and tools*
2. *Working software over comprehensive documentation*
3. *Customer collaboration over contract negotiation*
4. *Responding to change over following a plan*

*That is, while there is value in the items on the right, we value the items on the left more.*

**Highlights**

***The values of the Agile Manifesto are supported by a collection of 12 principles .***

The values of the Agile Manifesto are supported by a collection of 12 principles [4] which explore in greater detail the philosophical foundation of agile software methods. These principles are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity– the art of maximizing the amount of work not done– is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

**Highlights**

***Agile software development is an evolutionary (iterative and incremental) approach which regularly produces high quality software in a cost effective and timely manner via a value driven lifecycle.***

## Toward a definition

The values and principles of the Agile Manifesto provide a solid philosophical foundation for effective software development, but a precise definition would be helpful for describing a specific approach. In fact, there is no official definition for agile software development and there likely never will be. Here is a potentially useful working definition: [4]

> *Agile software development is an evolutionary (iterative and incremental) approach which regularly produces high quality software in a cost effective and timely manner via a value driven lifecycle. It is performed in a highly collaborative, disciplined, and self-organizing manner with active stakeholder participation to ensure that the team understands and addresses the changing needs of its stakeholders. Agile software development teams provide repeatable results by adopting just the right amount of ceremony for the situation they face.*

Let's explore some key concepts in this definition:

1. *Evolutionary* - Agile strategies are both iterative and incremental in nature. "Iterative" means that you are working on versions of functioning code through a series of activities that are repeated for each version, or build, until the project is complete. But this doesn't mean that the work itself is repetitive. On any given day you may do some requirements analysis, some testing, some programming, some design, some more testing, and so on. "Incremental" means that you add new functionality and working code to the most recent build, until such time as the stakeholder determines there is enough value to release the product..
2. *Regularly produces high quality software* - Agilists are said to be quality-focused. They prefer to test often and early, and the more disciplined practitioners even take a test-first approach, which means writing a single test and the just enough production code to fulfill that test (then they iterate). Many agile developers have adopted the practice of refactoring, which is a technique where you make simple changes to your code or schema which improves its quality without changing its semantics. Adoption of these sorts of quality techniques appears to succeed. Agile teams are more likely to deliver high quality systems than traditional teams [2]. Within IBM, we focus on 'consumability' within our software engineering teams. Consumability encompasses quality and other features such as ease of deployment and system performance [22].
3. *Cost-effective and timely manner* - Agile teams prefer to implement functionality in priority order, with the priority being defined by their stakeholders (or

**Highlights**

***Agile teams prefer to produce potentially shippable software each iteration, enabling stakeholders to determine when they wish to have a release delivered and thereby improving timeliness. Short iterations reduce the feedback cycle, improving the chance that agile teams will discover problems early.***

a representative thereof). Working in priority order enables agile teams to maximize the return on investment (ROI) because they are working on the high-value functionality as defined by their stakeholders, thereby increasing cost effectiveness. Agile teams also prefer to produce potentially shippable software each iteration (an iteration is a time-box, typically 2-4 weeks in length), enabling their stakeholders to determine when they wish to have a release delivered to them and thereby improving timeliness. Short iterations reduce the feedback cycle, improving the chance that agile teams will discover problems early (they "fail fast") and thereby enable them to address the problems when they're still reasonably inexpensive to do so.[5]

4. *Value-driven lifecycle* - One result of building a potentially shippable solution every iteration is that agile teams produce concrete value in a consistent and visible manner throughout the lifecycle.
5. *Highly collaborative* - People build systems, and the primary determinant of success on a development project is the individuals and the way that they work together. Agile teams strive to work as closely together and as effectively as possible. This characteristic must mark every engineer on the team, including those in the leadership roles [23].
6. *Disciplined* - Agile software development requires greater discipline on the part of practitioners than what is typically required by traditional approaches [31].
7 *Self organizing* - This means that the people who do the work also plan and estimate the work.
8. *Active stakeholder participation* - Agile teams work closely with their stakeholders, who include end users, managers of end users, the people paying for the project, enterprise architects, support staff, operations staff, and many more. Within IBM we distinguish between four categories of stakeholder: principles/sponsors, partners (business partners and others), end users, and insiders. These stakeholders, or their representatives (product owners in Scrum[6], or on-site customers in Extreme Programming[7], or a resident stakeholder in scaling situations), are expected to provide information and make decisions in a timely manner.
9. *Changing needs of stakeholders* - As a project progresses, stakeholders typically gain a better understanding of what they want, particularly if they're shown working (i.e., functional, though incomplete) software on a regular basis; consequently, they change their requirements as these reviews occur. Changes in the business environment, or changes in organization priority, will also motivate changes to the requirements.

**Highlights**

10. *Repeatable results* - Stakeholders are rarely interested in how you deliver a solution; they're only interested in what you deliver. In particular, they are often interested in having a solution that meets their actual needs, in spending their money wisely, in a high-quality solution, and in something that's delivered in a timely manner. In other words, they're interested in repeatable results, not repeatable processes.
11. *Right amount of ceremony for the situation* - "Ceremony" refers to the degree of process adherence (methodology) over the course of a project. High ceremony might involve, for example, copious documentation or formal reviews of diagrams and other schema. Agile approaches minimize ceremony in favor of delivering concrete value in the form of working software, but that doesn't mean they do away with ceremony completely. Agile teams will still hold reviews, when it makes sense to do so. Agile teams will still produce deliverable documentation, such as operations manuals and user manuals, and as do traditional teams [5].

### Criteria to determine if a team is agile

***Undisciplined "ad-hoc" teams often claim to be agile, because they've read an article or two about agile development, and interpret agility to mean any cool, liberated form of undocumented software creativity. These ad-hoc teams often run into trouble.***

A common problem in many organizations is that undisciplined "ad-hoc" teams often claim to be agile, because they've read an article or two about agile development, and interpret agility to mean any cool, liberated form of undocumented software creativity. These ad-hoc teams often run into trouble, and give actual agile teams a bad name. I suggest the following five criteria to determine if a team is truly agile:

1. *Working software* - Agile teams produce working software on a regular basis, typically in the context of short, stable, time-boxed iterations.
2. *Active stakeholder participation* - Agile teams work closely with their stakeholders, ideally on a daily basis.
3. *Regression testing* - Agile teams do, at a minimum, continuous developer regression testing.[8] Disciplined agile teams take a Test-Driven Development (TDD) approach.
4. *Organization* - Agile teams are self-organizing, and disciplined agile teams work within an appropriate governance framework at a sustainable pace. Agile teams are also cross-functional "whole teams," with enough people with the appropriate skills to address the goals of the team.
5. *Improvement* - Agile teams regularly reflect on, and disciplined teams also measure, how they work together and then act to improve on their findings in a timely manner.

**Highlights**

There are four important points to make about these criteria:

1. *You may still be working on fulfilling some criteria* - Your organization may be fairly new to agile and is still working to adopt some agile strategies. This is perfectly fine, as long as they explicitly recognize the gaps and plan to improve. However, if the team doesn't recognize the need to fulfill these five criteria, or believe that they're "special" for some reason and don't need to do so, then they're not agile no matter how adamant they are.
2. *The criteria are situational* - Several of the terms in the above criteria are underlined to indicate where your strategy needs to be flexible. For example, some agile teams will produce working software every two weeks whereas others may be in a more complex situation and may only do so every two months (although IBM culture routinely challenges even our most complex teams to integrate and stabilize frequently). Different situations require different strategies, meaning that one process size does not fit all.
3. *The criteria are easy to assess* - My experience is that I've always been able to identify ad-hoc teams who claim to be agile with the five listed criteria, but who very obviously fail in several of them. Teams that are truly agile are standouts.
4. *A non-agile team could pass* - It's conceivable that a non-agile team could meet all five criteria, although I have yet to run into one. If so, perhaps they could benefit from some agile ideas but it's likely that your organization has other teams in greater need of help than this one anyway – declare success and move on!

### The Agile Scaling Model (ASM)

***The Agile Scaling Model is a contextual framework for effective adoption and tailoring of agile practices for a system delivery team of any size.***

The Agile Scaling Model (ASM) is a contextual framework for effective adoption and tailoring of agile practices to meet the unique challenges faced by a system delivery team of any size. Figure 1 overviews the ASM, depicting how the ASM distinguishes between three scaling categories:

1. *Core agile development* - Core agile methods, such as Scrum and Agile Modeling, are self governing, have a value-driven system development lifecycle (SDLC), and address a portion of the development lifecycle. These methods, and their practices – such as daily stand up meetings and requirements envisioning – are optimized for small, co-located teams developing fairly straightforward systems.

2. *Disciplined agile delivery* - Disciplined agile delivery processes, which include Dynamic System Development Method (DSDM) and Open Unified Process (OpenUP), go further by covering the full software development lifecycle from project inception to transitioning the system into your production environment (or into the marketplace as the case may be). Disciplined agile delivery processes[9] are self organizing within an appropriate governance framework and take both a risk and value driven approach to the lifecycle. Like the core agile development category, this category is also focused on small, co-located teams delivering fairly straightforward systems. To address the full delivery lifecycle you need to combine practices from several core methods, or adopt a method which has already done so, and adopt a few (egads!) "traditional" practices such as doing a bit of up-front requirements and architecture modeling which have been tailored to reflect agile philosophies to round out your overall software process.
3. *Agility at Scale* - This category focuses on disciplined agile delivery where one or more scaling factors are applicable. The eight scaling factors are team size, geographical distribution, regulatory compliance, organizational complexity, technical complexity, organizational distribution, and enterprise discipline. All of these scaling factors are ranges, and not all of them will likely be applicable to any given project, so you need to be flexible when scaling agile approaches to meet the needs of your unique situation. To address these scaling factors you will need to tailor your disciplined agile delivery practices and in some situations adopt a handful of new practices to address the additional risks that you face at scale.

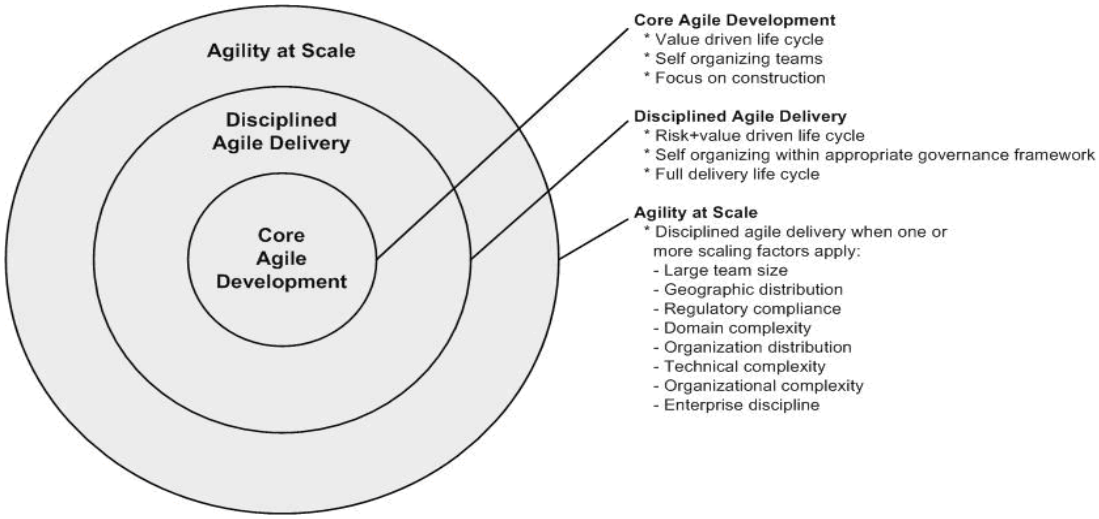


*Fig 1. Overview of the Agile Scaling Model (ASM)*

**Highlights**

***The first step in scaling agile approaches is to move from partial methods to a full-fledged, disciplined agile delivery process. The second step is to recognize your degree of complexity.***

The first step in scaling agile approaches is to move from partial methods to a full-fledged, disciplined agile delivery process. Mainstream agile development processes and practices, of which there are many, have certainly garnered a lot of attention in recent years. They've motivated the IT community to pause and consider new ways of working, and many organizations have adopted and been successful with them. However, these mainstream strategies (such as Extreme Programming (XP) or Scrum, which the ASM refers to as core agile development strategies) are never sufficient on their own; as a result organizations must combine and tailor them to address the full delivery lifecycle. When doing so the smarter organizations also bring a bit more discipline to the table, even more so than what is required by core agile processes themselves, to address governance and risk.

The second step to scaling agile is to recognize your degree of complexity. A lot of the mainstream agile advice is oriented towards small, co-located teams developing relatively straightforward systems. But once your team grows, or becomes distributed, or you find yourself working on a system that isn't so straightforward, you find that the mainstream agile advice doesn't work quite so well – at least not without modification.

IBM Rational advocates disciplined agile delivery as the minimum that your organization should consider if it wants to succeed with agile techniques. You may not be there yet, still in the learning stages. But our experience is that you will quickly discover how one or more of the scaling factors is applicable, and as a result need to change the way you work. Let's explore each of the ASM's scaling categories one at a time.

### Core agile development

Core agile development methods focus on a portion of the overall delivery lifecycle. Table 1 overviews several core agile methods, indicating the purpose or scope of the method as well providing a list of representative practices (the practice lists are not meant to be complete). It's interesting to note that several practices are supported by one or more methods, an indication of the compatibility between the methods. Disciplined agile delivery teams will typically mine the core agile methods for practices and ideas which are then combined to form a more robust process. Each method has its own unique focus and approach, a specific process scope which it addresses, and uses its own terminology (there is some overlap).

*Table 1. Core agile methods*

| Method | Purpose/Scope | Representative Practices |
|---|---|---|
| Agile Data (AD) | AD is a collection of practices which focuses on database development [24]. | • Agile Data Modeling<br>• Continuous Database Integration<br>• Database Refactoring<br>• Database Testing |
| Agile Modeling (AM) | AM is a collection of practices for light-weight modeling and documentation [10]. | • Active Stakeholder Participation<br>• Executable Specifications<br>• Iteration Modeling<br>• Prioritized Requirements (Ranked Work Item List)<br>• Requirements Envisioning |
| Extreme Programming (XP) | XP focuses on software construction and requires significant discipline on the part of practitioners. XP is often mined for construction practices by Scrum teams to address Scrum's lack of technical practices [25]. | • Collective Ownership<br>• Continuous Integration<br>• Pair Programming<br>• Refactoring<br>• Test-First Design<br>• Whole Team |
| Feature Driven Development (FDD) | FDD is a model-driven, short iteration agile software delivery process [26]. | • Development By Feature<br>• Domain Object Modeling<br>• Feature Teams<br>• Individual Class Ownership<br>• Regular Build |
| Scrum | Scrum focuses on project leadership and scope management. Scrum defines a high-level lifecycle for construction iterations and a handful of supporting practices [27]. | • Product Backlog (Ranked Work Item List)<br>• Scrum Meeting (Daily Stand-Up Meeting)<br>• Sprint/Iteration Demo<br>• Retrospectives |

**Highlights**

Scrum and XP are very popular within the mainstream agile community, in part because they are what developers want to hear – developers are at the center, working software is critical, bureaucracy is bad – and in part because they provide developers with a sense of ownership of the process that they follow. The latter is clearly a good thing, but these processes aren't the only thing that developers should be following. For example, AM isn't as popular as other approaches; many agilists like to downplay modeling and documentation, although it's interesting to note that the individual practices of AM often have very high adoption rates within the agile community, often higher than some of the Scrum and XP practices.

### Disciplined agile delivery

The consultants and developers who developed the manifesto did a good job; the manifesto itself conveys a key idea we can apply at this point. As we read, "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." Is it possible that we could improve certain aspects of the manifesto, particularly as it relates to large scale projects? The Agile Manifesto has a software development focus, yet software engineers consider what they build, really, to be solutions. This might include commercial-off-the-shelf (COTS) solutions that don't get built at all, but rather configured and integrated.

***It's great that the core agile approaches describe how to be effective at building software, but if stakeholders don't agree on what needs to be built then it doesn't matter how streamlined construction is. And if it doesn't work with existing infrastructure, then it really isn't much good in practice.***

The Agile Manifesto also has a construction focus. It's great that the core agile approaches describe how to be effective at building software, but if stakeholders don't agree on what needs to be built then it doesn't matter how streamlined construction is. The fact that we're building high quality software is great, but if it doesn't work with our existing infrastructure then it really isn't much good to us in practice. The fact that we've build potentially shippable software is great, but if we can't actually release it easily then it doesn't really matter. Lean software development [6], which complements agile strategies and in many ways explains why they work, tells us to optimize the whole, not just the parts. From the point of view of a single solution there is a little more to its lifecycle than construction. There are pre-construction activities, there are activities around deploying a release into production, there are activities around operating and supporting it once it's in production, and even activities around retiring the system from production.

**Highlights**

***We need to look beyond agile software development and consider the full complexities of solution delivery.***

***Disciplined agile delivery is an evolutionary (iterative and incremental) approach that regularly produces high quality solutions in a cost-effective and timely manner via a risk and value driven lifecycle.***

The point is that we need to look beyond agile software development and consider the full complexities of solution delivery. In fact, our experience is that the first "scaling factor" that organizations face with agile development is lifecycle scope. At IBM Rational we define disciplined agile delivery as:

> *Disciplined agile delivery is an evolutionary (iterative and incremental) approach that regularly produces high quality solutions in a cost-effective and timely manner via a risk and value driven lifecycle. It is performed in a highly collaborative, disciplined, and self-organizing manner within an appropriate governance framework, with active stakeholder participation to ensure that the team understands and addresses the changing needs of its stakeholders. Disciplined agile delivery teams provide repeatable results by adopting just the right amount of ceremony for the situation which they face.*

Let's explore the key differences with this definition over the previous definition:

1. *Full delivery lifecycle* - Disciplined agile delivery processes have lifecycles that are serial in the large and iterative in the small. Minimally they have a release rhythm that recognizes the need for start up/inception activities, construction activities, and deployment/transition activities. Better processes, which I'll discuss shortly, include explicit phases as well. It is very important to note that these are not the traditional waterfall phases – requirements, analysis, design, and so on – but instead different "seasons" of a project. In short, agile projects go through different phases in their life cycles, they are not just purely iterative.
2. *Solutions, not just software* - The term solution is far more robust, and accurate, than the term software. Disciplined agile delivery teams produce solutions, a portion of which may be software, a portion of which may be hardware, and a portion of which may be outside of the technical domain such as the manual processes associated with working with the system.
3. *Risk and value driven lifecycle* - Core agile processes are very clear about the need to produce visible value in the form of working software on a regular basis throughout the lifecycle. Disciplined agile delivery processes take it one step further and actively mitigate risk early in the lifecycle. For example, during project start up you should come to stakeholder concurrence regarding the project's scope, thereby reducing

**Highlights**

significant business risk, and prove the architecture by building a working skeleton of your system, thereby significantly reducing technical risk. They also help with transition to agile, allowing traditional funding models to use these milestones before moving to the finer grained iteration-based funding that agile allows.

4. *Self-organization within an appropriate governance framework* - Self-organization leads to more realistic plans and estimates more acceptable to the people implementing them. At the same time these self-organizing teams must work within an "appropriate governance framework" that reflects the needs of their overall organizational environment: such a framework explicitly enables disciplined agile delivery teams to effectively leverage a common infrastructure, to follow organizational conventions, and to work toward organizational goals.

***Project teams need to work within the governance framework of their organization. Effective governance programs should make it desirable to do so.***

The point is that project teams, regardless of the delivery paradigm they are following, need to work within the governance framework of their organization. Effective governance programs should make it desirable to do so. Our experience is that traditional, command-and-control approaches to governance, where senior management explicitly tells teams what to do and how to do it, don't work very well with agile delivery teams. We've also found that a lean approach to governance based on collaboration and enablement is far more effective [7]. Good governance increases the chance that agile delivery teams will build systems that fit into your overall organizational environment, instead of yet another stand-alone system that increases your overall maintenance burden and data quality problems.

Let's explore why a full delivery lifecycle view is important. The Scrum lifecycle, depicted in Figure 2, focuses on how to organize the work during a construction sprint (in Scrum iterations are called sprints). This lifecycle explicitly depicts several important agile practices – Ranked Work Item Lists (Product Backlog, Sprint Backlog), Time-Boxed Iterations (Sprints), Daily Stand Up Meeting (Daily Scrum Meeting), Retrospective, and Iteration Demo (Sprint Review) – which Scrum has popularized within the agile community.

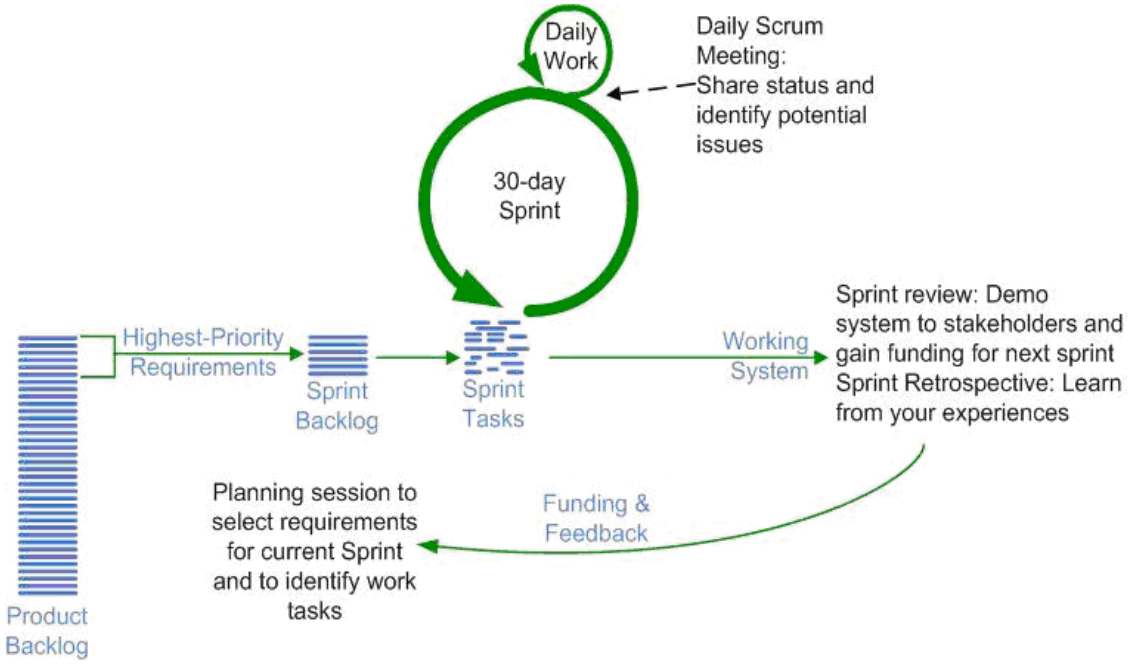


*Fig 2. Scrum construction lifecycle*

The Scrum lifecycle of Figure 2 isn't sufficient to meet all of the needs of software delivery teams, but it is an important foundation from which we can develop a full delivery lifecycle. To see this, consider the disciplined agile delivery lifecycle [8] of Figure 3. In addition to using sensible terminology, for example nobody sprints through a ten kilometer race, this lifecycle expands upon the Scrum construction lifecycle in three important ways:

1. *Explicit project phases* - The mainstream agile mantra is that agile software development is iterative, but the disciplined strategy is to recognize that agile delivery is really iterative in the small and serial in the large [9]. What we mean by iterative in the small is that from the point of view of your daily rhythm the work proceeds iteratively – each day you're likely to iterate back and forth between modeling, testing, programming, and management activities (to name a few). Serial in the large refers to the fact that your release rhythm proceeds through different project phases: at the beginning you focus on initiation or start-up activities, in the middle you focus on construction activities, and in the end you focus on deployment activities. As shown in Figure 3, the agile system delivery lifecycle explicitly reflects this by including the Inception phase. where you do some initial modeling, start putting together your team, gain initial project funding, put together your work

environment (including tools), and even do some initial development; the Elaboration & Construction phases, where you build the system; the Transition phase, where you harden your system and release it into production (or the marketplace); and a Production phase where you operate and support the system. Table 2 lists some of the agile methods which explicitly include phases.

4. *A full range of practices* - The lifecycle illustrated in Figure 3 explicitly includes, and in some cases implies, additional practices followed by disciplined agile teams. This includes initial requirements and architecture envisioning at the beginning of the project to increase the chance of building the right product in the right manner as well as system release practices.
5. *More robust practices* - A critical aspect of Figure 3 is that it explicitly reworks the product backlog of Figure 2 into the more accurate concept of a ranked work item list. Not only do agile delivery teams implement functional requirements, they must also fix defects (found through independent testing or by users of existing versions in production), provide feedback on work from other teams, take training courses, and so on. All of these activities need to be visible in the backlog and planned for, not just functional and non-functional requirements. Having a single work item stack, instead of several stacks (one for each type of work item), proves easier to manage in practice.

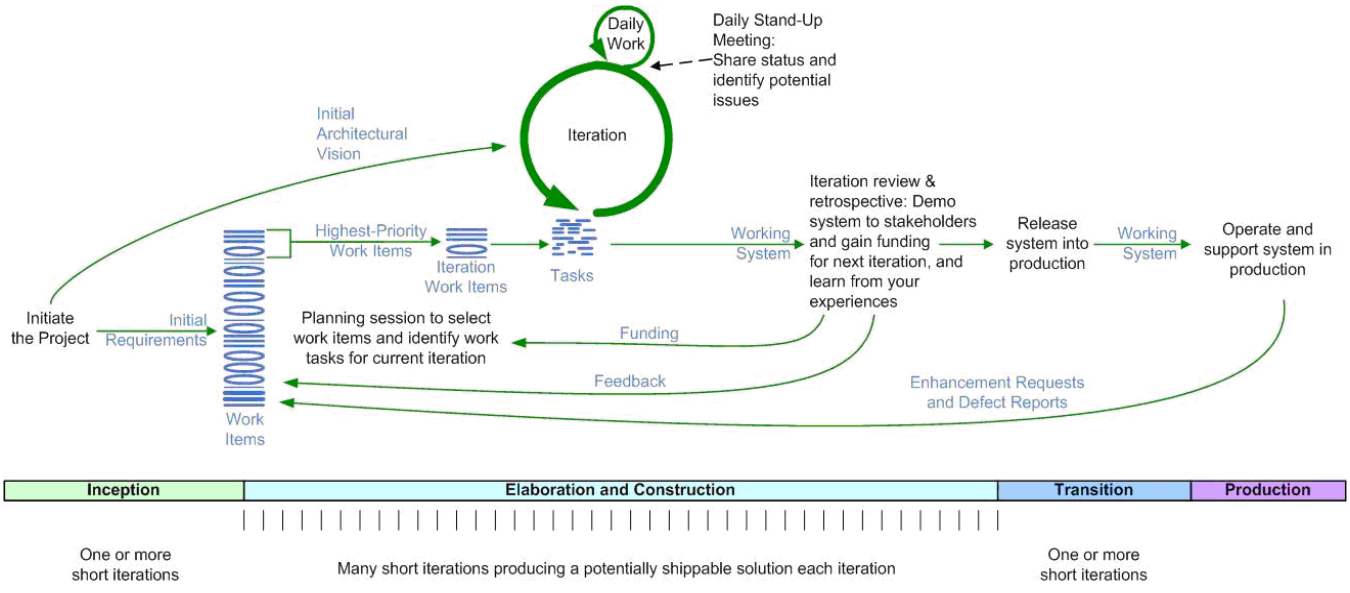


*Fig 3.Agile system delivery lifecycle*

*Table 2. Agile methods with distinct phases*

| Method | Purpose/Scope | Representative Practices |
|---|---|---|
| Agile Unified Process (AUP) | AUP is available via open source (www.ambysoft.com/unifiedprocess/agileUP.html) as a collection of HTML pages, with sparse descriptions written in point-form and links to online articles which provide greater detail. AUP combines and extends practices from Scrum, XP, AM and AD [28]. | • Architecture Envisioning<br>• Continuous Integration<br>• Database Refactoring<br>• Ranked Work Item List<br>• Requirements Envisioning<br>• Test-Driven Development (TDD) |
| Agile With Discipline (AWD) | This is the agile process, which they tailor to meet the unique needs of individual customers, followed by IBM's Accelerated Solution Delivery (ASD) practice. AWD has adopted practices from Scrum, XP, AM, Unified Process, and other processes and has been evolved by the ASD team over the years as they've applied it at scale with customers around the world. | • Continuous Integration<br>• Development Standards<br>• Pair Programming<br>• Refactoring<br>• Reuse<br>• Risk-Value Lifecycle<br>• Test Driven Development (TDD) |
| Eclipse Way | This is the process followed by the people working on Eclipse, an open source, Java-based development platform developed by hundreds of people world wide who are working for dozens of organizations. The "Eclipse project" is actually a program of many project teams, each of which are working on different Eclipse components or plug-ins, and each of which are often distributed themselves. The core team follows a 6 week iteration length, a reflection of the scale of the team, although sub-teams are welcome to adopt shorter iteration lengths as their situation permits (it's common to follow a divisor, 1, 2 or 3 weeks although this is not a requisite) [29]. | • API First (Architecture Envisioning)<br>• Burndown Tracking<br>• Component Centric<br>• Consume Your Own Output<br>• Continuous Integration<br>• Continuous Testing<br>• Feature Teams<br>• Ranked Work Item List |

*Table 2. Agile methods with distinct phases (continued)*

| Method | Purpose/Scope | Representative Practices |
|---|---|---|
| IBM Rational Unified Process (RUP) | RUP is a comprehensive process framework for iterative software delivery which can be instantiated anywhere from a very agile form to a very traditional form as your situation warrants [10, 11]. RUP includes a plethora of practices which are often described in detail, with supporting templates, examples, and guidelines. RUP one of several processes within Rational Method Composer (RMC). | • Concurrent Testing<br>• Continuous Integration<br>• Continuous Testing<br>• Evolutionary Design<br>• Release Planning<br>• Risk-Value Lifecycle<br>• Shared Vision<br>• Test-Driven Development (TDD) |
| Open Unified Process (OpenUP) | OpenUP, the definition of which is available via open source (www.eclipse.org/epf/), combines and extends practices from Scrum, XP, AM and IBM Rational Unified Process (RUP) for small, co-located agile teams which are building business applications. OpenUP practices are described briefly with prose but often backed up with detailed guidelines for anyone needing more information. OpenUP is the sweet spot between AUP's sparse description and RUP's comprehensive description [11]. | • Active Stakeholder Participation<br>• Continuous Integration<br>• Daily Standup Meeting<br>• Ranked Work Item List<br>• Risk-Value Lifecycle<br>• Test-Driven Development (TDD)<br>• Whole Team |

Disciplined agile project teams take the realities of the full system delivery lifecycle into account, not just the "fun stuff" encompassed by the much smaller construction lifecycle. This is important because it helps to make the complexities of software development and delivery explicit to everyone involved. The work required to get a project started is important, can be difficult, and it will often be several weeks before get funding for the construction effort. The agile construction effort itself is more difficult than we are led to believe, let alone the challenges of releasing software into production (there's usually a bit more to it than copying a few files onto a server). Finally, disciplined lifecycles explicitly include a production phase because not only are operations and support staff important

**Highlights**

***Many organizations will develop their own disciplined agile delivery process(es) by combining Scrum, practices from XP, and (sometimes unknowingly) practices from other processes such as AM, AD, and FDD. This strategy works, although it can be expensive and time consuming.***

project stakeholders they will often be the source of requirements changes (in the form of enhancement requests and defect reports) throughout the project. Information Technology Infrastructure Library (ITIL) [12], and IBM Tivoli Unified Process (ITUP) [13] which is based on ITIL, are excellent sources of information pertaining to Production phase activities.

Many organizations will develop their own disciplined agile delivery process(es) by combining Scrum, practices from XP, and (sometimes unknowingly) practices from other processes such as AM, AD, and FDD. This strategy works, although it can be expensive and time consuming compared to starting with a full disciplined agile delivery process. I've performed agile process assessments in dozens of organizations around the world, and whenever I've run into a team claiming to be following Scrum I've found invariably that they've developed a lifecycle very close to what is shown in Figure 3 and sometimes written extensive supporting process material to flesh it out. Even though these organizations had done a good job building their own processes from scratch, many of them recognized that they had wasted significant time and money by doing so; they would have greatly benefited by starting with an existing, more disciplined process.

### Agility at Scale

In the early days of agile, projects managed via agile development techniques were small in scope and relatively straightforward. The small, co-located team strategies of mainstream agile processes still get the job done in these situations. Today, the picture has changed significantly and organizations want to apply agile development to a broader set of projects. They are dealing with problems which require large teams; they want to leverage a distributed work force; they want to partner with other organizations; they need to comply with regulations and industry standards; they have significant technical or cultural environmental issues to overcome; and they want to go beyond the single-system mindset and truly consider cross-system enterprise issues effectively. Not every project team faces all of these scaling factors, nor do they face each scaling factor to the same extent, but all of these issues add complexity to your situation and you must find strategies to overcome these challenges. To deal with the many business, organization, and technical complexities your development organization is facing, your disciplined agile delivery process needs to adapt.

In addition to scaling your lifecycle to address the full range of needs for solution delivery, there are eight more scaling factors that may be applicable. Figure 4 illustrates these scaling factors, explicitly showing that each one represents a range of possibilities, from simple to complex: For each factor the simplest situation is on the left-hand side and the most complex situation on the right-hand side. When a project team finds that all seven factors are close to the left (simple), then their project can be managed in a disciplined agile delivery mode. But when one or more scaling factors moves to the right, they are in an agility at scale situation.

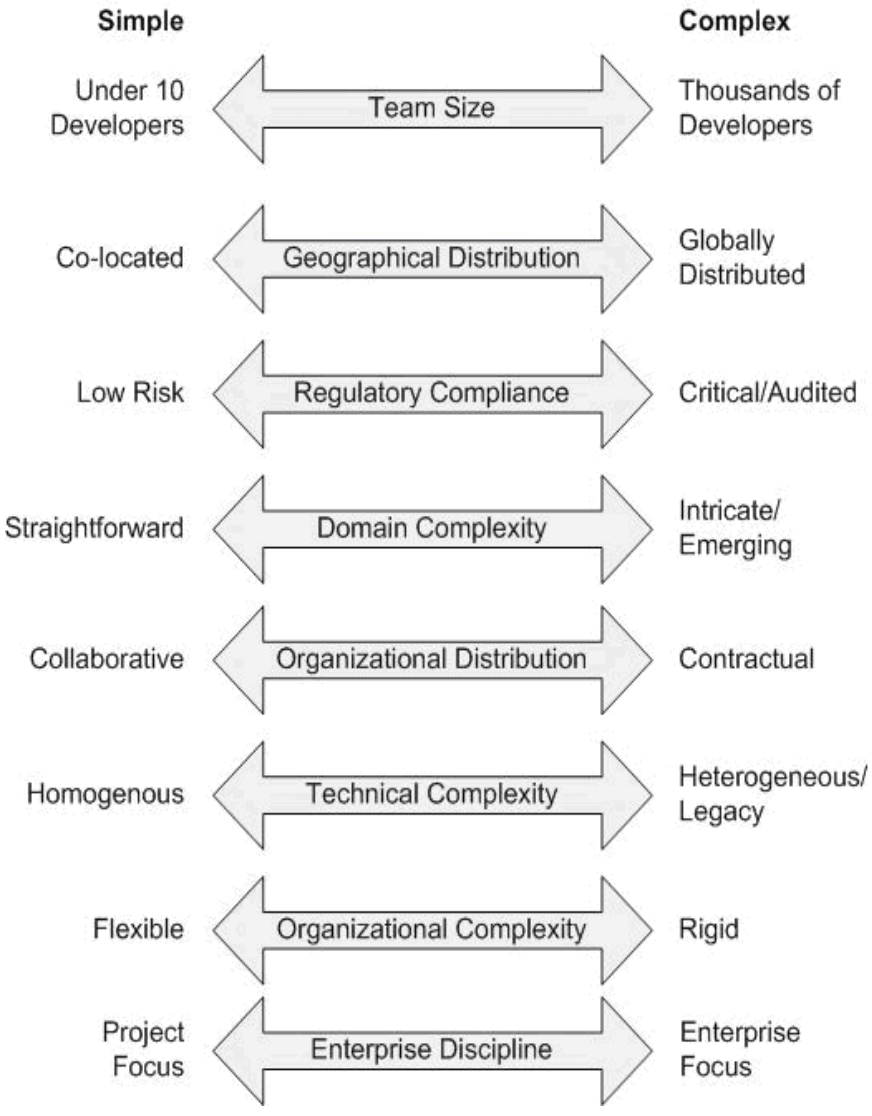


*Fig 4.Potential scaling factors for disciplined agile delivery*

**Highlights**

***Not every project team faces all of these scaling factors, nor do they face each scaling factor to the same extent, but all of these issues add complexity to your situation and you must find strategies to overcome these challenges.***

The eight scaling factors are:

1. *Team size* - Mainstream agile processes work very well for smaller teams of ten to fifteen people (any process often works for such teams), but what if the team is much larger? What if it's fifty people? One hundred people? One thousand people? As your team-size grows the communication risks increase and coordination becomes more difficult. The paper-based, face-to-face strategies of core agile methods start to fall apart.
2. *Geographical distribution* - What happens when the team is distributed – perhaps on floors within the same building, different locations within the same city, or even in different countries? What happens if you allow some of your engineers to work from home? What happens when you have team members in different time zones? Suddenly, effective collaboration becomes more challenging and disconnects are more likely to occur.
3. *Regulatory compliance* - What if regulatory issues – such as Sarbanes Oxley, ISO 9000, or FDA CFR 21 – are applicable? These mandates bring requirements of their own, often imposed from outside your organization in addition to the customer-driven product requirements. There is an increase in the complexity faced by your project team because they must interpret the regulations, which typically describe goals but do not prescribe specific strategies for achieving those goals, and then conform to those regulations appropriately. This may mean that they have to increase the formality of the work that they do and the artifacts that they create.
4. *Domain complexity* - Some project teams find themselves addressing a very straightforward problem, such as developing a data entry application or an informational Web site. Sometimes the problem domain is more intricate, such as the need to monitor a bio-chemical process or air traffic control. Or perhaps the situation is changing quickly, such as financial derivatives trading or electronic security assurance. Philippe Kruchten [32] argues that the rate of change within the domain, the criticality of the system, and the business model are critical contextual factors that affect your software process. More complex domains require greater emphasis on exploration and experimentation, including – but not limited to – prototyping, modeling, and simulation.

**Highlights**

***To deal with the many business, organization, and technical complexities your development organization is facing, your disciplined agile delivery process needs to adapt.***

5. *Organizational distribution* - Sometimes a project team includes members from different divisions, different partner companies, or from external services firms. The more organizationally distributed teams are, the more likely the relationship will be contractual in nature instead of collaborative. For example, in some projects people contributing to requirements, architecture, design, code are actually kept in the dark about the real product for security reasons and cannot even get network access to execute tests on their own work. A lack of organizational cohesion can greatly increase risk to your project due to lack of trust, thereby reducing willingness to collaborate, and may even increase the risks associated with ownership of intellectual property (IP). Many organizations are struggling to rethink their procurement processes and contracts to better build the trust required for successful system delivery within an organizationally distributed environment.
6. *Technical complexity* - Some applications are more complex than others. It's fairly straightforward to achieve high-levels of quality if you're building a new system from scratch, but not so easy if you're working with existing legacy systems and legacy data sources that are less than perfect. It's straightforward to build a system using a single platform, but not so easy if you're building a system running on several platforms or built using several disparate technologies. Sometimes the nature of the problem your team is trying to solve is very complex in its own right, requiring a complex solution.
7. *Organizational complexity* - Your existing organization structure and culture may reflect waterfall[10] values, increasing the complexity of adopting and scaling agile strategies within your organization. To make matters worse, different subgroups within your organization may have different visions for how they should work. Individually the strategies can be quite effective, but as a whole they simply don't align in a common direction. This can dramatically increase the risk to your project because there can be significant overlap in effort, including some work that negates the efforts being performed in parallel by others.
8. *Enterprise discipline* - Most organizations want to leverage common infrastructure platforms to lower cost, reduce time to market, improve consistency, and promote a sustainable pace. This can be very difficult if your project teams focus only on their immediate needs. To leverage common infrastructure, project teams need to take advantage of effective enterprise architecture, enterprise business modeling, strategic

reuse, and portfolio management disciplines. These disciplines must work in concert with, and better yet enhance, your disciplined agile delivery processes. But this doesn't come free. Your agile development teams need to include as stakeholders enterprise professionals -- such as enterprise architects and reuse engineers -- if not development team members in their own right. The enterprise professionals will also need to learn to work in an agile manner, a manner which may be very different compared to the way that they work with more traditional teams.

It is critical to recognize that each scaling factor represents a range of complexities, and that each project team will face a different combination of these complexities. The implication is that they will need to tailor the practices and tools that they adopt to reflect the realities of the situation in which they find themselves in. The first four scaling factors listed – team size, geographical distribution, regulatory compliance, and organizational distribution – are relatively straightforward to address via disciplined work, adoption of appropriate technology, and tailoring of practices to reflect the realities of each scaling factor. The other four scaling factors – domain complexity, technical complexity, organizational complexity, and enterprise discipline – are more difficult to address because environmental complexity often reflects systemic challenges within your organization and enterprise discipline requires a level of maturity that many organizations struggle to achieve (although most desire such discipline).

Addressing scaling factors such as team size, geographical distribution, regulatory compliance, and organizational distribution is fairly straightforward with disciplined practices, integrated tooling, and appropriate team structures. Organizational complexities can be far more difficult to overcome because many of them are cultural or systemic in nature, requiring years of concerted effort to overcome. Particularly challenging cultural issues include, but are not limited to, a serial/waterfall mentality among practitioners, the desire by the business to have accurate estimates and schedules early in the project, over specialization of staff, distrust between groups, and a poor relationship between IT and the business. Technical complexities such as poor quality data sources, poor quality legacy code, legacy code and data sources without corresponding regression test suites, and highly coupled systems can also be challenging to address. Many of these technical debt issues can be paid down in part through refactoring, both code refactoring [14] and database refactoring [15], and investment over time in building up

**Highlights**

***Scaling Factors vs. Complexity Factors***
*Team size, geographical distribution, organizational distribution, and enterprise discipline are typically seen as scaling factors. Regulatory compliance, technical complexity, and organizational complexity are often considered complexity factors, and they are from the point of view of a project team. But when you look at things from the point of view of adopting agile strategies across an organization, they're arguably scaling factors. So, I could either use two terms to describe these two categories or I could simplify things and use a single, albeit imperfect although still "good enough" term. The agile strategy is to favor simplicity over perfection, so I've chosen to use the single term "scaling factor" to represent both concepts.*

**Highlights**

regression test suites. Continuous integration [30], supported by tools such as IBM Rational Automation Framework, can dramatically help you to improve quality by identifying defects earlier in the lifecycle. Environment issues can be identified via the assessment activities within Measured Capability Improvement Framework (MCIF) [16], through self assessment via IBM Rational Self Check, or through automated real-time reporting via IBM Rational Insight, and then addressed through systematic improvement efforts.

Enterprise disciplines, such as enterprise architecture (both business and technical), strategic reuse, portfolio management, human resources, and enterprise administration, can also be challenging to address. The Enterprise Unified Process (EUP) [17] describes how to extend the Unified Process to address enterprise disciplines, although its advice can be applied to any disciplined agile process. Another source of information is Information Technology Infrastructure Library (ITIL). ITIL covers many great techniques, although they are often described in a very heavy and traditional manner; consider mining it for ideas but don't follow ITIL's prescriptions to the letter.

## Implications of ASM

***I often run into several issues. Everyone is interested in the implications for both their process and tooling, particularly when they've had previous experiences trying to scale agile techniques.***

When I work with organizations around the world to help them understand how to apply the ASM, I often run into several issues. Everyone is interested in the implications for both their process and tooling, particularly when they've had previous experiences trying to scale agile techniques. Many organizations, particularly the ones who believe that their existing processes are already "mature," will struggle with the concept of repeatable results. Organizations that have experience with IBM Rational Unified Process (RUP) always want to know whether RUP can be agile (of course it can). In this section I address these three issues.

### ASM and agile practices

The ASM promotes two strategies for tailoring agile development and delivery techniques to meet the challenges of solution delivery at scale. The first strategy is to tailor mainstream agile practices. For example, Scrum's Product Backlog practice can be evolved into Ranked Work Item List to address the challenges of scaling to a full delivery lifecycle [18]. A product backlog is a prioritized list of requirements that an agile project team will implement over time, a very good idea. Yet project teams also need to address defects, team members will be asked to review the work of other teams, team members will go on training, and so on.

**Highlights**

All of these activities are important work items, just as implementing a requirement is an important work item, which much be planned for and then addressed accordingly. So, to reflect the needs of the full delivery lifecycle you really need a ranked work item list, not just a product backlog of requirements.

Consider the practice of holding a Daily Stand Up Meeting. With a small, co-located team, it is fairly straightforward to get everyone together to share their status and identify potential problems that they face. At scale you can follow the same fundamental practice, holding a daily coordination meeting, but you need to tailor it accordingly. For example, large teams will often focus on identifying potential problems instead of focusing on the time-consuming status information that Scrum prescribes, or they'll hold sub-team stand-up meetings and then another overall coordination meeting (e.g. "Scrum of Scrums"). Very often, these sub-teams will simply update a common wiki, teamroom, or database with the results of their stand up meeting. Geographically distributed teams will need to involve electronic tools, even if it's simply a telephone, to aid in coordination. Teams in regulatory situations may need to record the results of the daily stand up meeting.

***Adopt new practices as needed. Large or geographically distributed teams organize themselves as a collection of feature teams, or as a collection of component teams. Teams in regulatory situations may need to adopt practices around formal documentation.***

The second strategy is to adopt new practices as needed.[11] For example, large or geographically distributed teams will follow organizational practices, such as organizing themselves as a collection of feature teams, or as a collection of component teams, or in some cases as a combination of the two strategies. Teams in regulatory situations may need to adopt practices around formal documentation.

### ASM and tooling

ASM makes it very explicit that different project teams face different situations. This is an important point. Different situations require a different combination of tools, the implication being that individual project teams will have their own unique tooling environments. Your organization's tool support team will need to deal with a variety of tooling combinations, which presents its own challenges in turn. For small, co-located agile teams developing a relatively straightforward system (in other words, teams who are in a disciplined agile delivery situation), the tooling strategy can be fairly simple. Stand-alone development tools are often sufficient, although integrated tools are clearly beneficial: manual tools such as whiteboards and index cards for modeling and planning typically work well. However, when you find yourself in an agility at scale situation and one or more scaling factors apply, then you need to change your strategy. Table 3 describes

how each of the scaling factors can affect your tooling strategy, and suggests some IBM products that you may want to consider to enable you to address these factors effectively. Table 3 is meant to be suggestive, not definitive: Because tool selection is situational you may also find that other IBM products are applicable.

*Table 3. Relating the scaling factors to tooling capability*

| Scaling Factor | Tooling Implications | Potential Tools |
|---|---|---|
| Team Size | • Greater integration, particularly around information sharing<br>• Automated metrics collection to enable effective governance<br>• Electronic modeling and planning tools required to share information amongst subteams | • IBM Rational Team Concert<br>• IBM Rational Insight<br>• IBM Rational Project Composer<br>• IBM Rational Requirements Composer<br>• IBM Rational Build Forge |
| Geographic Distribution | • Greater integration, particularly around information sharing<br>• Automated metrics collection to enable effective governance<br>• Electronic modeling and planning tools required to share information across locations | • IBM Rational Team Concert<br>• IBM Rational Insight<br>• IBM Rational Project Composer<br>• IBM Rational Requirements Composer<br>• IBM Rational Build Forge |
| Regulatory Compliance | • Tools should automate compliance as much as possible<br>• Tools should support process enablement to ensure that people comply to critical processes<br>• Automated metrics collection to enable effective governance and provide required reporting<br>• Electronic modeling and planning tools required to create permanent record | • IBM Rational Team Concert<br>• IBM Rational Insight<br>• IBM Rational Build Forge |

*Table 3. Relating the scaling factors to tooling capability (continued)*

| **Scaling Factor** | **Tooling Implications** | **Potential Tools** |
|---|---|---|
| Organizational Distribution | • Automated quality assessment tools, such as static and dynamic code analysis tools<br>• Automated metrics collection to enable effective governance<br>• Security control required governing access to project information, including source code | • IBM Rational Team Concert<br>• IBM Rational Insight<br>• IBM Rational Software Analyzer<br>• IBM Rational AppScan |
| Technical Complexity | • Code and schema visualization tools to understand legacy assets<br>• Enterprise modernization tools to reduce technical complexity<br>• Multi-platform development tools often required | • IBM Rational Team Concert for Z-series<br>• IBM Optim Datastudio |
| Domain Complexity | • Modeling tools to explore the problem domain<br>• Development tools to manage the project assets | • IBM Rational Requirements Composer<br>• IBM Optim Datastudio |
| Organizational Complexity | • Different organizational groups may have different tooling preferences, but may still share assets between teams | |
| Enterprise Discipline | • Project-level tools should co-exist, if not integrate with, enterprise-level tools (e.g. your system/application architecture modeling tool works with your enterprise architecture modeling tool) | • IBM Rational Insight<br>• IBM Rational Focal Point<br>• IBM Rational Asset Manager<br>• IBM Rational System Architect<br>• IBM Infosphere Data Architect |

**Highlights**

***It is critical that teams focus on creating value rather than choosing the latest tools to enhance individual team members' resumes.***

***Agile teams focus on producing repeatable results, such as delivering high-quality software which meets stakeholder needs in a timely and cost effective manner.***

There are two important implications for agile delivery teams. First, it is critical that teams focus on creating value rather than choosing the latest tools to enhance individual team members' resumes. Although different tools may be used based on the scaling profile of the project, the selection of tools to choose from should be pre-defined where appropriate. Second, each delivery team, even when they are small and co-located, must recognize that they are in effect part of a system of systems environment and therefore need to radiate information to a wider community. In short, some tooling beyond the team's local needs may still be necessary to support this reality.

### Repeatable results over repeatable processes

The definition of disciplined agile delivery indicates that disciplined agile teams focus on producing repeatable results, such as delivering high-quality software which meets stakeholder needs in a timely and cost effective manner. It doesn't indicate that disciplined agile delivery teams should follow repeatable processes. The difference is that because each team finds themselves in a unique situation, to be most efficient they need to follow a unique process. That "unique process" may be comprised of a relatively standard lifecycle and common practices such as architecture envisioning, database regression testing, non-solo development, and many others (granted, those practices may be tailored to reflect the situation too). The point is that each team in your organization may follow a different process, albeit processes which share similar components defined by a common process framework, while achieving the results required of them.

The danger with "repeatable processes" is that they grow in size over the years to address all possible situations, and as a result address none of them very well. Imagine a project team that found itself in an agility at scale situation because it was fairly large, and had regulatory compliance concerns. The team tailored its practices to meet their needs, and they were successful doing so. Then another project team came along and found itself in a smaller-scale, disciplined agile delivery situation. An organization focused on repeatable processes might have that team follow the same process that the previous team followed, even though some of the practices had been tailored to meet scaling factors that don't apply. In other words, the repeatable process included some aspects that were overkill for the new team, thereby impacting their ability to deliver in a timely manner or in a cost efficient manner. In the vast majority of organizations, when given the choice, stakeholders prefer to spend the money wisely and have the solution delivered in a timely manner, not to have the team follow a consistently "repeatable process."

**Highlights**

***There's nothing wrong with documenting a standard process framework (just keep it as light as possible) but that doesn't mean it is supposed to be slavishly followed.***

There's nothing wrong with documenting a standard process framework (just keep it as light as possible) but that doesn't mean it is supposed to be slavishly followed. Instead, think of it as a baseline for adaptation and continuous improvement. This is particularly true if you have project teams in widely varying situations because there is no possible way you could define a single "repeatable process" that effectively meets the needs of all of your project teams. In general, if a team is in a disciplined agile delivery situation, then strategies tailored to address one or more of the scaling factors will prove to be more than they need. Similarly, if they're in situation where one or more scaling factors apply, then strategies for small, co-located teams in straightforward will likely prove insufficient. This may explain why some organizations run into trouble with agile approaches; they're following agile advice that might be appropriate in simple situations even though it doesn't make sense to do so in their situation.

### ASM and RUP

The Rational Unified Process (RUP) is an example of an iterative delivery process meant to be tailored to meet the needs of your situation. Sometimes it is tailored to be very heavy and bureaucratic, which I don't recommend doing, and sometimes it is tailored to be very streamlined and agile. For years, IBM Rational has said that RUP done right is agile. RUP is full lifecycle and strives to address many of the scaling factors described in this paper. However, at the time of this writing, from the point of view of the ASM there are four challenges that RUP still needs to address:

1. RUP strives for a common process framework that you tailor, but ASM promotes common practices that you tailor instead. Having a number of smaller, cohesive practices to work with appears to be easier in practice for organizations that are attempting to improve their IT processes. The good news is that RUP is moving in this direction with recent releases of Rational Method Composer (RMC).
2. RUP implicitly targeted many complexity factors, such as team size, geographical distribution, regulatory compliance and environmental complexity, but it does not explicitly indicate where these factors are specifically targeted. This has resulted in guidance that is tough to understand for people who are not process experts. The question: "What parts of RUP are required for my project?" is one that RUP adopters often struggle with. Unfortunately, due to the first point above, many organizations try to come up with one process solution answer for all teams, which increases overall risk.

**Highlights**

3. RUP has become bloated from trying to address multiple complexity factors in a generic way. The introduction of RMC has helped. RUP is one part of the process materials included in RMC's process repository, allowing people to assemble practices for a specific project. However, RUP had fifteen years to grow before this and contains many recommendations that make sense only when a team needs to address a specific scaling factor. Once again, no guidance explicitly states this.
4. RUP's "scale it down" approach was too much for many organizations. This strategy required you to understand the entire process library before you could effectively cut it down to size – a daunting requirement. The agile approach is to "scale up" your process to meet the needs of your situation. Start with something small and valuable, reflect on your experiences, and modify your strategy accordingly. The implication is that you need to be experienced enough to identify potential process improvements, or be willing to experiment with various strategies, until you find what works best for you. More likely what's required is a combination of the two.

The bottom line is that RUP is a great resource that I highly recommend, but please use it knowing that our overall process offerings are still evolving and will continue to do so overtime. Don't let people's bad experiences with inappropriately heavy tailorings of RUP blind you to the value presented by the RMC process repository.

### Become as agile as you can be

***Successful process improvement across an entire organization can prove difficult in practice, often because you confront a wider range of challenges. At IBM Rational we've found strategies that help you increase your chances of success.***

Many organizations have been successful at adopting agile software development approaches [1, 2], in part because most of the focus up to now has been on pilot projects that prove the approach, or on a handful of projects within an organization. However, successful process improvement across an entire organization can prove difficult in practice, often because casting a wider net confronts a wider range of challenges. At IBM Rational we've found that the following strategies can help you to increase your chances of success at improving your software process:

1. *The goal is to get better, not to become agile* - Considering that the focus of this paper is on successfully adopting agile strategies, I realize this advice sounds contradictory. But nobody is going to give you a little gold star for being agile. They might, however, reward you for becoming more effective at system delivery. While agile techniques can help with this, we need to remember that there are still some pretty good ideas out there in the traditional community too.

**Highlights**

***For your continuous improvement efforts to be successful, you first need to identify your business goals and set priorities.***

2. *Have a continuous improvement plan* - For your continuous improvement efforts to be successful, you first need to identify your business goals and set priorities. IBM Rational offers an approach for continuous improvement, called Measured Capability Improvement Framework (MCIF), which helps organizations improve their software and systems delivery in order to increase revenue and lower costs. MCIF applies Rational capabilities, best practices and services to improve software and systems delivery. IBM Health Assessment can help you achieve those goals. IBM Health Assessment helps you navigate and select the right subset of practices, define your current capability (an "as-is" measure), a target capability improvement (a "to-be" measure), and aligns you to a roadmap for you to get from where you are today to your target improvement with measurable feedback all along the route. MCIF is intended to resolve the two predominant failure patterns of past process improvement initiatives: 1) self-inflicting too much process (rather than a subset of incremental practices), and 2) employing subjective rather than objective measures of progress.
3. *Gain some experience* - Adopt agile approaches on one or more medium-risk pilot project(s) to gain both organizational experience and to build expertise within your staff. It's important to expect to run into a few problems because pilot projects never go perfectly.
4. *Explicitly manage your process improvement efforts* - It's fairly easy to succeed at a handful of pilot projects; it's a bit more difficult to permanently adopt meaningful process improvements across your IT organization. A common agile strategy is for a team to regularly reflect on their approach to identify potential improvements, and then hopefully act on those improvements. Within IBM, we've found that teams who explicitly track their progress at adopting improvements are more successful than those who don't. MCIF includes tooling called IBM SelfCheck which helps teams do exactly this [19].
5. *Invest in your staff* - You need to train, educate, and mentor your staff in agile philosophies, processes, practices, and tooling. Focus on the people involved with the pilots at first and train them on a just-in-time (JIT) basis. Don't forget senior management, project management, and anyone interfacing with the pilot team because they need to change the way that they work too. Delivery teams exist within a larger IT ecosystem, the implication being that this wider ecosystem will also need to evolve to reflect the realities of disciplined agile delivery at scale.

**Highlights**

### Parting Thoughts

The first step to scaling agile strategies is to adopt a disciplined agile delivery lifecycle which scales mainstream agile construction strategies to address the full delivery process, from project initiation to deployment into production. The second step is to recognize which scaling factors, if any, are applicable to a project team, then tailor your adopted strategies accordingly to address the range of complexities with the team faces. The scaling factors are:

1. Team size
2. Geographical distribution
3. Regulatory compliance
4. Organizational distribution
5. Technical complexity
6. Domain complexity
7. Organizational complexity
8. Enterprise discipline

***The Agile Scaling Model provides a roadmap for understanding the complexities which you face when adopting and tailoring agile strategies.***

At IBM we've found that many customers find the agile message confusing, in part because of the multitude of voices within the agile community, but moreso because much of the mainstream agile rhetoric often seems to ignore or gloss over many important issues that our customers face on a daily basis. The Agile Scaling Model provides a roadmap for understanding the complexities which you face when adopting and tailoring agile strategies.

### Acknowledgements

I'd like to thank Robert Begg, Anthony Crain, Paul Gorans, Tony Grout, Chris Kolde, Paul Sims, Mike Perrow, and Lynn Thompson for their feedback which was incorporated into this white paper.

### About the Author

Scott W. Ambler is Chief Methodologist/Agile with IBM Rational and works with IBM customers around the world to improve their software processes. He is the founder of the Agile Modeling (AM), Agile Data (AD), Agile Unified Process (AUP), and Enterprise Unified Process (EUP) methodologies. Scott is the (co-)author of 19 books, including *Refactoring Databases*, *Agile Modeling*, *Agile Database Techniques*, *The Object Primer 3rd Edition*, *and The Enterprise Unified Process*. Scott is a senior contributing editor with *Information Week*. His personal home page is www.ibm.com/software/rational/leadership/leaders/#scott and his Agile at Scale blog is www.ibm.com/developerworks/blogs/page/ambler.

## References

1. Dr. Dobb's Journal's July 2009 State of the IT Union Survey - www.ambysoft.com/surveys/stateOfITUnion200907.html
2. Dr. Dobb's Journal's 2008 Project Success Survey - www.ambysoft.com/surveys/success2008.html
3. Agile Manifesto - www.agilemanifesto.org
4. Principles Behind the Agile Manifesto - www.agilemanifesto.org/principles.html
5. Dr. Dobb's Journal's 2008 Modeling and Documentation Survey - www.ambysoft.com/surveys/modelingDocumentation2008.html
6. Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash.* Boston: Addison Wesley.
7. Ambler, S.W. & Kroll, P. (2007). *Lean Development Governance* - https://www.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-ldg
8. Ambler, S.W. (2005). *The Agile System Development Lifecycle (SDLC)* - www.ambysoft.com/essays/agileLifecycle.html
9. Ambler, S.W. (2004). *The Object Primer 3rd Edition: Agile Model Driven Development with UML 2.0.* New York: Cambridge University Press.
10. Ambler, S.W. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process.* New York: Wiley Press.
11. Kroll, P. and MacIsaac, B. (2006). *Agility and Discipline Made Easy: Practices from OpenUP and RUP.* Boston: Addison-Wesley.
12. Information Technology Infrastructure Library (ITIL) Official Site. www.itil-officialsite.com
13. IBM Tivoli Unified Process (ITUP) - www.ibm.com/software/tivoli/governance/servicemanagement/itup/tool.html
14. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code. Menlo Park, CA*: Addison-Wesley Longman.
15. Ambler, S.W. and Sadalage P.J. (2006). *Refactoring Databases: Evolutionary Database Design.* Upper Saddle River, NJ: Addison Wesley.
16. Measured Capability Improvement Framework (MCIF) - www.ibm.com/software/rational/mcif/
17. Ambler, S.W., Nalbone, J., and Vizdos, M. (2004). *The Enterprise Unified Process: Enhancing the Rational Unified Process.* Boston: Addison Wesley
18. Ambler, S.W. (2009). *Product Backlogs at Scale.* www.ddj.com/architect/217701202
19. Kroll, P. and Krebs, W. (2008). *Introducing IBM Rational Self Check for Software Teams* - www.ibm.com/developerworks/rational/library/edge/08/may08/kroll_krebs/index.html
20. Stober, T. and Hansmann, W. (2010). *Agile Software Development: Best Practices for Large Software Development Projects.* New York: Springer Publishing.
21. Larman, C. and Vodde, B. (2009). *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum.* Upper Saddle River, NJ: Addison Wesley.
22. Kessler, C. & Sweitzer, J. (2007). *Outside-In Software Development: A Practical Approach to Building Stakeholder-Based Products.* IBM Press.
23. Pixton, P., Nickolaisen, N., Little, T., and McDonald, K. (2009). *Stand Back and Deliver: Accelerating Business Agility.* Upper Saddle River, NJ: Addison Wesley.
24. Ambler, S.W. (2004). *Agile Database Techniques: Effective Strategies for the Agile Development.* New York: Wiley Publishing.
25. Beck, K. (2000). *Extreme Programming Explained—Embrace Change.* Reading, MA: Addison Wesley Longman, Inc.
26. Palmer, S. R., and Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development.* Upper Saddle River, NJ: Prentice Hall PTR.
27. Beedle, M. & Schwaber, K. (2001). *Agile Software Development With SCRUM.* Upper Saddle River, New Jersey: Prentice Hall, Inc.
28. Ambler, S.W. (2005). The Agile Unified Process - www.ambysoft.com/unifiedprocess/agileUP.html
29. Velzen, T. (2008). *Skillful and Maneuverable: OpenUP and the Eclipse Way* - www.ibm.com/developerworks/rational/library/edge/08/jul08/vanVelzen/
30. Duvall, P. and Matyas, S. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk.* Upper Saddle River, NJ: Addison Wesley.
31. Ambler, S.W. (2007). The discipline of agile. Dr. Dobb's Journal, October 2007 - http://www.ddj.com/architect/201804241
32. Kruchten, P. (2009). *The Context of Software Development* - http://pkruchten.wordpress.com/2009/07/22/the-context-of-software-development/

## Footnotes

[1] Throughout this paper the term process shall also include the terms “method” and “methodology.” These terms are used interchangeably within the IT industry and for the sake of simplicity I have chosen to use the term “process.”

[2] This difference is discussed in, for example, Stober, T. and Hansmann, W. (2010). *Agile Software Development: Best Practices for Large Software Development Projects*. New York: Springer Publishing, and in Larman, C. and Vodde, B. (2009). *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Upper Saddle River, NJ: Addison Wesley.

[3] For a more detailed discussion of the Agile Manifesto, see “Examining the Agile Manifesto” at www.ambysoft.com/essays/agileManifesto.html

[4] Internally within IBM we use the more succinct “Agile is the use of continuous stakeholder feedback to produce high-quality consumable code through user stories (or use cases) and a series of short time-boxed iterations.” This definition isn’t as comprehensive, but instead focuses on several aspects which are critical within our corporate culture and assumes, rightly or wrongly, that people will pick up the rest over time.

[5] The *Dr. Dobb’s Journal* 2008 Project Success survey found that agile teams are in fact more likely to deliver good ROI than traditional teams and more likely to deliver in a timely manner.

[6] Scrum is a form of agile development that includes sets of practices and pre-defined roles for team members. Work is broken into sprints of several weeks’ duration, in which working software is created and improved until project completion. See *http://www.controlchaos.com/* for more information.

[7] Extreme Programming, or XP, is a form of agile development designed to quickly respond to changing customer requirements. It stresses programming in pairs of developers, frequent and extensive code review and testing, and a flat project management structure.

[8] Regression testing, essentially, tests whether changes to existing software have introduced new problems.

[9] Core agile methods such as Extreme Programming (XP) and Scrum require significant discipline for their practitioners to be successful. However, the methods themselves on their own don’t encompass the full range of discipline required for full system delivery. Disciplined agile delivery methods encompass a broader view, acknowledging the greater complexities a particular team may be faced with, and hence require greater business and technical discipline than core agile methods.

[10] “Waterfall” development methods emphasize detailed up-front planning and rigid sequential phases, much like traditional engineering disciplines. In software projects, this approach typically leads to late design changes and, consequently, excessive scrap and rework.

[11] In 2010 a follow-up white paper to this one entitled “Agility at Scale” will be published on IBM.com which will explore in detail how to apply ASM to tailor your agile process to address scaling issues.



IBM Corporation

Software Group

Route 100

Somers, NY 10589

U.S.A.

Produced in the United States of America

December 2009



RAW14204-USEN-00