DISCIPLINEDAGILE CONSORTIUM

White Paper Series

May 2014

Scaling Agile Software Development

Disciplined Agility at Scale

By Scott W. Ambler and Mark Lines

There are two fundamental visions about what it means to scale agile: Tailoring agile strategies to address the scaling challenges – such as geographic distribution, regulatory compliance, and large team size – faced by development teams and adopting agility across your organization. Both visions are important, but if you can't successfully perform the former then there is little hope that you'll be successful at the latter. This paper focuses on how to scale agile solution delivery strategies.
We begin with an overview of how to go about scaling agile delivery. We show how agile methods such as Scrum, Extreme Programming (XP), Kanban, Agile Modeling (AM), and others provide the process building blocks from which an overall agile delivery process can be tailored to meet your needs. We then overview the Disciplined Agile Delivery (DAD) framework, describing how it does the "heavy lifting" regarding putting together all of these process building blocks in a coherent manner. We then describe the complexities faced by agile teams at scale.

With this understanding of what it means to scale agile delivery, we're then in a position to see how to do so. The secret is in DAD's process goal-driven approach which guides you through the process options, and their trade-offs, available to agile teams. We focus on four of DAD's 22 process goals:

1. *Explore Initial Scope*
2. *Identify Initial Technical Strategy*
3. *Move Closer to a Deployable Release*
4. *Coordinate Activities*

These four typically take the brunt of the tailoring to deal with scaling challenges. We work through three different scaling scenarios and describe how an agile team would modify its strategy to succeed in those situations. We finish with an overview of what it means to be a truly agile enterprise, one that can scale agile strategies for both solution delivery and across the entire organization.

Contents

Introduction

Upwards to 90% of IT organizations have adopted agile software development techniques in some manner [1]. Although the results have been positive for the most part they have been varied, with some organizations finding that agile works well for them when the teams are small and face relatively straightforward situations. When this is not the case they find that their teams struggle, either spending an inordinate amount of effort to determine how to be agile in the situation that they face or sometimes even failing. It doesn't have to be this hard. The good news is that many organizations are applying agile techniques at scale and are succeeding in doing so [1]. The Disciplined Agile Delivery (DAD) framework captures the strategies that organizations successful at scaling agile apply in practice, and more importantly provides straightforward guidance for when, and when not, to apply these strategies. In this paper we address five key questions:

1. What does it mean to scale agile solution delivery?
2. What challenges do teams face when scaling agile?
3. What is and why Disciplined Agile Delivery (DAD)?
4. How does a process goal-driven strategy enable scaling?
5. How do disciplined agile teams work at scale?

Scaling Agile Solution Delivery

Let's begin with the end in mind. *Figure 1* summarizes a safe and proven strategy for scaling agile delivery strategies at the team level. There are three features of this strategy:

1. **Basic agile and lean methods.** At the base are methods such as Scrum, Extreme Programming (XP), Agile Modeling, Kanban, Agile Data, and many others. These methods are the source of practices, principles, and strategies that are the bricks from which a team will build its process.
2. **Disciplined Agile Delivery (DAD).** Building on mainstream methods is the DAD process decision framework, providing an end-to-end approach for agile software delivery. DAD provides the process mortar required to combine the process bricks, effectively doing the "heavy lifting" to describe how all of these great agile strategies fit together.
3. **Agility at scale.** Teams operating at scale apply DAD in a context driven manner to address the scaling factors which they face. These teams may be large, they may be geographically distributed in some way, they may face compliance constraints, they may be addressing a complex domain or technical environment, or they may be organizationally distributed in some manner. And usually combinations thereof. You will soon see that without the solid foundation provided by DAD, agility at scale is incredibly difficult to achieve.

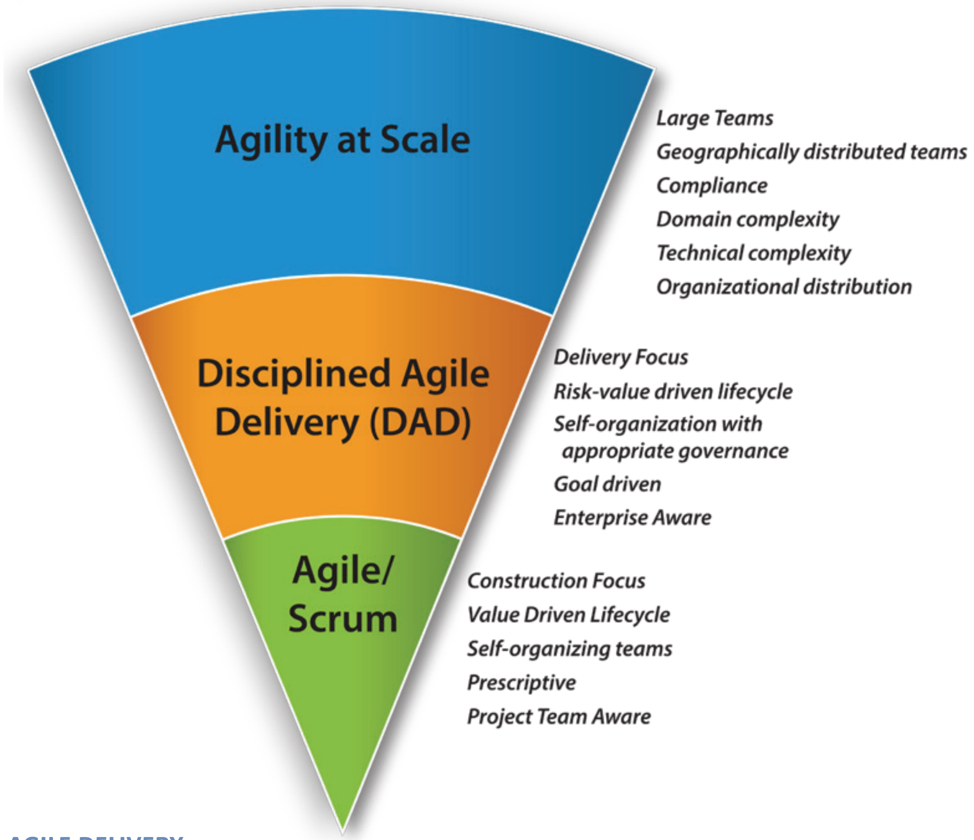

FIGURE 1. SCALING AGILE DELIVERY.

CHALLENGES WITH MAINSTREAM AGILE AND SCALING

The majority of organizations have adopted, or are in the process of adopting, agile software development techniques [1]. For the most part this is working out well for them, with organizations reporting that on average agile teams are producing better quality, achieving higher levels of stakeholder satisfaction, quicker time to delivery, and better return on investment (ROI) compared with traditional strategies [2]. However, organizations are also experiencing challenges with applying agile strategies at scale [1]. We believe that this happens for a variety of reasons because mainstream agile methods are:

1. **Software focused instead of solution focused.** The Agile Manifesto [3] promotes the philosophy that working software is the primary measure of success, a philosophy that is taken to heart by many agilists. However, software is only part of the picture. "Software development teams" are also dealing with hardware related issues, writing supporting documentation, evolving the business process around the usage of the system, and sometimes even evolving the organizational structure of the people involved in using the system. Shouldn't we explicitly consider all of those issues, not just software?
2. **Construction focused at the expense of delivery.** Agile methods such as Scrum focus on the Construction portion of the lifecycle, yet agile teams in practice still need to perform initiation activities as well as release activities. Without advice for the full delivery lifecycle, they are likely to fall victim to adopting a Water-Scrum-Fall approach [4,5]. They are also likely to forgo the benefits provided by a bit of up-front, strategic thinking. Similarly, integrating your development efforts effectively with operations and support processesbecomes difficult, even though it is a key aspect of your overall DevOps strategy.
3. **Oriented towards small teams in relatively straightforward situations.** Much of the agile advice is oriented towards small teams of up to ten people, who are either co-located or near located, who have ready access to their primary stakeholders, and who are working on software that can be easily organized into a series of small releases. What about large teams? What about geographically distributed teams? What about teams working in complex situations? Can't they be agile too?
4. **Prescriptive at the expense of flexibility.** Despite the marketing rhetoric agile methods such as Scrum are rigidly prescriptive. For example, in

Scrum there is one way to address changing requirements called a product backlog, there is one way to coordinate activities within a team called a daily Scrum meeting, and so on. And the Scrum community is very clear that this is the way it has to be done, otherwise you're following a dreaded "Scrum But" strategy. Surely teams in different situations will work in different ways, and surely they would need several options to choose from to achieve these goals? Indeed, proponents of lean techniques have argued that iteration-based approaches like Scrum are not always the best option.

5. **Too narrowly defined.** Agile methods tend to focus on a portion of the overall delivery process. For example Agile Modeling focuses on modeling and documentation practices, Agile Data on database development techniques, and Scrum on leadership and requirements change management. Having focused methodologies like this is wonderful, but they all leave it to you to figure out how to fit them together into a cohesive whole that is right for you. In reality too much of this work is left to you and most teams struggle. Most people don't have the breadth of expertise across the myriad of agile methods to make sense of the options.
6. **Team focused at the expense of the overall enterprise.** Mainstream methods such as Scrum stress the value of sheltering the team from external distractions, in effect encouraging "silo development team" behavior that ignores the larger enterprise picture. In many cases teams use this as an excuse to not collaborate with other delivery teams and enterprise groups resulting in inconsistencies and miscommunications. This in turn can lead the team to unknowingly increase technical debt, miss opportunities for reuse, and even implement functionality that exists in other products.

Mainstream agile methods provide a great starting point but require significant effort to make them scalable. We need something better.

Disciplined Agile Delivery

To address these challenges the Disciplined Agile Delivery (DAD) process decision framework provides a more cohesive approach to agile solution delivery [6]. To be more exact, here's a definition:

> "The Disciplined Agile Delivery (DAD) process decision framework is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value delivery lifecycle, is goal-driven, is enterprise aware, and is scalable."

There are several key aspects of the DAD framework that support scaling agile delivery:

1. **Solution focused.** The DAD framework explicitly recognizes that teams work on solutions that involve software, hardware, supporting documentation, the business process and organization structure. This is important for scaling because it puts teams in a better mindset to understand, and then fulfill, the full range of needs of their stakeholders.
2. **Full delivery lifecycle.** As *Figure 2* depicts, DAD promotes a full end-to-end delivery lifecycle from team initiation all the way to delivering the solution to your stakeholders. Unlike other agile methods, DAD doesn't prescribe a single lifecycle because it recognizes that one process size does not fit all. In addition to the Scrum-based lifecycle there is also a lean lifecycle, a continuous delivery lifecycle, and even an exploratory lifecycle (think Lean Startup) [7]. This is important for scaling because teams need to understand how all aspects of development – architecture, analysis, testing, programming, design, and so on – fit together from end-to-end if they are to tailor their approach to address the challenges that they face at scale.
3. **Process goal-driven.** DAD includes advice about technical practices such as those from Extreme Programming (XP) as well as the modeling, documentation, and governance strategies missing from both Scrum and XP. But, to avoid the problems with prescriptive approaches seen in other agile methods, the DAD framework takes what we call a goal-driven approach. In doing so DAD provides contextual advice regarding viable alternatives and their trade-offs, enabling you to tailor DAD to effectively address the situation in which you find yourself. By describing what works, what doesn't work, and more importantly why, DAD provides the guidance that is critical to successfully scale agile delivery approaches.
4. **Enterprise aware.** DAD teams work within your organization's enterprise ecosystem, as do all other teams. Typically there are existing systems already in production and minimally your solution shouldn't impact them. Better yet your solution will hopefully leverage existing functionality, services, and data available in production. You will

often have other teams working in parallel to your team, and you may wish to take advantage of a portion of what they're doing and vice versa. Your organization may be working towards business or technical visions to which your team should contribute. Hopefully an agile governance strategy exists which should enhance what your team is doing, improve project visibility and reduce risk.

What Does it Mean to Scale Agile Delivery?

What scaling factors should we consider when tailoring our approach? Figure 3 summarizes the six scaling factors of the Software Development Context Framework (SDCF) [8] which is based on Philippe Kruchten's work on "situational agility" [9]. The figure indicates the range of each of its six factors. On the left-hand side is the simple extreme and on the right-hand side the challenging extreme. Any given team will find itself somewhere on all six scaling factors, hopefully closer to the simpler extreme on the left than on the right. In the various IT surveys over the years we have found evidence that organizations are applying agile at all levels of scale, most recently in the 2012 Agility at Scale survey [1]. Let there be no doubt that organizations are attempting to scale agile delivery.

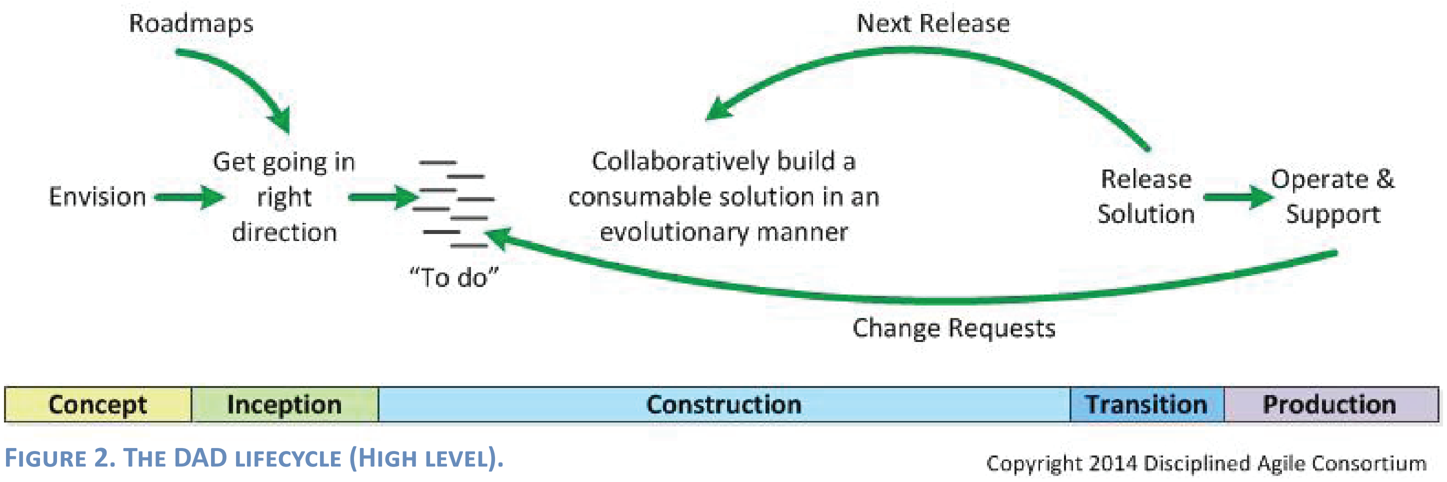
Roadmaps
Next Release
Envision
Get going in right direction
"To do"
Collaboratively build a consumable solution in an evolutionary manner
Release Solution
Operate & Support
Change Requests
Concept
Inception
Construction
Transition
Production
Copyright 2014 Disciplined Agile Consortium

Figure 2. The DAD lifecycle (High level).

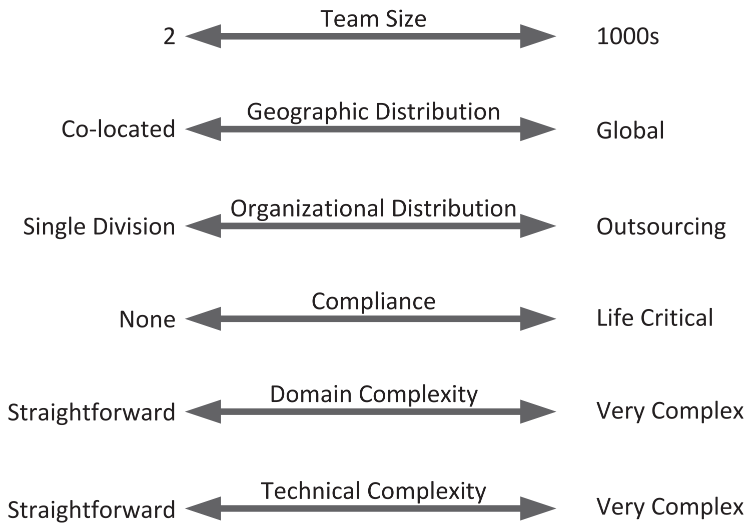

Figure 3. Scaling factors faced by agile teams.

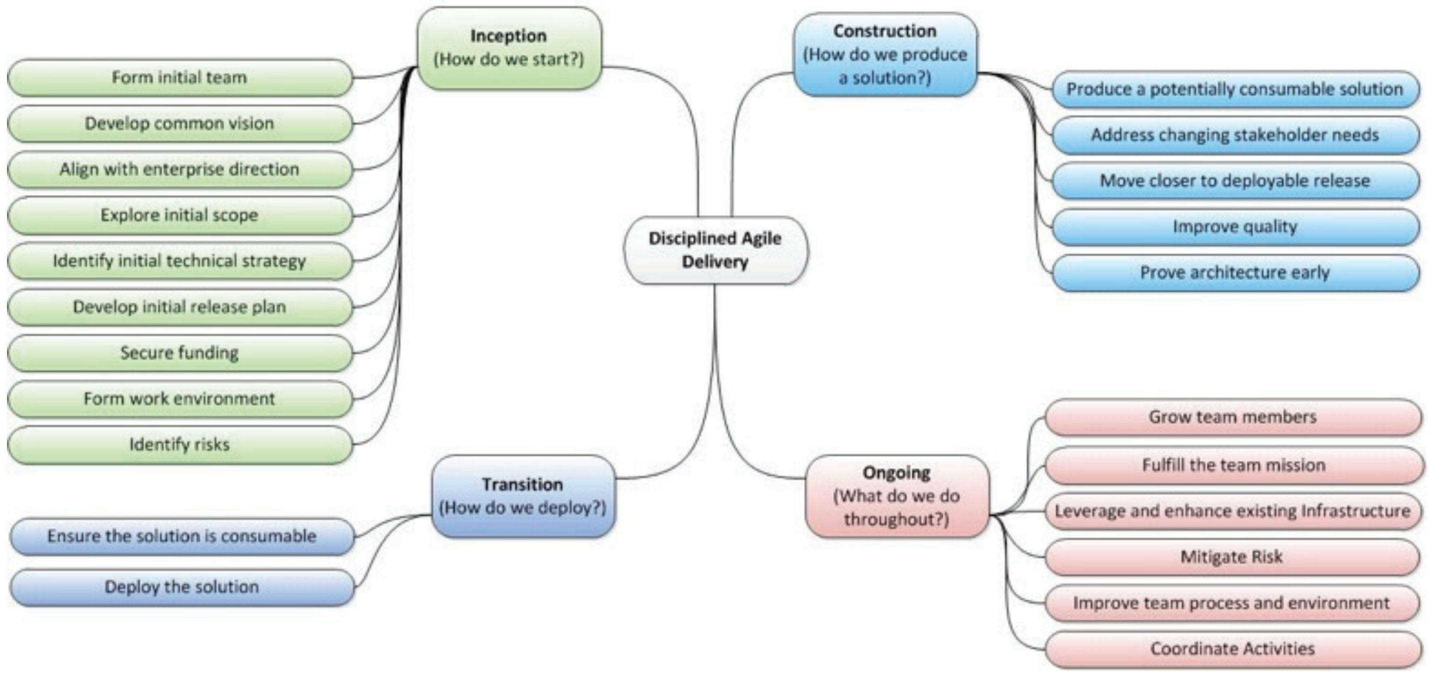
Inception
(How do we start?)
Form initial team
Develop common vision
Align with enterprise direction
Explore initial scope
Identify initial technical strategy
Develop initial release plan
Secure funding
Form work environment
Identify risks
Construction
(How do we produce a solution?)
Produce a potentially consumable solution
Address changing stakeholder needs
Move closer to deployable release
Improve quality
Prove architecture early
Disciplined Agile Delivery
Transition
(How do we deploy?)
Ensure the solution is consumable
Deploy the solution
Ongoing
(What do we do throughout?)
Grow team members
Fulfill the team mission
Leverage and enhance existing Infrastructure
Mitigate Risk
Improve team process and environment
Coordinate Activities

Figure 4. The Goals of Disciplined Agile Delivery.

A Goal-Driven Approach to Scaling

DAD's process goal-driven approach encourages your teams to avoid being prescriptive and thereby be more flexible in your approach. For example, where Scrum prescribes a value-driven Product Backlog approach to managing requirements DAD instead says that during Construction you have the goal of addressing changing stakeholder needs. DAD also indicates that there are several issues surrounding that goal that you need to consider, and there are several techniques/practices that you should consider adopting to do so. DAD goes further and describes the advantages and disadvantages of each technique and in what situations it is best suited for. Yes, Scrum's Product Backlog approach is one way to address changing stakeholder needs but it isn't the only option nor is it the best option in many situations. *Figure 4* shows the goals that are consistent with any type of agile team, whether it is custom development or implementing a packaged application.

The interesting thing is that with a goal-driven approach it becomes much easier to understand how to scale agile. Depending on the context of the situation that a team finds itself in you will address each goal differently. The strategy for a small, co-located team facing a fairly straightforward situation in a non-regulatory environment works well for that team, the same strategy prescribed to a team in a different situation would put that team at risk of failure. Instead of prescribing a single way of working that is optimized for a specific situation we need to instead allow, and better yet enable, teams to adopt strategies that reflect the context of the situation that they face.

We've found that four of the twenty-two process goals seem to take about 80% of the tailoring impact. These goals are:

1. Explore Initial Scope
2. Identify Initial Technical Strategy
3. Move Closer to a Deployable Release
4. Coordinate Activities

In the following section we describe considerations for each of these goals as they relate specifically to the challenges at scale. We do not attempt to describe each of the various process choices for all types of projects as they are already well described in both the DAD book [6] and on the DAD blog (DisciplinedAgileDelivery.com).

Explore Initial Scope

When a disciplined agile project or product team begins a release, one of the process goals which they will likely need to address is Explore Initial Scope [10], the goal diagram for which is shown in *Figure 5.* This is sometimes referred to as initially populating the backlog in the Scrum community, but as you'll soon see there is far more to it than just doing that. This is an important goal for several reasons. First, your team needs to have at least a high level understanding of what they're trying to achieve, they just don't start coding. Second, in the vast majority of organizations IT delivery teams are asked fundamental questions such as what are you trying to achieve, how long will it take, and how much will it cost. Having an understanding of the scope of your effort is important input into answering those sorts of questions.

Let's review some of the basic strategies depicted by the process goal diagram when you are identifying your initial scope. You have several process factors that you should address. To what level of detail will you capture the requirements, if at all? Your approach to managing your work during Construction, such as Scrum's product backlog or perhaps a leaner approach using a work item pool, will drive some of your decisions around level of detail – the more flexible your approach to managing change the less detail you are likely to need in any up-front requirements. What views will you consider? DAD recommends starting with at least a combination of usage modeling, domain modeling, and non-functional requirements. Many teams will find that informal modeling sessions around whiteboards will be sufficient, although sometimes more formal modeling sessions, such as Joint Application Design (JAD) strategies or stakeholder interviews will work best. Finally, how will non-functional requirements pertaining to availability, security, performance, and many other factors be addressed? How do scaling issues affect the manner in which you fulfill your goal of identifying your initial scope? Your strategies will be driven by four of the six scaling factors described above, namely; domain complexity, regulatory compliance, geographic distribution, and organizational distribution. Obviously the greater the domain complexity the greater the need to invest in initial requirements modeling, although that doesn't necessarily imply that you need to create massive requirements documents. Depending on the regulations, and we highly recommend that you read the regulations, you may need more formal requirements documentation and a more formal approach to requirements change management. Geographic distribution may require you to invest more effort in capturing and communicating the requirements,

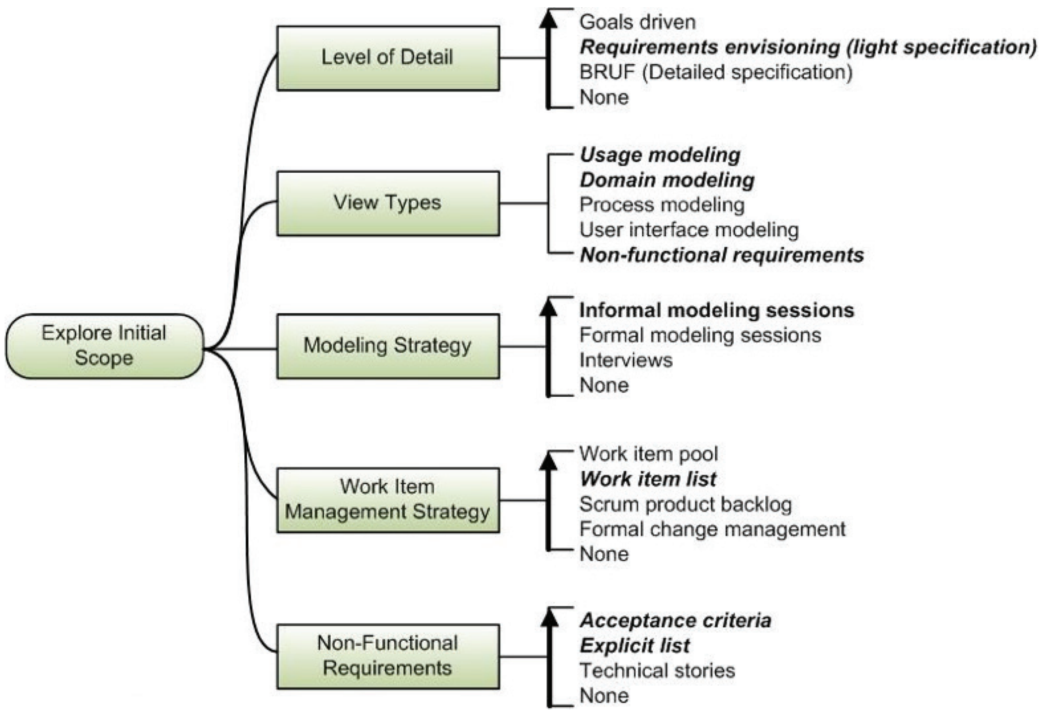

Figure 5. Process goal diagram: Explore Initial Scope.

Process Goal Diagram Notation

The rounded rectangle indicates a process goal, the squared rectangles indicate process factors (sometimes called issues) that you may need to consider, and the lists in the right hand column represent potential strategies or practices that you may choose to adopt to address those issues. The lists with an arrow to the left are ordered, indicating that in general the options at the top of the list are more preferable from an agile point of view than the options towards the bottom. The highlighted options (bolded and italicized) indicate default starting points for teams looking for a good place to start but who don't want to invest a lot of time in process tailoring right now. Each of these practices/strategies has advantages and disadvantages, and none are perfect in all situations, which is why it is important to understand the options you have available to you. A typical team will make hundreds of process decisions and these diagrams can be used to ensure that the various options are considered.

although sophisticated teams recognize that they are better off to fly key people together for initial modeling and planning efforts. Organizational distribution, particularly where you are working with one or more external service providers, may also require a more sophisticated approach to exploring the initial scope. Later in this paper we walk through several case studies and discover in greater detail how this process goal is affected at scale.

Identify Initial Technical Strategy

Another process goal that teams need to address is Identify Initial Technical Strategy [11], *Figure 6*. This is sometimes referred to as initial architecture envisioning or simply initial architecture modeling. This is an important process goal for several reasons. First, the team should think through, at least at a high level, their architecture so as to identify a viable strategy for moving forward into Construction. A little bit of up-front thinking can increase your effectiveness as a team by getting you going in a good direction early in the lifecycle. It can also help to avoid injection of unnecessary technical debt as a result. Second, the team should strive to identify the existing organizational assets, such as web services, frameworks, or legacy data sources that they can potentially leverage while producing the new solution desired by their stakeholders. By doing this you increase the chance of reuse, thereby avoiding adding technical debt into your organizational ecosystem, and more

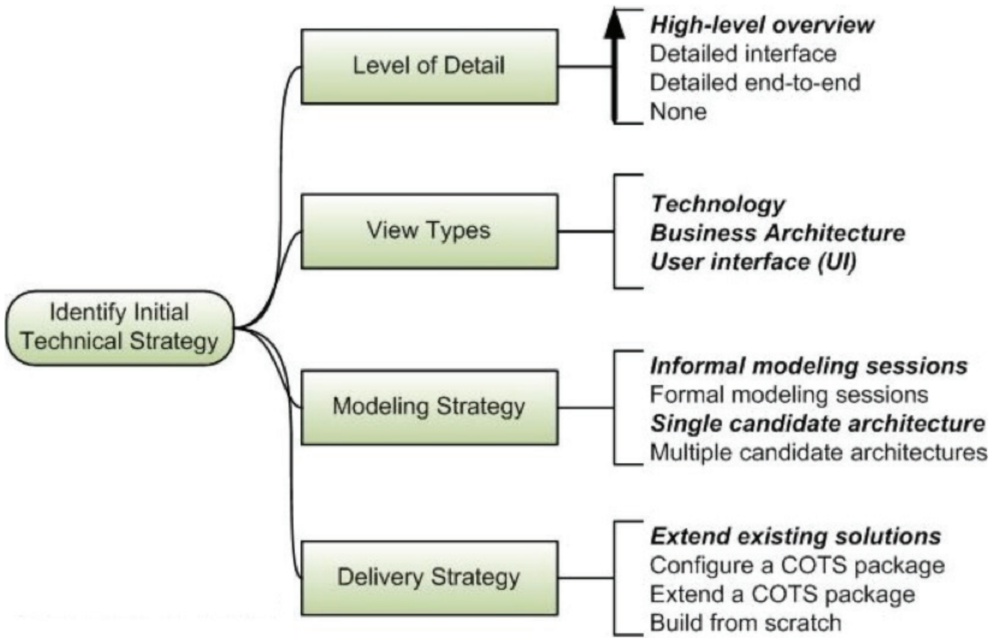
Identify Initial Technical Strategy
Level of Detail
High-level overview
Detailed interface
Detailed end-to-end
None
View Types
Technology
Business Architecture
User interface (UI)
Modeling Strategy
Informal modeling sessions
Formal modeling sessions
Single candidate architecture
Multiple candidate architectures
Delivery Strategy
Extend existing solutions
Configure a COTS package
Extend a COTS package
Build from scratch

Figure 6. Process goal diagram: Identify Initial Technical Strategy.

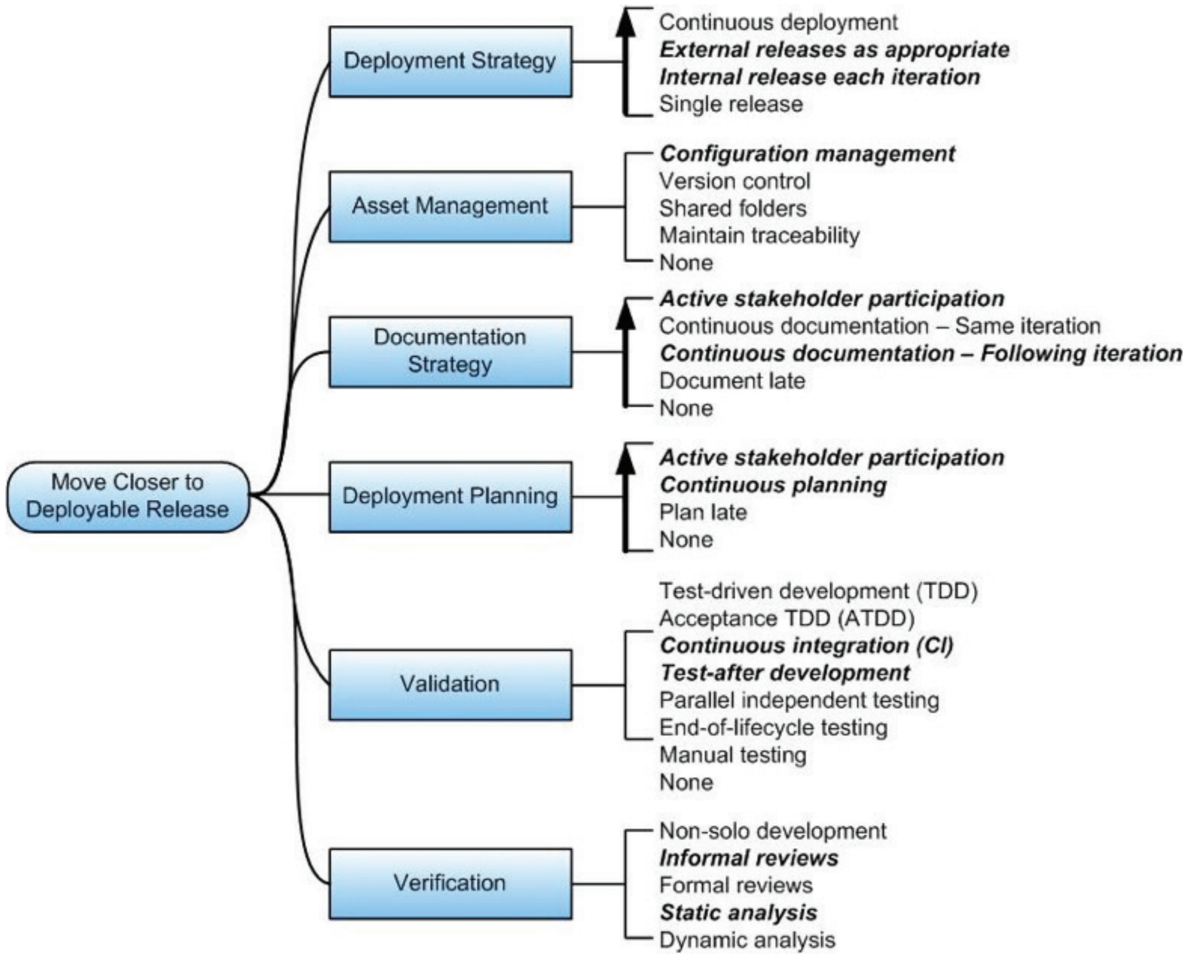

Figure 7. Process goal diagram: Move Closer to a Deployable Release.

importantly you reduce the time and cost of delivering a new solution as the result of reuse. You will do this by working with your organization's enterprise architects, if you have any. This is an aspect of DAD's philosophy of working in an enterprise aware manner.

When you are identifying the potential technical strategy(s) you have several process factors that you should address. As with initial scoping how much detail you go into when documenting the architecture, the views that you create, and your approach to modeling are important considerations. Furthermore, will you be considering one or more candidate architectures and what is your overall delivery strategy? Will your team be building a solution from scratch, evolving an existing solution, or working with a commercial off the shelf (COTS) package?
With regard to scaling challenges, your approach to addressing this goal is particularly affected by the scaling factors of technical complexity, regulatory compliance, geographic distribution, and organizational distribution. The greater the technical complexity a team faces the more effort they are likely to need to invest in initial architectural modeling. Regulatory compliance will likely motivate a more sophisticated approach to addressing architectural issues and risks. Geographic distribution is likely to motivate your team to specify the interfaces of architectural components in greater detail, enabling dispersed or distributed team members to work on those components in greater isolation and thereby reducing collaboration overhead. These specifications may be in the form of documents or better yet executable tests. Organizational distribution may motivate your team to create an architecture where some components or subsystems are completely isolated from others, enabling you to restrict the access that external organizations have to your intellectual property (IP).

Move Closer to a Deployable Release

One of the process goals that a disciplined agile team must address during Construction is Move Closer to a Deployable Release [12], see *Figure 7*. Ideally, a team will continually move closer to having a version of their solution that provides sufficient functionality to its stakeholders. This implies that the solution is a deployable release which is a minimally viable product (MVP) that adds greater business value than its cost to develop and deploy.

Move Closer to a Deployable Release is an important process goal for several reasons. First, it encompasses the packaging aspects of solution development (other important development aspects are addressed by its sister goal Produce a Potentially Consumable Solution). This includes artifact/asset management options such as version control and configuration management as well as your team's deployment strategy. Second, it provides deployment planning options, from not planning at all (yikes!) to planning late in the lifecycle to the more DevOps-friendly strategies of continuous planning and active stakeholder participation. Third, this goal covers critical verification and validation (V&V) strategies, many of which push testing and quality assurance "left in the lifecycle" so that they're performed earlier and thereby reducing the average cost of fixing any defects

This process goal is primarily affected by the scaling factors of domain complexity, technical complexity, and compliance. For example, the more technically complex the environment you face the greater the need to adopt both continuous deployment (CD) and continuous deployment planning strategies. Either domain or technical complexity will motivate a team to adopt more sophisticated V&V strategies. Regulatory compliance, and greater domain or technical complexity, may motivate your organization to create an independent test team which takes on more complex or expensive forms of testing that the delivery teams will struggle with otherwise. Compliance concerns may also motivate more sophisticated approaches to asset management as well as documentation.

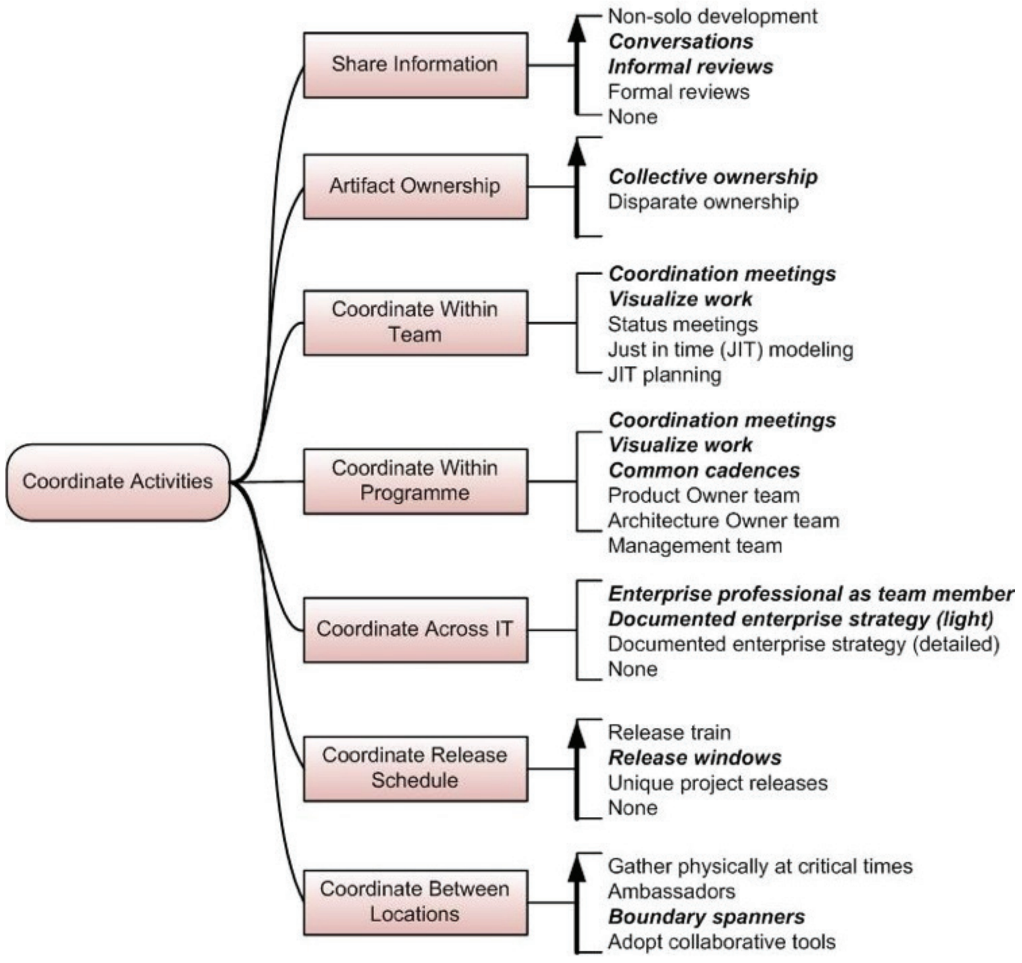

Figure 8. Process goal diagram: Coordinate Activities.

Coordinate Activities

The process goal diagram for the ongoing goal Coordinate Activities [13] is shown in *Figure 8*. This diagram is interesting for several reasons. First, some of the issues are team focused, in particular Artifact Ownership and Coordinate Within Team. Second, several issues reflect the fact that DAD teams are enterprise aware and thus describe strategies to coordinate with others external to the team. For example, your team may need to coordinate with your organization's enterprise architects and operations staff, potentially adopting some of the strategies captured by Coordinate Across IT (and you are also likely to do so via Share Information strategies). If your organization has a release group then your team may need to adopt one or more Coordinate Release Schedule strategies (or, if there's no release team then your team will still need to coordinate releases with other delivery teams, and with your operations team, somehow). Third, several issues address scaling. For example, large teams (often called programs) will find that they need to adopt strategies called out by Coordinate Within Program. Teams that are geographically or organizationally distributed will need to consider strategies from Coordinate Between Locations. Naturally if you don't face a scaling issue such as geographic distribution then the issue Coordinate Between Locations isn't something you need to consider.

Let's work through how the scaling factors could affect the way that you tailor your agile process to address the process goal Coordinate Activities. A small co-located team may choose to coordinate their activities via conversations, visualizing their work via task boards and whiteboard sketches, regular coordination meetings, and collective ownership of artifacts. A large team that is organized into several subteams may also need to adopt a leadership team where the team leads regularly coordinate project management issues, the product owners regularly coordinate requirements management issues, and the architecture owners regularly coordinate technical issues across the subteams. A team that is geographically distributed may find that they need to have people that act as boundary spanners (people who act as key liaisons at each site) and even invest in flying some people to meet together physically at key points in the project. A team in a regulatory environment may find that they need to share some information via formal reviews. A team facing a technically complex environment is likely to adopt the architecture owner team portion of a leadership team and a team facing a complex domain is likely to have a product owner team. A team where some of the work is outsourced to different vendors may choose to adopt a disparate ownership strategy for some project artifacts.

Agility at Scale: Context Counts

Now that we understand the scaling factors potentially faced by agile teams, and how these factors can affect the way we address various process goals, we're in a position to discuss the realities of scaling agile strategies within a team. As described above, organizations first begin experimenting with agile through adopting methods such as Scrum with a few ideas from XP, typically running a few pilot projects to learn about the fundamentals of agile. A common realization is that teams need to go far beyond Scrum, which is exactly what Scrum's proponents tell you, and start adding in techniques from other sources to address the issues that Scrum doesn't. Until recently the only recourse that organizations had was to formulate their own process, often still calling it Scrum, an endeavour that typically proves time consuming, expensive, and difficult when agile process expertise isn't available. But now organizations have the DAD

DAD and SAFe

We are often asked about the relationship between DAD and the Scaled Agile Framework (SAFe) [14]. SAFe is an instance of a set of process decisions made for a specific approach to scaling. SAFe is prescriptive in many ways. For example it requires the use of multiple teams working in parallel as release trains with common cadences with regard to when their iterations start and end. It can in practice be quite difficult for a group of teams to be able to reliably deliver a given set of features on a common date. DAD suggests that the use of common cadences is a process choice, but does not prescribe it as the only reasonable strategy. And while multiple teams working in parallel may make sense, we have seen other patterns such as one large team, or a medium sized core team with several sub-teams work successfully. Scenario 'A' depicts an example of how DAD can be used to arrive at a set of choices that to a large degree mirror the SAFe framework's defined processes.

Scaling Factor	Situation Faced
Team Size	106 total team members available for this initiative
Geographic Distribution	Team members are distributed across 3 offices within 2 time zones of each other
Compliance	Internal compliance only
Organizational Distribution	Business stakeholders are willing to relocate to the team work areas
Domain Complexity	The problem is a bit challenging although “business as usual” for this organization
Technical Complexity	The team will work with known technologies and leverage existing infrastructure wherever appropriate.

Table 1. The situation faced by the team.

framework, a hybrid approach with a lot of the hard thinking done about how these agile techniques all fit together and when (and when not) to use each one. With DAD’s goal-driven approach, it becomes easy to tailor DAD to address the scaling factors appropriately.

So let’s consider three hypothetical scenarios where the realties and challenges of scaling require some explicit process decisions to be made. The following three scenarios depict examples of process choices that we might have made, however you may have chosen different strategies. For each scenario we provide an example of how the goal-driven approach to process decision making can result in the creation of an instance of the DAD framework for the context at hand.

The tables below, which depict these process decisions are not meant to be comprehensive. There are many more decisions that need to be made for each of the twenty-two DAD goals. We have focused on the goals and the decisions related to various factors for scaling agile successfully.

Scenario A: Large Development Team

For this scenario describes a large software development initiative for which building a large number of features in parallel is required to meet an aggressive timeline. *Table 1* describes the scaling challenges that this initiative faces.

As part of DAD’s goal of Form Initial Team we formed eleven teams consisting of between seven and twelve team members with features allocated from the program backlog to each of their respective work item lists. We use the existing organizational structure consisting of distributed teams across three offices in different cities with some team members working part-time at home.

Table 2 depicts the process decisions for each of the key goals related to this large-scale initiative.

Scenario B: Geographically and Organizationally Distributed Team

The customer organization, a large Canadian financial institution, has a team in Toronto Canada and they are outsourcing most aspects of the development of a new mobile application to a service provider in Pune India. There is a time zone difference of 10-½ hours between the two cities. The primary stakeholders, the Product Owner (PO), and two business analysts who work with the PO are based in Toronto. Toronto also has an IT team made up of a team lead and two senior testers. In Pune

DAD Goal	Factor	Process Decision	Strategy
Explore Initial Scope	Level of Detail	**Requirements envisioning (light specification)**	The Chief Product Owner will facilitate the creation of a Vision for the team during the Inception phase.
	Modeling Strategy	**Informal modeling sessions**	In Inception the Product Ownership team will meet together with other stakeholders to obtain a common understanding of the initial scope.
	Work Item Management Strategy	**Work item list**	A Program Backlog of features will be will created as well as separate work item lists for each team as features are decomposed and allocated to project teams.
	Non-Functional Requirements	**Explicit List**	Architecture and Product Owners will meet to agree upon system-wide non-functional requirements for all delivered features.
Align with Enterprise Direction	Adopt Common Guidelines	**High level, enforced**	During Inception the Architecture Owners outline common guidelines and agree that certain guidelines are not optional. Additionally a User Experience (UX) expert will collaborate with all the teams to ensure consistency and optimize the user experience.
	Coordinate with Enterprise Teams	**Collaborative, formal**	The Enterprise Architecture (EA) and Product Management teams agree to formalize regular coordination meetings to ensure consistency and manage dependencies across teams.
Identify Initial Technical Strategy	Level of Detail	**High-level overview**	The Architecture Owners model the architecture at a high level.
	View Types	**Technology, Business Architecture, User Interface (UI)**	Technology diagrams will be created using a few UML diagrams, business architecture will be modeled with free form diagrams, and UI prototypes will be hand-drawn.
	Modeling Strategy	**Informal Modeling sessions, Single candidate architecture**	All models will be created and evolved in informal modeling sessions. An existing reference architecture will be used.
	Delivery Strategy	**Extend existing solutions**	While most of the application will be built from scratch, some services will be used from existing systems.
Move Closer to Deployable Release	Deployment Strategy	**External releases as appropriate, Internal release each iteration**	Each team will create an internal release for each of four iterations per release. Iterations are two weeks in length. After four iterations, the work will be integrated and released externally to stakeholders via a Transition phase.
	Asset Management	**Configuration management**	The team will share a GIT repository for all project artifacts.
	Documentation Strategy	**Active stakeholder participation, Continuous documentation – same iteration**	The product owner will document user documentation in the iteration in which the features are delivered. The architecture owner will update the architectural handbook in the same fashion.
	Validation	**Continuous integration, Test-after development**	The teams are not yet ready to adopt test-driven development at either the acceptance or developer level. Test-after programming will be the initial approach. Teams will automate their build processes and regression test execution.
	Verification	**Non-solo development, Informal reviews**	Teams will selectively do non-solo development specifically for modeling sessions and some coding. Informal code reviews will be done as needed.

CONTINUED ON NEXT PAGE...

Table 2. Example of a DAD approach to scaling for a large development team.

DAD Goal	Factor	Process Decision	Strategy
Coordinate Activities	Share Information	**Conversations, Informal and Formal Reviews**	Information sharing to be primarily via direct conversations.
	Artifact Ownership	**Collective ownership**	All artifacts will be visible across teams. Team members can work on all artifacts within their own teams.
	Coordinate Within Programme	**Coordination meetings, Visualize work. Common cadences**	Architecture and Product Owners will have regularly scheduled informal meetings to share information about work details, priorities, dependencies and issues. All teams will visualize their work on virtual planning and work boards viewable by all stakeholders. All teams will deliver work in two-week iterations, with common start and end dates.
	Coordinate Across IT	**Enterprise professional as team member, Documented enterprise strategy (light)**	Each architecture owner is a member of the enterprise architecture team. At least one team member from the database and quality assurance groups will be part of each team and will ensure that the enterprise strategies that have been documented are consistent with the teams' work.
	Coordinate Release Schedule	**Release Windows**	After every four iterations of Construction and one week of Transition an external release will be deployed to stakeholders for the combined work of all the teams. During Transition people will finalize documentation for the release, fix any problems found as the result of testing, plan and model ahead for the coming iteration, work with support staff to learn about new features being deployed, and take the opportunity to take training or vacation.
	Coordinate Between Locations	**Gather physically at critical times**	At the beginning of each release all teams will gather at a common location for two days of release planning.

Table 2 Continued. Example of a DAD approach to scaling for a large development team.

there is a development team made up of a Team Lead (who is also the Architecture Owner), a "proxy PO", and seven developers. The proxy PO acts as the PO for the Pune team and liaisons with the actual customer PO in Toronto on a daily basis.

Early in the project they flew the proxy PO, the team lead, and two developers from Pune into Toronto for a two week modeling and planning workshop. This workshop was performed in a large dedicated modeling room. When it came to exploring the initial scope, the requirements for the application were captured via a collection of epics and user stories, a few high-fidelity screen mock-ups (to help set user interface conventions), and a user-interface flow diagram depicting the main screens and the navigation between them. These requirements were captured using a combination of Microsoft Word and Microsoft Visio and stored in SharePoint. Use of these tools to facilitate communication across the distributed teams was part of their strategy to adopt collaborative tools. The team decided to adopt a work item list approach for managing changing stakeholder needs and the stories were initially prioritized by the PO with advice from the Architecture Owner about technical risk considerations. The fifteen highest priority stories had acceptance criteria defined for them in detail, and most stories had at least a few criteria identified so that the proxy PO had a sufficient understanding of what each story entailed.

To identify the initial technical strategy the architecture owner and two developers work closely with the technical stakeholders such as enterprise architects and operations staff. They need to learn about the existing infrastructure, to meet key contacts within the customer's IT department whom they'll work with during Construction to obtain information and help, and to formulate a viable technical strategy. Most of the actual modeling is performed informally on whiteboards and key diagrams are captured towards the end of the two weeks using Visio. These views included a free-form layered architecture diagram, a high-level conceptual domain model, and several UML sequence diagrams depicting design patterns for common interactions between the mobile application and backend systems. The mobile application itself will be built from scratch by the Pune team and it will leverage existing back-end services already in place within the customer's IT infrastructure.

The team addressed the Move Closer to a Deployable Release process goal in a straightforward manner. The Pune team is taking a developer-level test-driven development (TDD) approach to programming. The continuous integration (CI) strategy includes code analysis tools which provide metrics that are monitored by the team lead at the customer organization. They pair on an as needed basis and hold code reviews within the team weekly. The application is being released internally at the end of each one-week iteration, and into production every three to four months as appropriate. Deployment planning is discussed weekly between the team leads. The internally released application is deployed into a testing environment where the Toronto-based independent testers validate that the latest build works within their IT environment and is usable. Creation of supporting deliverable documentation lags two days behind development and is written by the Pune team. This documentation is provided to the independent testers immediately upon its availability so that it can be validated along with the software previously delivered.

The Coordinate Activities goal is also addressed in a straightforward manner. The use of a proxy PO is related to a Coordinate Between Locations strategy to use boundary spanners. When it comes to coordination within the teams each team holds its own daily coordination meeting and both have visual information radiators in the form of whiteboards where their key diagrams are maintained. There is also a project dashboard, the teams are working with Microsoft Team Foundation Server (TFS), which displays quality and progress metrics automatically generated by their development tools. Both teams use TFS to manage their work backlog. To coordinate across teams the Toronto-based PO has a sixty minute teleconference every day with the proxy PO first thing in the morning to answer any requirement-related questions. The team members regularly communicate with one another via an internal discussion board on SharePoint, often to discuss potential defects found by the independent testers in Toronto. Every 4 months the PO and Team Leads meet in Toronto for a week to do long range release planning and general coordination between the locations (this is in effect a mini-Inception phase).

From a Table to Prose

For Scenarios B and C, rather than showing the explicit decisions made for Scenario A using a table we have instead chosen to take a prose approach to describing the scenarios in order to demonstrate how the goal-driven strategies are interwoven with the challenging realities found in scaling situations.

Scenario C: Medium-Sized Team, Domain Complexity and Regulatory

This is a software development team of fifteen people working out of a single location. Each person has their own cubicle and there is also a permanently reserved team room where they've installed whiteboards and a corkboard. Some people work from home one or two days a week as per the companies flexible work policy. The team works for a medical device manufacturer developing device management systems. There are three other hardware development teams working in parallel to the software team. Together they are working on a family of related medical devices, each of which is on a different release schedule. All development is subject to Federal Drug Administration (FDA) regulations.

To explore the initial scope the team creates light weight use cases and a detailed list of non-functional requirements. They also create low fidelity screen mockups for the seven primary screens and whiteboard sketches of the overall business process. The sketches are first captured using a digital camera and then transcribed into more formal drawings using Enterprise Architect from Sparx Systems. The sketches remain on the whiteboards in the team room and evolved as needed throughout the Construction phase. Requirements are elicited via a combination of informal modeling sessions in their team room and via interviews, all of which are led by the Product Owner. The requirement specification, a combination of the previously mentioned artifacts, is captured using a Wiki. This documentation would be about 35 pages if the team printed it out. The team also chose a lean work item pool strategy to manage work items using a JIRA virtual Kanban board.

To identify the initial technical strategy the team takes an informal approach to modeling, as they did with their scoping efforts. Initial requirements and architecture modeling takes about one month for the first release of the application because the software team works closely with the three hardware development teams to understand their needs. After considering the possibility of modifying an existing system built a few years ago, the team decides it would be better to build the new solution from scratch. The architecture views include a free-form logical diagram of the various software modules and a physical free-form diagram for the software supporting each of the three devices being supported. Key architectural decisions and the reasons behind them are captured using a Wiki and diagrams are captured using Enterprise Architect.

The Move Closer to a Deployable Release process goal implementation was greatly affected by the need to work with the hardware teams and to do so in a regulatory compliant manner. The software development team needed to adopt a deployment cadence that reflected the deployment schedules of the devices. The hardware development teams were running on twelve to eighteen month release cycles depending on the device, although these releases were staggered so that a new product was released every six months or so. All teams had adopted a strict configuration management (CM) strategy to asset management which reflected regulatory requirements. The team also adopted acceptance TDD and developer TDD approaches, in part driven by the regulatory requirements for quality and in part because it had the side effect of covering most of the traceability requirements in the regulations. The whole team testing efforts of the development team were enhanced by support from the organizations test team, which tested accepted working builds from the team every two to three weeks. This independent test effort was also driven by regulatory requirements. Regulatory compliance also motivated ongoing code reviews and documentation of the results of each review. As their diagrams evolved on the whiteboards once a week the team's technical writer would update the corresponding electronic versions accordingly. The technical writer also paired with developers on a rotating basis to help them write the supporting documentation for the software (also in conformance to regulations).

Coordinating activities is fairly straightforward. The software team meets for ten minutes on a daily basis around their JIRA Kanban board which is displayed on a large touch screen monitor in their team room. Anyone working from home that day calls in and views the shared board over a Webex session. Everyone is expected to have JIRA updated with their latest information before the coordination meeting, enabling the team to focus their discussions on anything blocking them. The team's Architecture Owner meets with the architects on the hardware teams on a weekly basis, and as needed throughout the week, to coordinate technical issues. Similarly the team leads meet once a week to discuss any issues between the teams as well as the release schedule. The PO meets with her counterparts on the device teams on a daily basis to coordinate requirements-related issues.

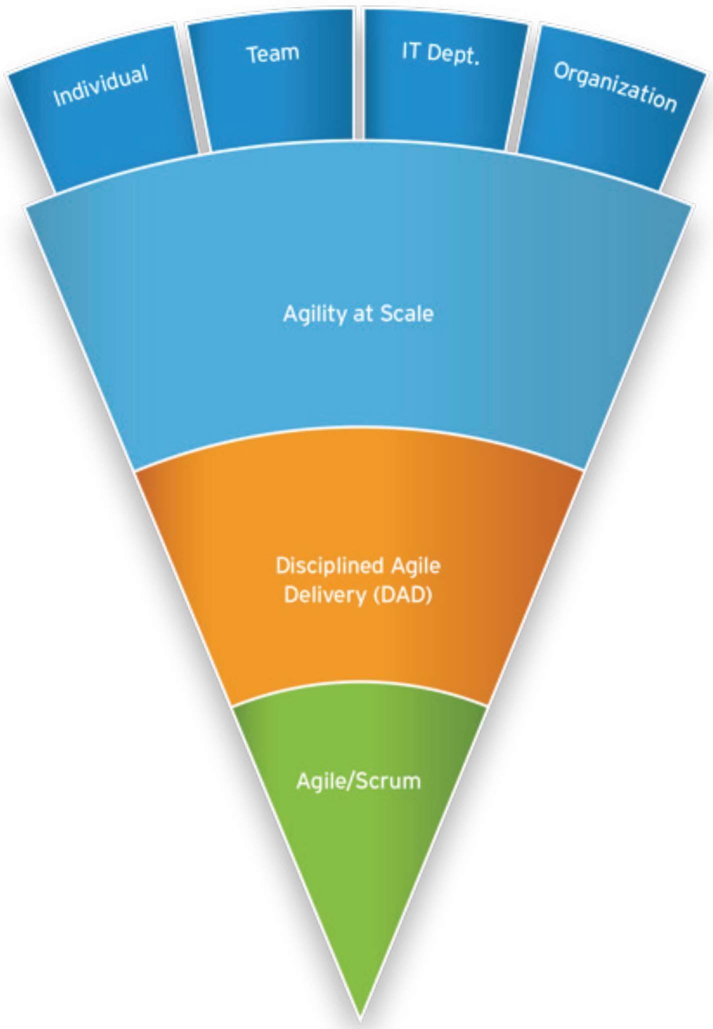

Figure 9. Scaling agile delivery across your entire organization.

Parting Thoughts

We began by describing how there are two lines of thought when it comes to agility at scale: First, how to scale agile delivery, the focus of this paper; and second how to scale agile strategies across the organization. Scaling across your organization requires you to help individuals, teams, and departments to adopt an agile mindset and agile ways of working together. *Figure 9* overviews both of these visions. Our experience is that you need to first scale agile delivery before you can think about scaling agile across the organization. Only then can your organization truly operate as an Agile Enterprise, the topic of a future paper.

The ultimate goal is enterprise agility, where the mainstream business processes have adopted agile ways of working and where IT use is optimized. You need to evolve your organization's view of IT from "run the business" to innovation, and from implementing tactical strategies to long-term strategies that affect the future of the business. Organizations that adopt agile and lean practices in the IT department, analytics and a learning mindset in the mainstream business, and Lean Startup (and related processes) as a driver for innovation can transform themselves into agile enterprises. Such an organization has reduced risk by relying on rapid evaluation of evidence to drive business strategy, and uses feedback loops in the IT department and the mainstream business to guide activities and deliver what the customer wants. The defining characteristics of an agile enterprise are reflected in its adherence to the principles and values of agile and lean thinking.

References and Recommended Resources

1. Ambler, S.W. (2012). *2012 DDJ State of the IT Union Survey Results*. http://www.ambysoft.com/surveys/stateOfITUnion201209.html
2. SA+A (2013). *2013 IT Project Success Rates Survey Results*. http://www.ambysoft.com/surveys/success2013.html
3. Beck, K. et. al. (2001). *Manifesto for Agile Software Development*. http://agilemanifesto.org/
4. West, D. (2011). *Water-Scrum-Fall is the Reality of Agile for Most Organizations Today*. http://www.cohaa.org/content/sites/default/files/water-scrum-fall_0.pdf
5. Ambler, S.W. & Lines, M. (2013). *Going Beyond Scrum: Disciplined Agile Delivery*. http://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf
6. Ambler, S.W. and Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Development in the Enterprise*. IBM Press.
7. Ambler, S.W. and Lines, M. (2012). *Full Agile Delivery Lifecycles*. http://disciplinedagiledelivery.com/lifecycle/
8. Ambler, S.W. (2013). *The Software Development Context Framework*. http://disciplinedagiledelivery.com/2013/03/15/sdcf/
9. Kruchten, P. (2013). *Contextualizing agile software development*. Journal of Software: Evolution and Process 25(4): 351-361.
10. Ambler, S.W. and Lines, M. (2013). *Exploring Initial Scope on Disciplined Agile Teams*. http://disciplinedagiledelivery.com/2013/07/17/exploring-initial-scope-on-disciplined-agile-teams/
11. Ambler, S.W. and Lines, M. (2014). *Identify Your Initial Technical Strategy*. http://disciplinedagiledelivery.com/2014/02/10/identifying-your-initial-technical-strategy/
12. Ambler, S.W. and Lines, M. (2014). *Move Closer to a Deployable Release*. http://disciplinedagiledelivery.com/2014/02/22/move-closer-to-a-deployable-release/
13. Ambler, S.W. and Lines, M. (2013). *Coordinating Activities on Agile Delivery Teams*. http://disciplinedagiledelivery.com/2013/07/12/coordinating-actitivies/
14. *Scaled Agile Framework (SAFe) home page*. http://scaledagileframework.com/

Acknowledgements

We would like to acknowledge the help of the following people: Beverley Ambler, Mike Bowler, David Dame, Carson Holmes, Julian Holmes, Glen D. Little, Jas Madhur,

About the Authors

Scott Ambler is a Senior Consulting Partner of *Scott Ambler + Associates*, working with organizations around the world to help them to improve their software processes. He provides training, coaching, and mentoring in disciplined agile and lean strategies at both the project and organizational level. Scott is the founder of the Agile Modeling (AM), Agile Data (AD), Disciplined Agile Delivery (DAD), and Enterprise Unified Process (EUP) methodologies. He is the (co-)author of several books, including Disciplined Agile Delivery, Refactoring Databases, Agile Modeling, Agile Database Techniques, The Object Primer 3rd Edition, and The Enterprise Unified Process. Scott is a senior contributing editor with Dr. Dobb's Journal and his company's home page is ScottAmbler.com.

Mark Lines is Managing Partner at *Scott Ambler + Associates*. He is an Agile Coach and co-creator of the Disciplined Agile Delivery framework. Mark is co-author with Scott Ambler of Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. Mark is a "disciplined" agile coach, helping organizations all over the world transform from traditional to agile methods. He writes for many publications including the Cutter Consortium and is a frequent speaker at industry conferences. Mark blogs about DAD at DisciplinedAgileDelivery.com. He is also a Founding Member of the Disciplined Agile Consortium (DAC), the certification body for disciplined agile.

About The Disciplined Agile Consortium

The Disciplined Agile Consortium (DAC), http://DisciplinedAgileConsortium.org, is the home of the Disciplined Agile Delivery (DAD) process decision framework. The mission of the DAC is to help organizations and individuals around the world understand and adopt disciplined agile ways of working. We share these strategies via white papers, workshops, conference presentations, and through certification of individual practitioners.

The disciplined agile certification program is based on the following principles:

- Certifications must provide value
- Certifications must be earned
- Certifications must be respectable
- Certifications must be focused
- Certification is part of your learning process
- Certified professionals have a responsibility to share knowledge

There are three practitioner certifications:

1. **Disciplined Agilist Yellow Belt.** This beginner certification indicates to colleagues and employers that you are eager to learn disciplined agile strategies that enable you to increase your skills and abilities as a software professional.
2. **Disciplined Agilist Green Belt.** This intermediate certification indicates that you are experienced at DAD and are on your way to becoming a generalizing specialist. You have the potential to be a "junior coach" under the guidance of a senior coach (someone who is likely a Disciplined Agile Black Belt).
3. **Disciplined Agilist Black Belt.** This expert certification indicates that you are a trusted expert with significant proficiency at DAD. You can coach other people in disciplined agile strategies and advise organizations in the adoption and tailoring of the DAD framework.

DISCIPLINEDAGILE
CONSORTIUM