DISCIPLINED AGILE CONSORTIUM

White Paper Series

October 2013

Going Beyond Scrum

Disciplined Agile Delivery

By Scott W. Ambler

This paper describes, step-by-step, how to evolve from today's Scrum vision of agile software development to a disciplined agile solution delivery. It begins with a brief overview of the agile software development movement and its implications. We then overview the Scrum method with its associated benefits and drawbacks, and then how to move beyond Scrum to a full delivery process framework called Disciplined Agile Delivery (DAD). DAD is a governed, hybrid approach that provides a solid foundation from which to scale agile solution delivery within enterprise-class organizations. The steps to do this are:

1. *Focus on consumable solutions, not just potentially shippable software*
2. *Extend Scrum's construction lifecycle to address the full delivery lifecycle*
3. *Move beyond method branding*
4. *Adopt explicit governance strategies*
5. *Take a goal-based approach to enable tailoring and scaling*

Agile Software Development

In 2001 a group of experienced software professionals gathered at the Snowbird ski resort in Utah to explore how to effectively develop software. The Agile Manifesto [1] was the result of that meeting, a philosophical treatise described in terms of four value statements supported by twelve principles. The values of the manifesto are written in terms of X over Y, with the observation that while both X and Y are important to successful software development the manifesto author's experience that X was by far the more important of the two. For example, the first value is "Individuals and interactions over processes and tools". What the authors are saying is that although the processes followed by a team and the tools that they use are important, the people and the way that they collaborate are greater determinants of the success of your software development team. Throughout this paper we use the same format to capture critical suggestions pertaining to the successful application of agile strategies.

One concept that is clearly promoted by the Agile Manifesto, and one that since its publication we have seen a groundswell of support for, is a focus on the team. Alistair Cockburn captured this philosophy best when he claimed that "software development is a team sport" [2]. It requires teams to build software-based solutions, not just individuals, the implication being that to succeed at the modern software development game we must find ways that enable teams of people to work together effectively.

The manifesto paved the way for mainstream adoption of existing lighter-weight methods such as Scrum and Extreme Programming (XP) and laid the foundation for new methods such as Agile Modeling (AM), Outside In Development (OID) and many others. Each of these methods has its strengths and weaknesses, focusing on some aspects of the software delivery process but downplaying or even missing others. When someone claims to be working on a team following the X method, a quick inspection reveals that they're following X with strategies adopted from Y, Z, and other sources.

At the time of this writing Scrum is by far the most popular of the agile methods, but it is far from complete for any software development team. Figure 1 depicts the Scrum lifecycle, the focus of which is construction. The Scrum method focuses on change management – requirements are managed in the form of a prioritized stack called a product backlog that is allowed to evolve over time as your customer's understanding of their needs evolves – and on leadership. It purposely doesn't address technical practices and is explicit about the need to look to other sources for such. It promotes the idea that software should be delivered incrementally in short time boxes called sprints, and that each software increment should be "potentially shippable" in that customers have the option to have the software deployed into production at the end of the sprint if they deem the software sufficient for their needs. Scrum teams are self-

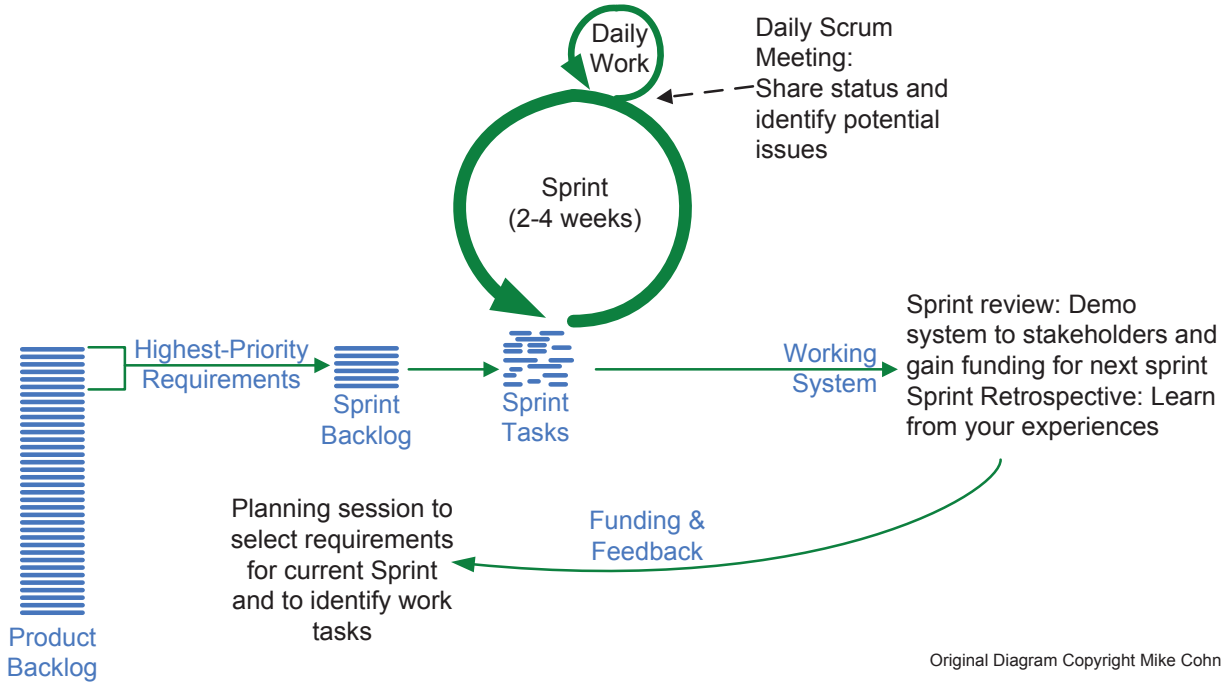

Figure 1. The Scrum Construction Lifecycle.

organizing and embrace the idea that requirements will evolve over time, enabling them to respond to change easily. Scrum has helped to popularize the strategy that it's better to respond to change, and thereby build software your customer actually wants, instead of following a plan and thereby build software that fulfills a specification that is no longer relevant.

Disciplined Agile Delivery

Many organizations start their agile journey by adopting Scrum because it describes a good strategy for leading agile software teams. However, Scrum is only part of what is required to deliver sophisticated solutions to your stakeholders. Invariably teams need to look to other methods to fill in the process gaps that Scrum purposely ignores. When looking at other methods there is considerable overlap and conflicting terminology that can be confusing to practitioners as well as outside stakeholders. Worse yet people don't always know where to look for advice or even know what issues they need to consider.

To address these challenges the Disciplined Agile Delivery (DAD) process decision framework provides a more cohesive approach to agile solution delivery [3]. To be more exact, here's a definition: "The Disciplined Agile Delivery (DAD) decision process 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."

Let's explore some of the key aspects of the DAD framework. DAD is a hybrid approach which extends Scrum with proven strategies from Agile Modeling (AM), Extreme Programming (XP), Unified Process (UP), Kanban, Lean Software Development, Outside In Development (OID) and several other methods. Although DAD was originally developed by IBM, it is a non-proprietary, freely available framework that does not require IBM tooling in any way. DAD extends the construction-focused lifecycle of Scrum to address the full, end-to-end delivery lifecycle from project initiation all the way to delivering the solution to its end users. It also supports lean and continuous delivery versions of the lifecycle – unlike other agile methods, DAD doesn't prescribe a single lifecycle because it recognizes that one strategy does not fit all. DAD includes advice about the 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, instead of the prescriptive approach seen in other agile methods, including Scrum, the DAD framework takes a goals-driven approach. In doing so DAD provides contextual advice regarding viable alternatives and their trade-offs, enabling you to tailor DAD to effectively address your particular situation. By describing what works, what doesn't work, and more importantly why, DAD helps you adopt strategies that are right for you.

One of DAD's philosophies is that it focuses on the delivery of consumable solutions, not just potentially shippable software. In addition to software we create supporting documentation. The software runs on hardware that may need to be upgraded and/or redeployed. We potentially change the business process around the usage of the system we're producing. We may even affect changes to the organization structure of the people using the system. This means that we're not just producing "potentially shippable software" but instead are producing "potentially shippable solutions" that solve a larger business need. Moreover, producing something that is just "potentially shippable" isn't what our stakeholders actually want. What they really desire is something that's consumable, something that they can easily understand, adopt, support, and evolve to help them achieve their goals.

An important strength of agile is its focus on working closely with the customer of the software so to increase the chance that what you produce value for them. Non-agile teams may choose to rely on written, detailed requirement specifications – specifications which often prove to be inconsistent and just plain incorrect – which the team then builds to. These specifications, in addition to detailed schedules, architecture definitions, and budgets, often form a contract between the software team and their customers. Agilists prefer the less risky strategy of working collaboratively with our stakeholders all the way through the lifecycle, showing them our work as we go and acting on the feedback that we receive. Notice how we switched from the term customer to stakeholder. In DAD we prefer to talk in terms of stakeholders. The original term "customer" is more associated with end-users and buyers, which for some software teams was an important shift of emphasis away from technology to business needs. The relentless focus on customers is inadequate for enterprise environments which tend instead to operate on client chains, with "customers" sometimes not directly accessible. Disciplined agilists also consider other stakeholders such as operations staff, support staff, enterprise architects, internal auditors, and many more.

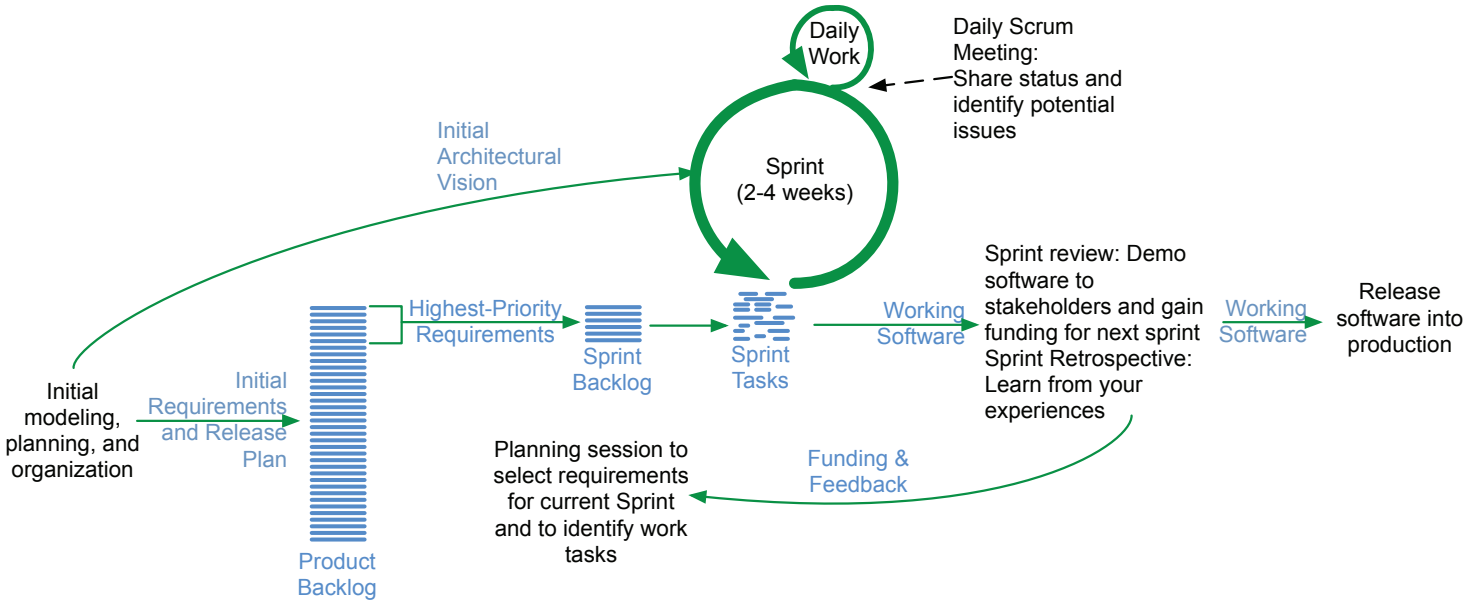

Figure 2. A delivery version of the Scrum lifecycle.

Full Delivery Lifecycle

Disciplined agile teams recognize that there is some up-front project initiation/inception work that occurs early in a project and similarly some deployment/transition effort that occurs towards the end of a project. The end result is that DAD promotes the idea that you need to adopt a full delivery lifecycle, not just a construction-focused lifecycle, an important differentiator over Scrum. In fact the July 2013 Agile State of the Art survey found that agile teams reported spending an average of one month project initiation work [4]. Similarly, the November 2010 Agile State of the Art survey found that agile teams spent an average of one month on transition efforts [5]. We've found that without explicit guidance many agile teams suffer from common mistakes such as following a Water-Scrum-Fall approach where a traditional, overly heavy project initiation phase occurs, followed by a Scrum construction phase, and ending with an overly heavy traditional deployment phase [6]. These heavy project initiation and delivery phases increase the time it takes to deliver, the cost to deliver, and reduce the chance that the team will produce something their stakeholders actually desire. It is possible, and very desirable, to keep both project initiation and solution deployment efforts as light and streamlined as your situation warrants and DAD

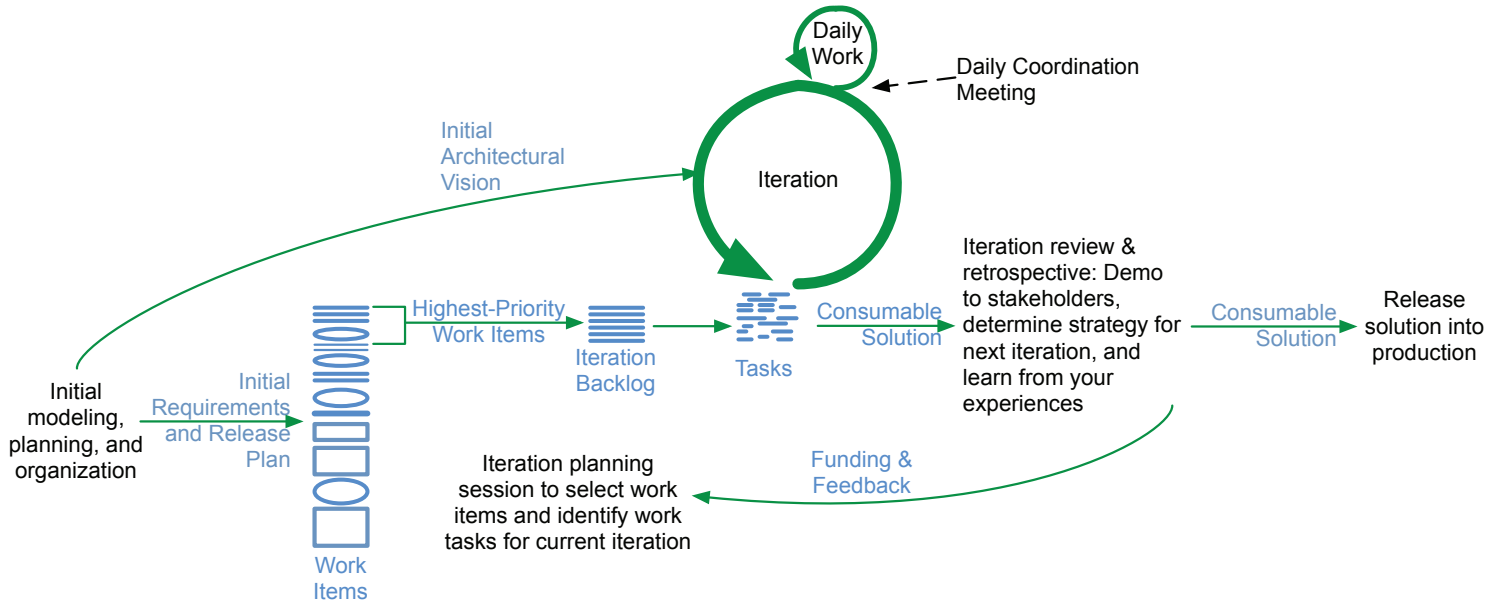

Figure 3. A non-branded delivery lifecycle.

provides guidance to do exactly that.

Let's consider what the Scrum construction lifecycle of Figure 1 would look like if it were to explicitly address a full delivery lifecycle. Figure 2 depicts what we refer to as a "Scrum Delivery Lifecycle". This lifecycle explicitly depicts project initiation activities such as initial requirements elicitation, sometimes referred to as backlog population, initial architecture modeling, and initial release planning. During this period you may be performing other activities such as forming the team, setting up your work environment, and securing funding for the rest of the project. This lifecycle explicitly indicates that the software will need to be released into production, or in the case of a commercial product into the marketplace, at some point. Our experience is that starting with a full delivery lifecycle such as this helps teams to avoid slipping into a Water-Scrum-Fall approach and the disadvantages associated with it.

Although many Scrum terms sound silly – you don't "sprint" through an entire project and what the heck is a "Scrum Master" – this change in terminology helped people to think outside of the traditional system development lifecycle (SDLC) box. We're seeing the same problem now with existing Scrum teams that are struggling to think outside of the Scrum box, and we've found that a simple change in terminology (e.g. iteration instead of sprint, team lead instead of Scrum Master) provides a clear signal to people that we're making another process improvement leap. Interestingly, the "New Deal" for Software Development [7] advocates that we move away from the method branding we've seen in recent years to the adoption of clear and consistent terminology. DAD reflects this mindset. So, instead of rebranding Scrum meetings to Disciplined Agile Meetings we instead adopted the term "coordination meeting" which clearly represents what is going on. Similarly DAD prefers terms such as retrospective instead of sprint retrospective, demo instead of sprint demo, and so on. Strategies such as coordination meetings, demos, and retrospectives were generally accepted practices long before Scrum came along. In many enterprises the business appreciates simpler, more familiar terminology, and this is asgood enough a reason as any to maintain the known terminology. Figure 3 depicts a non-branded version of Figure 2.

One improvement depicted in Figure 3 is how it depicts the prioritized stack of work. We use the term work items instead of product backlog and we depict the list differently. We graphically indicate that there are different types of things that teams work on, not just new features. For example, disciplined agile team members will work on new functionality, fixing defects,

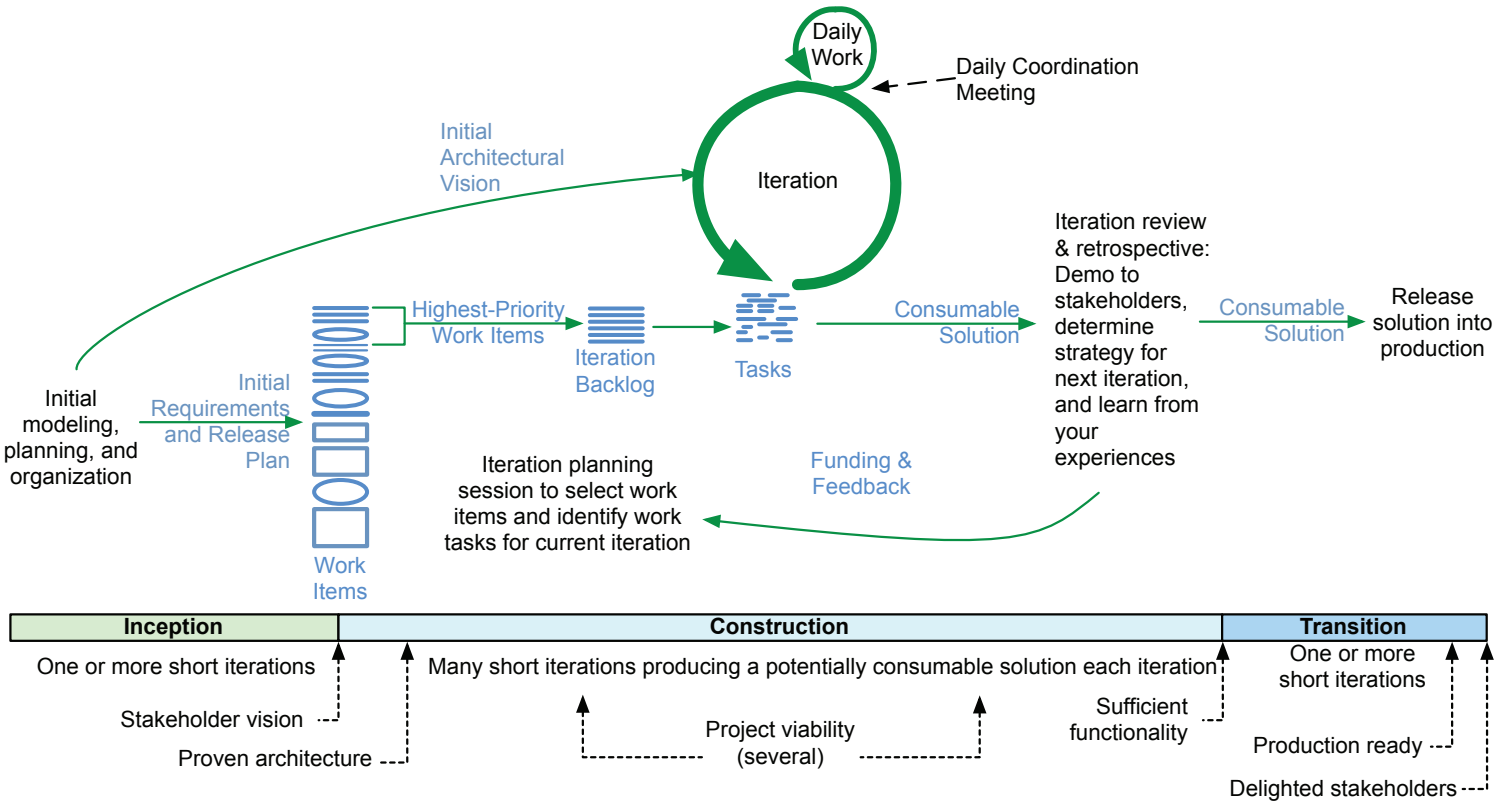

Figure 4. A governed agile delivery lifecycle.

helping other teams (perhaps by reviewing their work or mentoring them), large refactoring efforts and other technical work, R&D spikes, regulatory documentation submissions, and so on. All of these work items should be prioritized and scheduled accordingly. Also, work items are often different sizes. It is common practice to explore the details of high priority work items, and to disaggregate them into small chunks, and to not invest much time exploring or refactoring low-priority work items as you may never get around to actually working on them if the details or priorities evolve.

We can still improve the lifecycle for enterprise-class environments. A differentiator of the DAD framework is that it explicitly recognizes that agile teams, just like other types of teams, are governed. Governance establishes chains of responsibility, authority, communication, and funding in support of the overall enterprise's goals and strategy. It also establishes measurements, policies, standards and control mechanisms to enable people to carry out their roles and responsibilities effectively. You do this by balancing risk versus return on investment (ROI), setting in place effective processes and practices, defining the direction and goals for the department, and defining the roles that people play with and within the department.

Figure 4 visually adds two important aspects of governance, named phases and light-weight milestones. The named phases – Inception, Construction, and Transition – help to identify the nature of the work that a team should be focused on at the time, and provides context for how to organize that work. More importantly the lifecycle now calls out several important milestones that guide disciplined agile teams to lower overall project risk. Several of these milestones, in particular Project Viability and Sufficient Functionality (what lean practitioners refer to as a minimum viable product or minimum marketable release) are built into Scrum but are not explicit on the lifecycle diagram. While the phase names were adopted directly from the Unified Process (UP) and several of the milestones as well, we renamed the UP milestones to something more descriptive. For example DAD's Stakeholder Vision corresponds to UP's Lifecycle Objectives milestone.

There has been much ado made over the strategy of self-organizing teams within the agile community, and rightfully so as it is an effective strategy. However, the reality is that agile teams generally don't have the freedom to do anything that they want and to do this work in any way that they want. Instead they must work within the scope and constraints of a larger, organizational ecosystem. The DAD framework recognizes this and instead promotes the idea that disciplined agile teams should not work in an isolated manner but instead should be self-organizing with appropriate governance to guide them to greater levels of success.

Goals for the Inception Phase	Goals for Construction Phase Iterations	Goals for the Transition Phase
- Form initial team - Develop common project vision - Align with enterprise direction - Explore initial scope - Identify initial technical strategy - Develop initial release plan - Form work environment - Secure funding - Identify risks	- Produce a potentially consumable solution - Address changing stakeholder needs - Move closer to deployable release - Improve quality - Prove architecture early	- Ensure the solution is consumable - Deploy the solution
Ongoing Goals		
- Fulfill the project mission - Grow team members - Address risk	- Improve team process and environment - Leverage and enhance existing infrastructure - Coordinate activities	

Figure 5. The Goals of Disciplined Agile Delivery.

Disciplined Agile Teams Are Process Goal Driven

One process size does not fit all. Starting in the 1970s, a common assumption was that an "industrialized" approach where specialists focused on their own portion of the work and then passed it on to the next specialist(s) was an effective way to organize IT work. This lead to the waterfall or "V" model where software development teams were formed from specialists with roles such as solution architect, business analyst, programmer, tester, project manager, and many others handed batches of work back and forth to one another. They did this following a common process, often described in intricate detail and supported by documentation templates, in the belief that such prescriptive bureaucracy could lead to predictability. What it led to was expensive solutions that were often late to market, costly, and didn't meet the actual needs of their stakeholders. Sadly, many organizations today still suffer from this mindset and struggle to adopt modern agile strategies as a result.

Even today with agile software development it's comfortable to think that prescriptive strategies such as managing changing requirements in the form of a product backlog, holding a daily meeting where everyone answers three questions, having a single requirements owner, and other ideas will get the job done. But we all know that this isn't true in all situations. There are many strategies for managing requirements change, there are different ways to coordinate within a team, there are different ways to explore stakeholder needs, and so on. Each of these strategies has advantages and disadvantages and each has a range of situations where they are appropriate. A strategy that works for a small co-located team will put a large geographically distributed team at risk. A strategy that works well in a non-regulatory environment may result in people's deaths in a regulatory one (or more likely fines because hopefully you'll be caught before you ship). So, if you want to build an effective team you need to be able to select the right strategy for the situation you find yourself in. DAD describes a straightforward, easy to consume process strategy that is goal-driven. This strategy has a visual component, process goal diagrams which summarize the fundamental process decision points, and a textual component, goals tables which capture and describe the options and their tradeoffs.

This goal-driven approach enables DAD to avoid being prescriptive and thereby be more flexible and easier to scale than other agile methods. For example, where Scrum prescribes a value-driven Product Backlog approach to managing requirements, DAD instead says

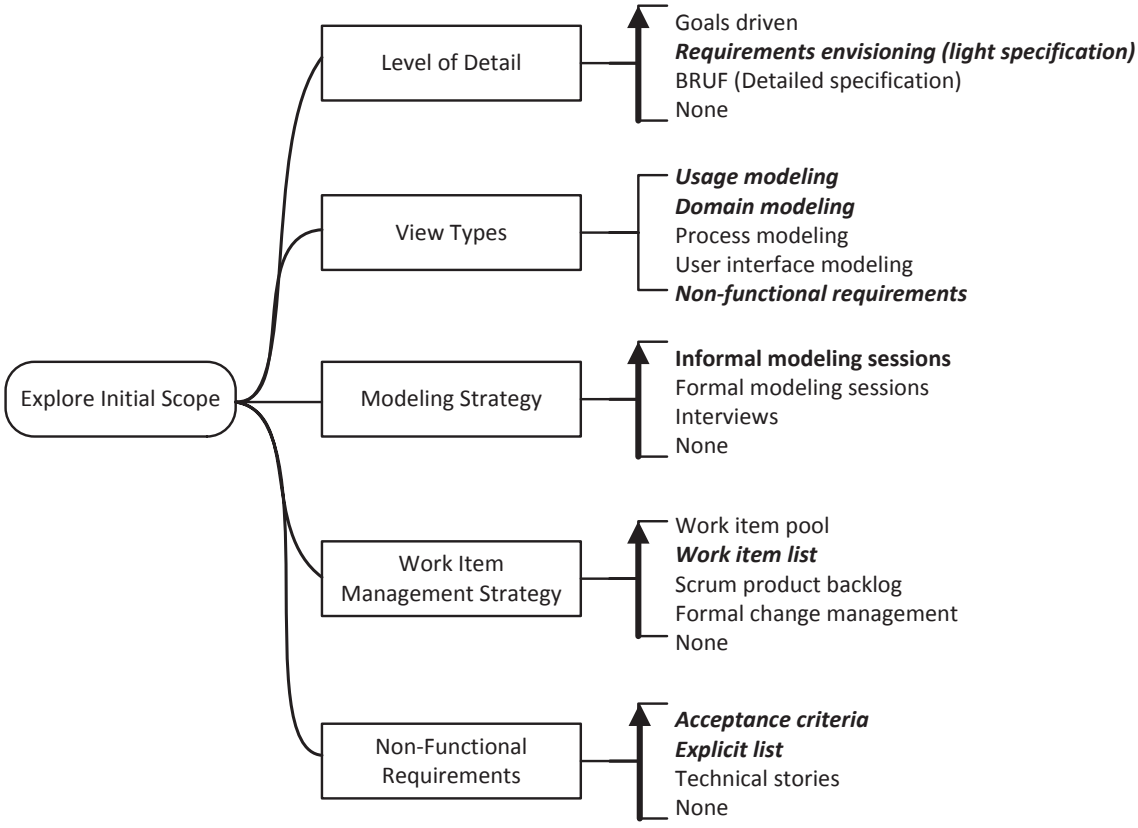

Figure 6. Process goal diagram: Explore Initial Scope.

that during construction you have the goal of addressing changing stakeholder needs. DAD also indicates that there are several issues pertaining to that goal that you need to consider, and there are several techniques/practices that could potentially address each issue. 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 5 shows the goals that are consistent with any type of project regardless of type, whether it be custom development or implementing a package for instance.

Figure 6 shows an example of a process goal diagram for the Explore the Initial Scope goal. This goal is an important part of the Inception phase so that we can move towards obtaining stakeholder consensus that it makes sense to move into the Construction phase and begin building the solution. For each issue there are a number of choices. Some choices, such as work item stack, are bolded and italicized. This highlighting is meant to indicate good choices as a place to start for a typical DAD project. Some issues show an arrow beside the options which is an indication that the choices at the top are typically the most effective and the better alternatives to strive for. A typical project will make hundreds of process decisions and these diagrams can be used to ensure that the various options are considered. An example of a decision might be what view type might we use to depict scope? DAD recommends starting with a combination of usage modeling, domain modeling,

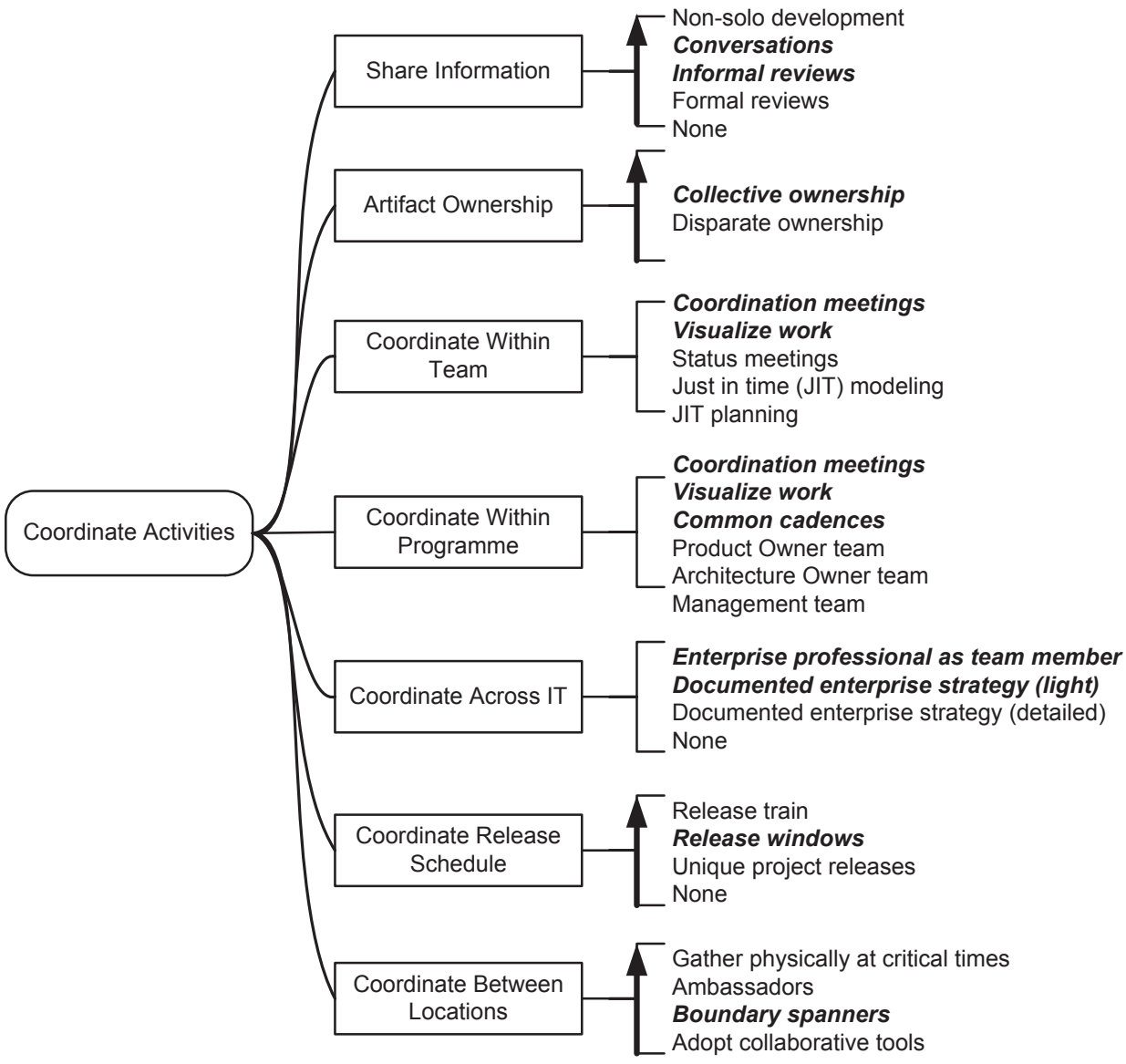

Figure 7. Process goal diagram: Coordinate Activities.

and non-functional requirements. For usage modeling, user stories are the most popular agile approach, but you could also create use case diagrams or personas as needed.

A second example of a process goal diagram, in this case for the ongoing goal Coordinate Activities, is shown in Figure 7. 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 organization 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 factors (discussed in detail later in this paper). For example, large teams (often called programmes) will find that they need to adopt strategies called out by Coordinate Within Programme. 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.

There are several fundamental advantages to taking a goal driven approach to agile solution delivery. First, it makes your process options very clear. Figure 5, in combination with the more detailed process goals diagrams (such as in Figure 7) nicely illustrate the range of agile practices available. Second, the diagrams support process tailoring by making the process decisions explicit. Third, scaling of agile delivery strategies is enabled by making the strengths and weaknesses of each practice clear (this advice is currently captured as textual tables in the DAD book). More on this later. Fourth, a goals-based approach makes it clear what risks you're taking on because it makes your process decision options and their tradeoffs explicit. Fifth, it takes the guesswork out of extending agile methods to address the context faced by a delivery team.

So far we've identified two potential challenges with DAD's goal-driven approach when working with customer organizations. First, it makes the complexities of solution delivery explicit. Although some of us want to believe that the simplistic strategies of other agile methods will get the job done we inherently know that software development, or more accurately solution delivery, is in fact a complex endeavor in practice. Second, some people just want to be told what to do and actually prefer a prescriptive approach. DAD mitigates this problem a bit by suggesting default starting points but even this can be overwhelming for some people. Interestingly, when we were writing the book two of our 30+ reviewers were adamantly against giving people choices because they felt it was better to adopt a more prescriptive approach as we see in older agile methods.

Disciplined Agile Teams are Enterprise Aware

Enterprise awareness is one of the key aspects of the DAD framework. The observation is that DAD teams work within your organization's organizational ecosystem, as do all other teams. There are often many existing systems currently in production and minimally your solution shouldn't impact them. Better yet, your solution will hopefully leverage existing functionality 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 which your team should contribute to. A governance strategy exists which hopefully enhances what your team is doing.

Enterprise awareness is important for several reasons. First, you can reduce overall delivery time and cost by leveraging existing assets or by creating new assets in alignment with an enterprise-level strategy that will be reused by upcoming projects. In other words, DAD teams can spend less time reinventing the wheel and more time producing real value for their stakeholders. Second, by working closely with enterprise professionals DAD teams can more easily get going in the right direction. Third, it increases the chance that your delivery team will help to optimize the organizational whole, and not just the "solution part" that it is tasked to work on. As the lean software development movement aptly shows, this increases team effectiveness by reducing time to market.

Figure 8 summarizes the four levels of awareness that an IT professional may exhibit:

- **Individual awareness.** From this viewpoint it's all about how someone can change themselves by gaining new skills, insights, experiences, and so on.

Figure 8. There are different levels of awareness.

- **Team awareness.** Here the focus is how the team can learn and improve together. This has been a primary philosophy of the agile community for quite some time, mostly to our benefit but sometimes to our detriment. Solutions are developed by teams, so by promoting a greater focus on the team, agilists are able to improve their overall productivity a bit. But, if the efforts of that team aren't well aligned with the overall goals of the organization then doing work that doesn't need to be done doesn't really help.
- **Enterprise awareness.** People are motivated to consider the overall needs of their organization, to ensure that what they're doing contributes positively to the goals of the organization and not just to the suboptimal goals of their team. This is an example of the lean principle of optimizing the whole, in this case the organization, over local optimization within just the team.
- **Community awareness.** People consider the needs of their community, doing what they can to give back by sharing knowledge, by striving to learn themselves, and by helping others who might not necessarily be in their organization or even known to them.

Enterprise awareness is an important aspect of self-discipline because as a professional you should strive to do what's right for your organization and not just what's right for yourself. Teams developing in isolation may choose to build something from scratch, or use different development tools, or create different data sources, when perfectly good ones that have been successfully installed, tested, configured, and fine-tuned already exist within the organization. Disciplined agile professionals will:

- **Work closely with enterprise professionals.** It takes discipline to work with enterprise professionals such as enterprise architects, data administrators, portfolio managers, or IT governance people who may not be completely agile yet, and have the patience to help them. It takes discipline to work with your operations

and support staff in a DevOps manner throughout the lifecycle, particularly when they may not be motivated to do so.

- **Adopt and follow enterprise guidance.** Your organization may have, or hopes to one day have, a range of standards and guidelines (guidance) that it wants delivery teams to adopt and follow. This may include guidance for coding, user interface development, security, and data conventions to name a few. Following common guidance increases the consistency and maintainability of your solutions, and thus your overall quality.
- **Leverage enterprise assets.** There may be many enterprise assets, such as reusable code, patterns, templates, and data sources that you can use and evolve.
- **Enhance your organizational ecosystem.** The solution being delivered by a DAD team should minimally fit into the existing organizational ecosystem – the business processes and systems supporting them – it should better yet enhance that ecosystem. Furthermore, experienced DAD teams will even fix problems that they run into via proven refactoring techniques, thereby reducing the costs of maintaining these assets and extending their useful lives.
- **Adopt a DevOps Culture.** DAD teams will work with operations and support staff closely throughout the lifecycle, particularly the closer you get to releasing into production. This collaboration reduces the risk of deployments and ensures a smooth transition to support groups. DevOps philosophies and strategies are baked right into DAD.
- **Share learnings.** DAD teams are learning oriented, and one way to learn is to hear about the experiences of others. The implication is that DAD teams must also be prepared to share their own learnings with other teams. To do this organizations might choose to support agile discussion forums, informal presentations, training sessions delivered by senior team members, and internal conferences to name a few strategies.
- **Adopt appropriate governance strategies.** Effective governance strategies should enhance that which is being governed. An appropriate approach to governing agile delivery projects, and we suspect other types of efforts, is based on motivating and then enabling people to do what is right for your organization. What is right will of course vary, but this typically includes motivating teams to take advantage of, and to evolve, existing corporate assets following common guidelines to increase consistency, and working towards a shared vision for your organization. Appropriate governance is based on trust and collaboration. Appropriate governance

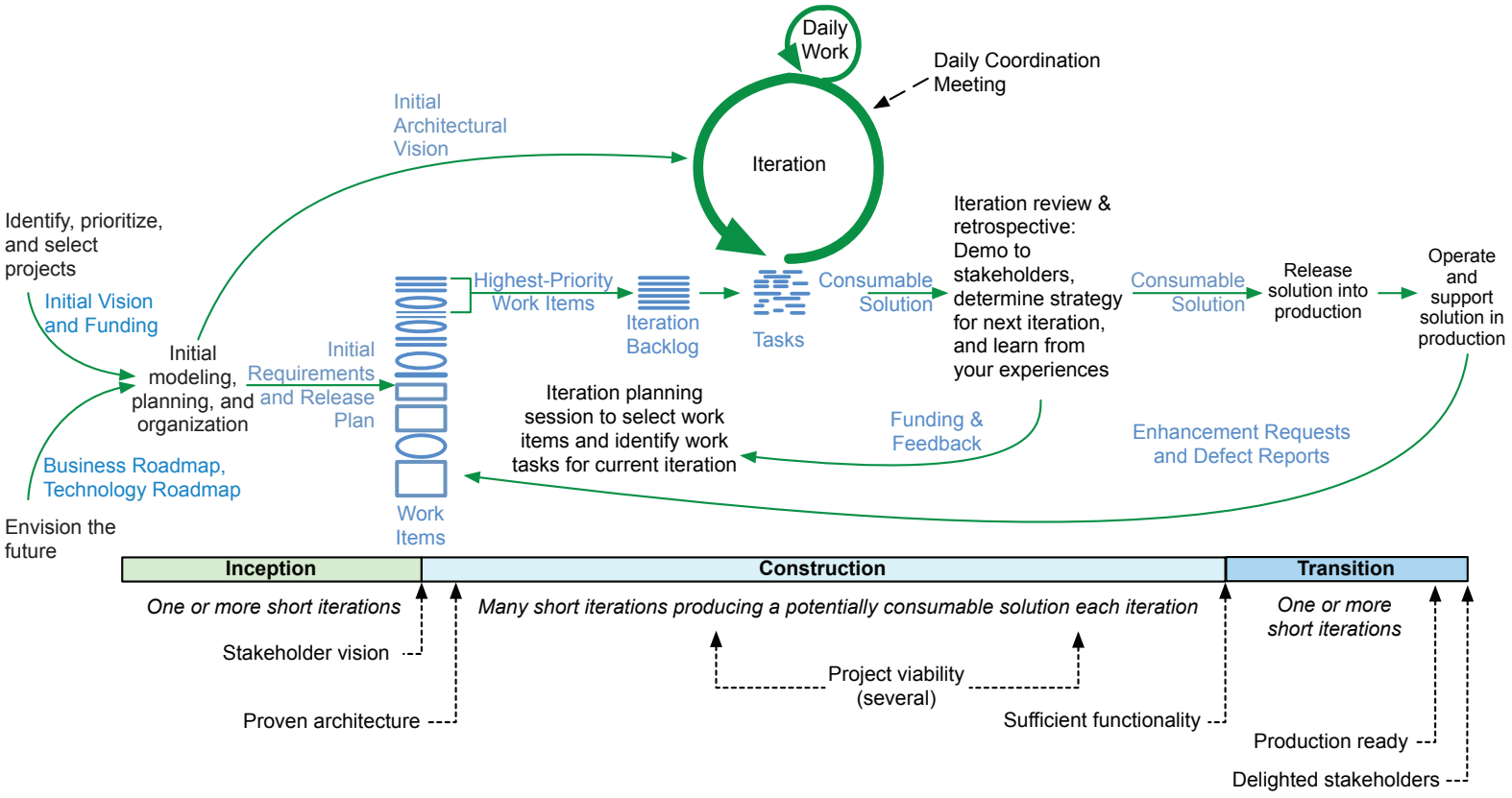

Figure 9. The DAD Agile lifecycle.

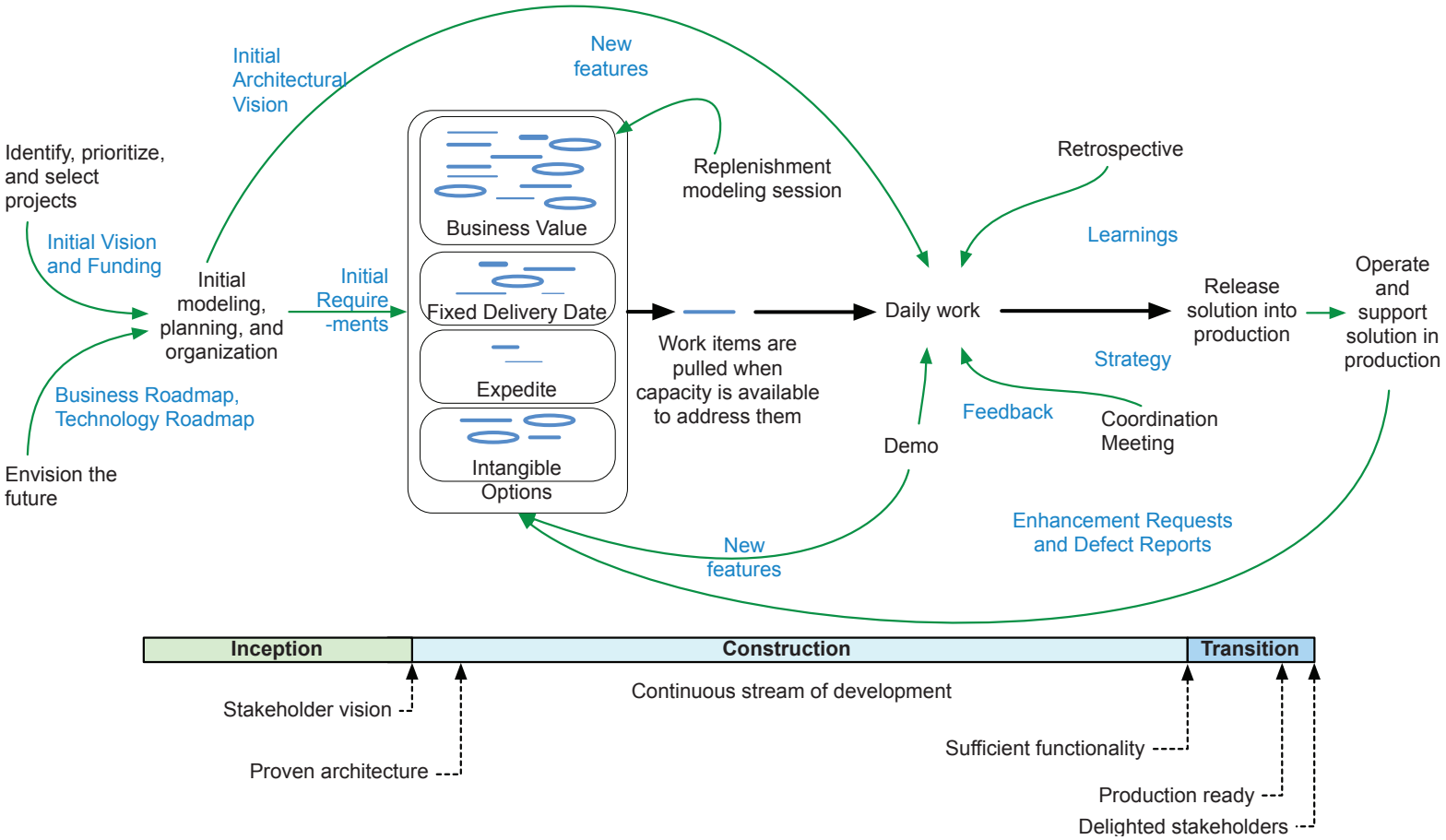

Figure 10. The DAD Lean lifecycle.

strategies should enhance the ability of DAD teams to deliver business value to their stakeholders in a cost effective and timely manner. Unfortunately many existing IT governance strategies are based on a command-and-control, bureaucratic approach which often proves ineffective in practice. Chapter 20 of the DAD book provides a comprehensive discussion of agile governance [3].

- **Open and honest monitoring.** Although agile approaches are based on trust, smart governance strategies are based on a "trust but verify and then guide" mindset. An important aspect of appropriate governance is the monitoring of project teams through various means. One strategy is for anyone interested in the current status of a DAD project team to attend their daily coordination meeting and listen in, a strategy promoted by the Scrum community. Although it's a great strategy that we highly recommend, it unfortunately doesn't scale very well because the senior managers responsible for governance are often busy people with many efforts to govern, not just your team. Hence the need for more sophisticated strategies such as a "development intelligence" approach supported via automated dashboards.

Being enterprise aware has several important implications for the delivery lifecycle. First, to help teams understand the enterprise context that they operate in we should explicitly depict major collaboration flows with other parts of the organization. Figure 9 shows how to do so by evolving the governed agile delivery lifecycle of Figure 4. Note that these flows are not necessarily artifact based, they may represent other forms of communication such as face-to-face discussion.

The second implication is that one lifecycle does not fit all. We have worked with several organizations, some as small as thirty IT staff, that had teams that followed very different lifecycles. For teams that are new to agile the lifecycle of Figure 9 is a great place to start. But, because of the agile philosophy of actively striving to learn and improve your approach teams start to evolve away from the Scrum-based lifecycle. It is common for them to realize that practices such as iteration planning, iteration modeling, retrospectives, and demos do not need to be on the same cadence, that instead they

should be done on a just-in-time (JIT) manner. Once they start implementing these improvements the concept of an iteration/sprint disappears in favor of a continuous flow of delivery. Teams will often realize that there can be significant overhead in maintain a prioritize stack of work items (what Scrum calls a product backlog) and instead decide to pull work into their process JIT when they have the capacity to do so. As a result a team may choose to evolve the lifecycle of Figure 9 into something that looks like Figure 10 based on their growing skills and experiences working in a disciplined agile manner.

The lean lifecycle of Figure 10 is common on sustainment teams responsible for maintaining or evolving one or more solutions. These teams typically get a steady stream of enhancement requests, plus the occasional defect reports, that are best dealt with in a pull-based manner.

Over time the book end phases, Inception and Transition, shrink until you have more of a continuous delivery (CD) type of lifecycle, as shown in Figure 11. In this case we show a CD version of the lean lifecycle from Figure 10 but we could have shown a CD version of the Scrum-based agile lifecycle of Figure 9. Interestingly, we recently spoke with a data warehouse (DW)/business intelligence (BI) team taking an agile delivery approach. For the first and second release of their solution they followed a project lifecycle similar to that of Figure 9, then adopted a lean CD approach as in Figure 11 so that they could respond quickly to requests for new reports and queries. In short, disciplined agile analytics!

Our experience is that some organizations are reticent to recognize the need to support more than one delivery lifecycle within their IT department, often because they recognize that this increases the difficulty for cross-project activities such as portfolio management, governance, and enterprise architecture. We've also found that some organizations are relieved that there is a multiple-lifecycle framework such as DAD available. This occurs when they run into trouble with methods such as Scrum that prescribe a single lifecycle yet have teams in situations where a lean approach is more appropriate. The organizations that desire a single strategy across all of their IT delivery teams tend to believe that repeatable processes are desirable, a dubious assumption at best. Our experience is that your business stakeholders are rarely interested in this, instead they would much rather have repeatable results. For example, stakeholders typically desire to have their IT investment spent wisely, to have solutions that meet their actual needs, to have

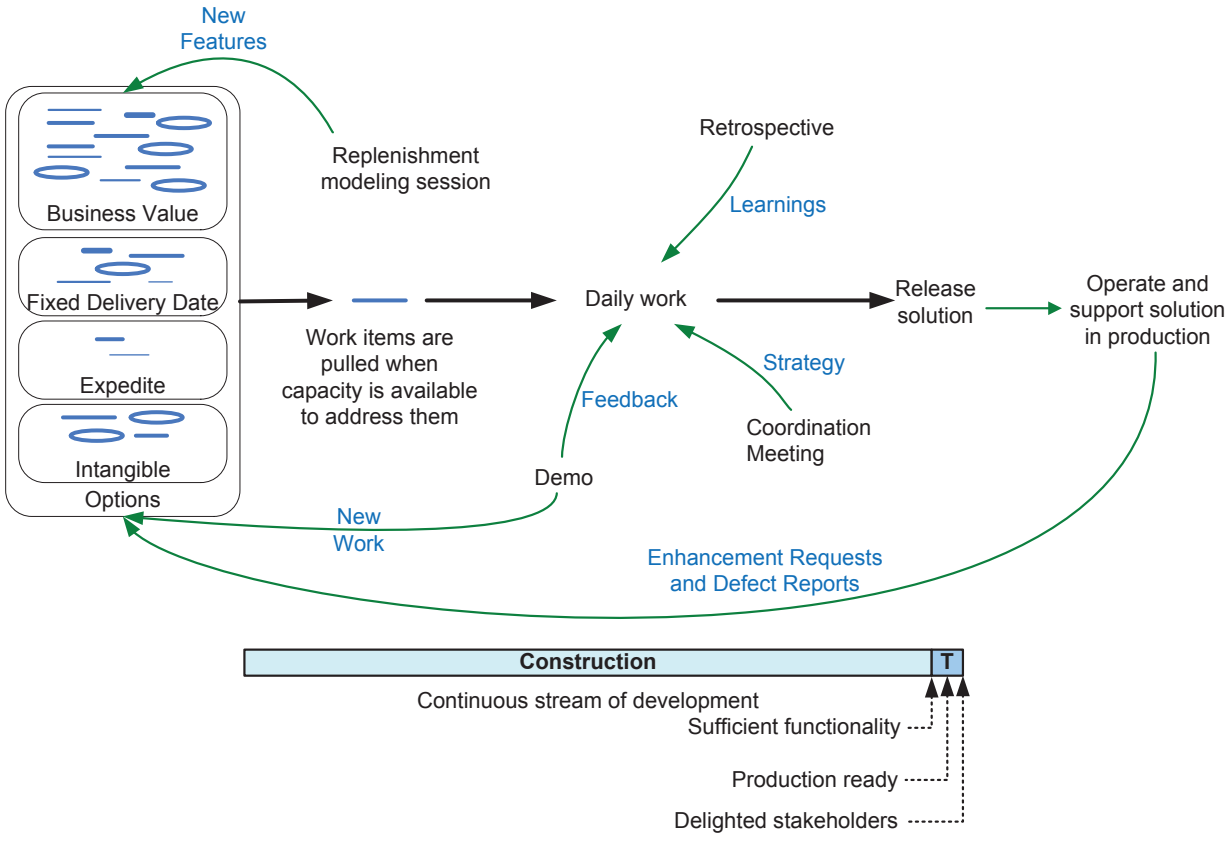

Figure 11. DAD's lean continuous delivery lifecycle.

sufficient quality, and to have the solutions in a timely manner. Enabling teams to select, and then tailor, a lifecycle which reflects the realities of the situation that they face is much more likely to lead to repeatable results than a single "repeatable" process will.

Parting Thoughts

This paper described how to evolve from today's Scrum vision of agile software development to a disciplined agile solution delivery. These steps are:

1. Focus on consumable solutions, not just potentially shippable software
2. Extend Scrum's construction lifecycle to address the full delivery lifecycle
3. Move beyond method branding
4. Adopt explicit governance strategies
5. Take a goal-based approach to enable scaling

Scrum is a good start, but enterprise-class organizations need an approach which is a bit more robust. The Disciplined Agile Delivery (DAD) process decision framework is that approach. For more information about DAD, please visit DisciplinedAgileDelivery.com. For disciplined agile certification, please visit DisciplinedAgileConsortium.org. If you have any questions about DAD, or feedback about this paper to share with us, please contact us at ScottAmbler.com.

References and Recommended Resources

1. Beck, K. et. al. (2001). Manifesto for Agile Software Development. http://agilemanifesto.org/
2. Cockburn, A. (1998). Cooperative game manifesto for software development. http://alistair.cockburn.us/Cooperative+game+manifesto+for+software+development
3. Ambler, S.W. and Lines, M. (2012). Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Development in the Enterprise. IBM Press.
4. Ambler, S.W. (2013). 2013 Agile Project Initiation Survey Results. http://www.ambysoft.com/surveys/projectInitiation2013.html
5. Ambler, S.W. (2010). November 2010 Agile State of the Art Survey Results. http://www.ambysoft.com/surveys/agileStateOfArt201011.html
6. 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
7. Kennaley, M. et. al. (2013). The "New Deal" for Software Development. http://www.software-development-experts.com/the-new-deal.aspx

Acknowledgements

We would like to acknowledge the help of the following people: Kevin Aguanno, Beverley Ambler, Mike Bowler, Riaan du Toit, Julian Holmes, Mark Lines, Glen Little, Mark Kennaley, and Adam Murray.

About the Author

Scott 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 Disciplined Agile Delivery with Mark Lines, the Managing Partner of SA+A. He is also (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.

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 Agile 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 Agile 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 Agile 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.