IBM Software
Design and development

Thought Leadership White Paper

Disciplined Agile Delivery: An introduction

Make no mistake, agile is not a fad. When mainstream agile methods such as Scrum and Extreme Programming (XP) were introduced, the ideas contained in them were not new, nor were they even revolutionary at the time. In fact, many of them have been described in-depth in other methods such as Rapid Application Development (RAD), Evo, and various instantiations of the Unified Process, not to mention classic books such as Frederick Brooks' *The Mythical Man Month*. It should not be surprising that working together closely in co-located teams and collaborating in a unified manner towards a goal of producing working software produces results superior to those based on working in specialized silos concerned with individual rather than team performance. It should also come as no surprise that reducing documentation and administrative bureaucracy saves money and speeds up delivery.

Agile was once considered viable only for small, co-located teams; more recently, improvements in product quality, team efficiency, and on-time delivery—all attributable to agile practices—have caused larger teams to take a closer look at adopting agile principles in their environments. A recent study conducted by the *Agile Journal* determined that 88 percent of companies, many with over 10,000 employees, are using or evaluating agile practices on their projects. Agile is truly poised to become the dominant software development paradigm. This trend is also echoed in other industry studies, including one conducted by *Dr. Dobb's Journal* which found a 76 percent adoption rate of agile techniques, and within those organizations doing agile, 44 percent of the project teams on average are applying agile techniques in some way.

Unfortunately, we need to take adoption rate survey results with a grain of salt: A subsequent Ambysoft survey found that only 53 percent of people claiming to be on "agile teams" actually were. It is clear that agile methods have been overly hyped by various media over the years, leading to abuse and misuse; in fact, the received message regarding agile appears to have justified using little or no process at all. For too many project teams this resulted in anarchy and chaos, leading to project failures and a backlash from the IT community that prefers more traditional approaches.

Properly executed, agile is not an excuse to be undisciplined. It is clear that the execution of mainstream agile methods such as XP have always demanded a disciplined approach, certainly more than traditional approaches such as waterfall methods—don't mistake the high ceremony of many traditional methods to be a sign of discipline, rather it's a sign of rampant and often out-of-control bureaucracy. However, mainstream agile methods don't provide enough guidance for the typical enterprise. Mature implementations of agile recognize a basic need in enterprises for a level of rigor that core agile methods dismiss as not required, such as governance, architectural planning, and modeling. Most mainstream agile methods admit that their strategies require significant additions and adjustments to scale beyond teams of about eight people who are working together in close proximity. Furthermore, most Fortune 1000 enterprises and government agencies have larger solution delivery teams that are often geographically distributed, so the required tailoring efforts can prove both expensive and risky. It is time for a new generation of agile process framework.

Here are the big ideas in this paper:

- People are the primary determinant of success for IT delivery projects.
- Moving to a Disciplined Agile Delivery process is the first step in scaling agile strategies.
- Disciplined Agile Delivery (DAD) is an enterprise-aware hybrid software process framework.
- Agile strategies should be applied throughout the entire delivery life cycle.
- Agile teams are easier to govern than traditional teams.

Context counts—The agile scaling model

To understand the need for the Disciplined Agile Delivery (DAD) process framework you must start by recognizing the realities of the situation you face. The Agile Scaling Model (ASM) is a contextual framework that defines a roadmap to effectively adopt and tailor agile strategies to meet the unique challenges faced by an agile software development team. The first step to scaling agile strategies is to adopt a Disciplined Agile Delivery life cycle that scales mainstream agile construction strategies to address the full delivery process from project initiation to deployment into production. The second step is to recognize which scaling factors, if any, are applicable to a project team and then tailor your adopted strategies to address the range of complexities the team faces.

The ASM, depicted in Figure 1, defines three process categories:

1. **Core agile development:** Core agile methods—such as Scrum, XP, and Agile Modeling (AM)—focus on construction-oriented activities. They are characterized by value-driven life cycles where high-quality, potentially shippable software is produced on a regular basis by a highly collaborative, self-organizing team. The focus is on small (<15 member) teams which are co-located and are developing straightforward software.

2. **Disciplined Agile Delivery:**[1] These methods—including the DAD process framework (described in this paper) and Harmony/ESW—address the full delivery life cycle from project initiation to production. Where appropriate, they add lean governance techniques to balance self organization and add a risk-driven viewpoint to the value-driven approach to increase the chance of project success. Like core agile methods, these methods focus on small co-located teams developing straightforward solutions.

3. **Agility@Scale:** This is Disciplined Agile Delivery, where one or more scaling factors apply. The scaling factors that an agile team may face include team size, physical distribution, organizational distribution, regulatory compliance, cultural or organizational complexity, technical complexity, and enterprise disciplines (such as enterprise architecture, strategic reuse, and portfolio management).

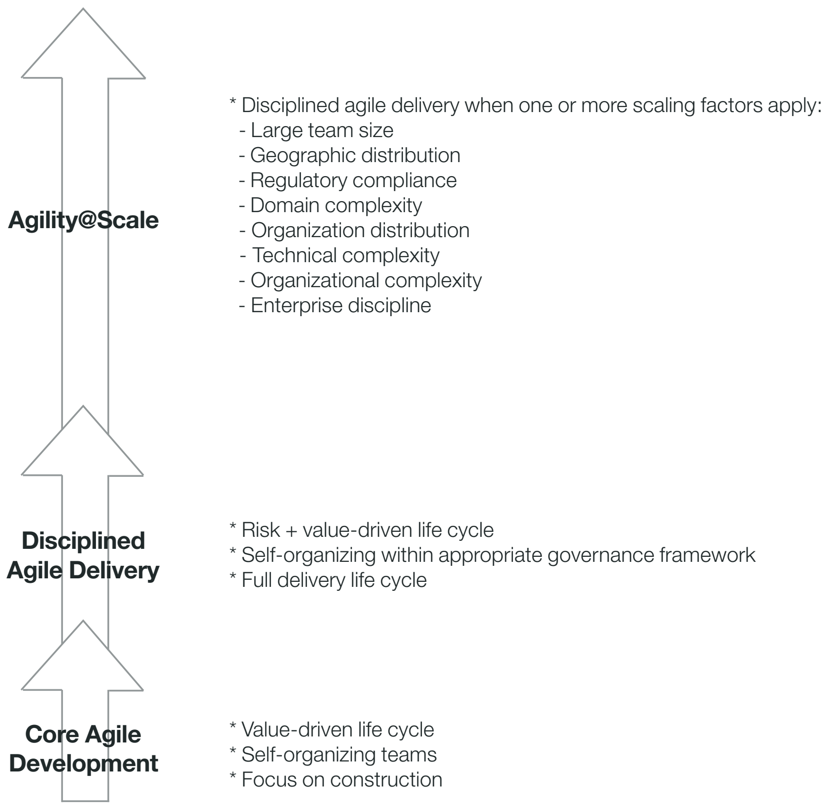

Agility@Scale
* Disciplined agile delivery when one or more scaling factors apply:
- Large team size
- Geographic distribution
- Regulatory compliance
- Domain complexity
- Organization distribution
- Technical complexity
- Organizational complexity
- Enterprise discipline
Disciplined Agile Delivery
* Risk + value-driven life cycle
* Self-organizing within appropriate governance framework
* Full delivery life cycle
Core Agile Development
* Value-driven life cycle
* Self-organizing teams
* Focus on construction

Figure 1: The Agile Scaling Model (ASM)

This paper describes the DAD process framework. In most cases we assume that your team is small (<15 people), either co-located or near-located (in the same building), and working on a relatively straightforward solution.

What is the Disciplined Agile Delivery (DAD) process framework?

Let's begin with a definition:

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

From this definition, you can see that the DAD process framework has several important characteristics. These characteristics are:

- People first
- Learning-oriented
- Agile
- Hybrid
- IT solution focused
- Goal-driven delivery life cycle
- Risk and value driven
- Enterprise aware.

To gain a better understanding of DAD, let's explore each of these characteristics in greater detail.

People first

Alistair Cockburn refers to people as "nonlinear, first-order components" in the software development process. His observation, based on years of ethnographic work, is that people and the way that they collaborate are the primary determinants of success on IT efforts. This philosophy, reflected in the first value statement of the Agile Manifesto, permeates DAD. DAD team members should be self-disciplined and DAD teams should be self organizing and self aware. The DAD process framework provides guidance which DAD teams leverage to improve their effectiveness, but it does not prescribe mandatory procedures.

The traditional approach of having formal handoffs of work products (primarily documents) between different disciplines—such as requirements, analysis, design, test, and development—creates bottlenecks and is a huge waste of time and money. Handoffs between people often create misunderstandings and injection of defects and are described in lean software development as one of the seven sources of waste. When we create a document we will not document our complete understanding of what we are describing and inevitably some knowledge is "left behind" as tacit knowledge that is not passed on. It is easy to see how, after many handoffs, the eventual deliverable may bear little resemblance to the original intent. In an agile environment, the boundaries between disciplines should be torn down and handoffs minimized in the interest of working as a team rather than a group of specialized individuals.

In DAD we foster the strategy of cross-functional teams made up of cross-functional people. There should be no hierarchy within the team, and team members are encouraged to be cross-functional in their skill set and indeed perform work related to disciplines other than their specialty. The increased understanding gained beyond a team member's primary discipline results in more effective use of resources and the reduced reliance on formal documentation and sign-offs.

As such, agile methods deemphasize roles based strictly on skillsets in favor of primary roles that can include a variety of skills. Accordingly, the five primary roles of DAD are:

1. **Stakeholder:** A stakeholder is someone who is materially impacted by the outcome of the solution. The stakeholder is clearly more than an end user: A stakeholder could be a direct user, indirect user, manager of users, senior manager, operations staff member, the "gold owner" who funds the project, support (help desk) staff member, auditor, your program/portfolio manager, developers working on other systems that integrate or interact with the one under development, or maintenance professionals potentially affected by the development and/or deployment of a software project.

2. **Product owner:** The product owner is the individual on the team who speaks as the "one voice of the customer." They represent the needs and desires of the stakeholder community to the agile delivery team. As such, he or she clarifies any details regarding the solution and is also responsible for maintaining a prioritized list of work items that the team will implement to deliver the solution. While the product owner may not be able to answer all questions, it is their responsibility to track down the answer in a timely manner so that the team can stay focused on their tasks. Having a product owner working closely with the team to answer any question about work items as they are being implemented substantially reduces the need for documentation. Each DAD team, or sub-team in the case of large programs organized into a team of teams, has a single product owner. A secondary goal for a product owner is to represent the work of the agile team to the stakeholder community. This includes arranging demonstrations of the solution as it evolves and communicating project status to key stakeholders.

3. **Team member:** The team member focuses on producing the actual solution for stakeholders. Team members will perform testing, analysis, architecture, design, programming, planning, estimation, and many more activities as appropriate throughout the project. Note that not every team member will have every single one of these skills, at least not yet, but they will have a subset of them and they will strive to gain more skills over time. Team members are sometimes described by core agile methods as "developers" or simply as programmers. However, in DAD we recognize that not every team member necessarily writes code.

4. **Team lead:** The team lead is the agile coach, helping to keep the team focused on delivering work items and fulfilling their iteration goals and commitments that they have made to the product owner. He or she acts as a true leader, facilitating communication, empowering them to self-optimize their processes, ensuring that the team has the resources that it needs, and removing any impediments to the team (issue resolution) in a timely manner.

5. **Architecture owner:** The architecture owner makes the architecture decisions for the team and facilitates the creation and evolution of the overall solution design. Architecture is a key source of project risk and someone needs to be responsible for ensuring the team mitigates this risk. Note that the architecture owner doesn't dictate the architecture of the solution, but instead leads its formulation. On small projects the team lead is often the architecture owner.

Notice that tester and business analyst are not primary roles in the DAD process framework. Rather, a generic team member should be capable of doing multiple things. A team member who specializes in testing might be expected to volunteer to help with requirements, or even taking a turn at being the Scrum Master (team lead). This doesn't imply that everyone needs to be an expert at everything, but it does imply that as a whole the team should cover the skills required of them, and should be willing to pick up any missing skills as needed.

Team members should be "generalizing specialists"—a specialist in one or more disciplines but with general knowledge of other disciplines as well. More important, generalizing specialists are willing to collaborate closely with others, to share their skills and experiences with others and to pick new skills up from the people they work with. A team made up of generalizing specialists requires few handoffs between people, enjoys improved collaboration because the individuals have a greater appreciation of the background skills and priorities of the various IT disciplines, and can focus on what needs to be done as opposed to focusing on whatever their specialties are.

DAD teams and team members should be:
- Self-disciplined, in that they commit only to the work which they can accomplish and then perform that work as effectively as possible.
- Self-organizing, in that they will estimate and plan their own work and then proceed to collaborate iteratively to do so.
- Self-aware, in that they strive to identify what works well for them, what doesn't, and then learn and adjust accordingly.

Although people are the primary determinant of success for IT projects, in most situations it isn't effective to simply put together a good team of people and let them loose on the problem at hand. If you do this the teams run several risks, including investing significant time in developing their own processes and practices, in identifying the wrong processes and practices, in not identifying the right processes and practices, and in tailoring those processes and practices ineffectively. In other words, people are not the *only* determinant of success. The DAD process framework provides coherent, proven advice that agile teams can leverage and thereby avoid or at least minimize the risks described above.

Learning-oriented

In the years since the Agile Manifesto, we've discovered that the most effective organizations are the ones that promote a learning environment for their staff. There are three key aspects which a learning environment must address. The first is domain learning—how are you exploring and identifying what your stakeholders need, and perhaps more importantly how are you helping them to do so? The second is learning to improve your process at the individual, team, and enterprise levels. The third is technical learning, which focuses on understanding how to effectively work with the tools and technologies being used to craft the solution for your stakeholders.

The DAD process framework suggests several strategies to support domain learning, including initial requirements envisioning, incremental delivery of a potentially consumable solution, and active stakeholder participation through the life cycle. To support process-focused learning, DAD promotes the adoption of retrospectives where the team explicitly identifies potential process improvements, a common agile strategy, as well as continued tracking of those improvements. Within IBM Software Group we've found that agile teams that held retrospectives improved their productivity more than teams that did not, and teams that tracked their implementation of the identified improvement strategies were even more successful. Technical learning often comes naturally to IT professionals, many of whom are eager to work with and explore new tools, techniques, and technologies. This can be a double-edged sword—although they're learning new technical concepts they may not invest sufficient time to master a strategy before moving on to the next one, or may abandon a perfectly fine technology simply because they want to do something new.

There are many general strategies for improving your learning capability. Improved collaboration between people correspondingly increases the opportunities for people to learn from one another, and high collaboration is a hallmark of agility. Investing in training, coaching, and mentoring are productive learning strategies as well. Less intuitive, though, is the value in moving away from specialization within your staff and instead fostering more robust skills—valuing, that is, the generalizing specialist. Progressive organizations aggressively promote learning opportunities for their people outside their specific areas of specialty as well as opportunities to actually apply these new skills.

If you're experienced with, or at least have read about, agile software development, then the previous strategies should sound very familiar. Where the DAD process framework takes learning further is through enterprise awareness. Core agile methods such as Scrum and XP are typically project focused, whereas DAD explicitly strives to both leverage and enhance the organizational ecosystem in which a team operates. So DAD teams should leverage existing lessons learned from other agile teams and also take the time to share their own experiences. The implication is that your IT department needs to invest in a technology for socializing the learning experience across teams. In 2005, IBM Software Group implemented internal discussion forums, wikis, and a center of competency (some organizations call them centers of excellence) to support their agile learning efforts. A few years later they adopted a Web 2.0 strategy based on IBM® Lotus® Connections to support enterprise learning.

Agile

The DAD process framework adheres to and enhances the values and principles of the Agile Manifesto. Teams following either iterative or agile processes have been shown to produce higher quality, provide greater return on investment (ROI), provide greater stakeholder satisfaction, and deliver quicker as compared to either a traditional/waterfall approach or an ad-hoc (no defined process) approach. High quality is achieved through techniques such as continuous integration (CI), developer regression testing, test-first development, and refactoring. Improved ROI comes from a greater focus on high-value activities, through working in priority order, through automation of as much of the IT drudgery as possible, through self organization, through close collaboration, and in general from working smarter, not harder. Greater stakeholder satisfaction is achieved by enabling active stakeholder participation, by incrementally delivering a potentially consumable solution with each iteration, and by enabling stakeholders to evolve their requirements throughout the project.

A hybrid process framework

DAD is the formulation of many strategies and practices from both mainstream agile methods as well as other sources. The DAD process framework extends the Scrum construction life cycle to address the full delivery life cycle while adopting strategies from several agile and lean methods. Many of the practices suggested by DAD are commonly discussed in the agile community—such as continuous integration (CI), daily coordination meetings, and refactoring—and some are "advanced" practices commonly applied but, for some reason, not commonly discussed. These advanced practices include initial requirements envisioning, initial architecture envisioning, and end-of-life cycle testing, to name a few.

The DAD process framework is a hybrid: i.e., it adopts and tailors strategies from a variety of sources. A common pattern we've frequently seen within organizations is that they adopt the Scrum process framework, and then do significant work to tailor ideas from other sources to flesh it out. This sounds like a great strategy, and it certainly is if you're a consultant specializing in agile adoption, until you notice that organizations tend to tailor

Scrum in the same sort of way. So, why not start with a more robust process framework which has done this common work in the first place? The DAD process framework adopts strategies from the following methods:

1. **Scrum:** The focus of Scrum is on project leadership and some aspects of requirements management. DAD adopts and tailors many ideas from Scrum, such as working from a stack of work items in priority order, having a product owner responsible for representing stakeholders, and producing a potentially consumable solution from each iteration. However, DAD abandoned most of Scrum's terminology—nobody sprints through a race, people get hurt in rugby scrums, and don't get us going on the term "master"—with the exception of the term product owner.

2. **Extreme programming (XP):** XP is an important source of development practices for DAD, including but not limited to continuous integration (CI), refactoring, test-driven development (TDD), collective ownership, and many more.

3. **Agile Modeling (AM):** As the name implies, AM is the source for DAD's modeling and documentation practices. This includes requirements envisioning, architecture envisioning, iteration modeling, continuous documentation, and just-in-time (JIT) model storming.

4. **Unified Process (UP):** DAD adopts many of its governance strategies from agile instantiations of the UP, including OpenUP and Agile Unified Process (AUP). In particular, this includes strategies such as having light-weight milestones and explicit phases. We also draw from the Unified Process' focus on the importance of proving out the architecture in the early iterations and reducing all types of risk early in the life cycle.

5. **Agile Data (AD):** As the name implies AD is a source of agile database practices, such as database refactoring, database testing, and agile data modeling. It is also an important source of agile enterprise strategies, such as how agile teams can work effectively with enterprise architects and enterprise data administrators.

6. **Kanban:** DAD adopts two critical concepts—limiting work in progress and visualizing work—from Kanban, which is a lean framework. These concepts are in addition to the seven principles of lean software development (eliminate waste, build in quality, create knowledge, defer commitment, deliver quickly, respect people, and optimize the whole).

Solutions over software

The DAD approach will advance your focus from producing software to providing solutions—which is where real business value lies for your stakeholders. A fundamental observation is that as IT professionals we do far more than just develop software. Yes, software is clearly important, but in addressing the needs of our stakeholders we will often provide new or upgraded hardware, change the business/operational processes that stakeholders follow, and even help change the organizational structure in which our stakeholders work.

This shift in focus requires your organization to address some of the prejudices that crept into the Agile Manifesto. We fully endorse the manifesto, but its original signatories were primarily software developers, software development consultants, or both. It is little wonder that the language of their manifesto shows a bias towards software development, which is one of many areas of expertise involved in the complete software delivery life cycle. The DAD process frameworks promotes activities that explicitly address user experience (UX), database, business process, and documentation issues (to name a few) to help project teams think beyond software development alone.

Goal-driven delivery life cycle

DAD addresses the project life cycle from the point of initiating the project through construction to the point of releasing the solution into production. We explicitly observe that each iteration is NOT the same. Projects do evolve and the work emphasis changes as we move through the life cycle. To make this clear, we carve the project into phases with light-weight milestones to ensure that we are focused on the right things at the right time, such as initial visioning, architectural modeling, risk management, and deployment planning. This differs from mainstream agile methods, which typically focus on the construction aspects of the life cycle; details about how to perform initiation and release activities, or even how they fit into the overall life cycle, are typically vague and left up to you.

Time and again, whenever either of us worked with a team which had adopted Scrum we found that they had tailored the Scrum life cycle into something similar to Figure 2, which shows the life cycle of a DAD project.[2] This life cycle has several critical features:

1. **It's a delivery life cycle:** The DAD life cycle extends the Scrum construction life cycle to explicitly show the full delivery life cycle from the beginning of a project to the release of the solution into production (or the marketplace).

2. **There are explicit phases:** The DAD life cycle is organized into three distinct, named phases, reflecting the agile coordinate-collaborate-conclude (3C) rhythm.

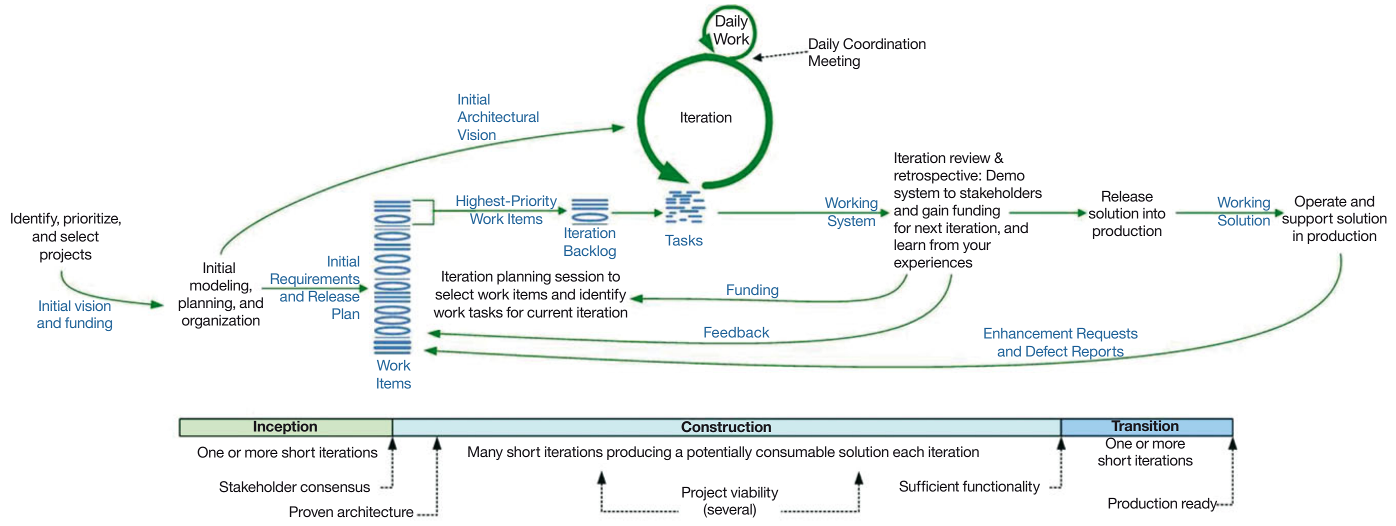

Daily Work
Daily Coordination Meeting
Iteration
Initial Architectural Vision
Identify, prioritize, and select projects
Initial vision and funding
Initial modeling, planning, and organization
Initial Requirements and Release Plan
Work Items
Highest-Priority Work Items
Iteration Backlog
Tasks
Iteration planning session to select work items and identify work tasks for current iteration
Working System
Iteration review & retrospective: Demo system to stakeholders and gain funding for next iteration, and learn from your experiences
Funding
Feedback
Release solution into production
Working Solution
Operate and support solution in production
Enhancement Requests and Defect Reports
Inception
Construction
Transition
One or more short iterations
Many short iterations producing a potentially consumable solution each iteration
One or more short iterations
Stakeholder consensus
Proven architecture
Project viability (several)
Sufficient functionality
Production ready

Figure 2: The Disciplined Agile Delivery (DAD) life cycle

Goals for the Inception Phase	Goals for Construction Phase Iterations	Goals for the Transition Phase
- Identify the vision for the project - Bring stakeholders to agreement around the vision - Identify initial technical strategy, initial requirements, and project plan - Form initial team - Secure funding - Identify risks	- Produce a potentially consumable solution - Address changing stakeholder needs - Move closer to deployable release - Maintain or improve upon existing levels of quality - Address highest risk(s)	- Ensure the solution is production ready - Ensure the stakeholders are prepared to receive the solution - Deploy the solution into production

Ongoing Goals	
- Fulfill the project mission - Grow team members skills - Enhance existing infrastructure	- Improve team process and environment - Leverage existing infrastructure

Table 1: Goals addressed throughout a DAD project

3. **The delivery life cycle is shown in context:** The DAD life cycle recognizes that activities to identify and select projects occur long before, sometimes even years before, their official start. It also recognizes that the solution produced by a DAD project team must be operated and supported once it is delivered into production, and that important feedback comes from people using previously released versions of the solution.

4. **There are explicit milestones:** The milestones are an important governance and risk reduction strategy inherent in DAD.

One of the challenges in describing a process framework is that you need to provide sufficient guidance to help people understand it, but if you provide too much guidance you become overly prescriptive. As we've helped various organizations improve their software processes over the years, we've come to believe that the various process proponents are coming from one extreme or the other. Either there are very detailed processes descriptions—the IBM Rational® Unified Process (RUP) is one such example—or there are very light-weight process descriptions, Scrum being a perfect example. The challenge with RUP is that many teams didn't have the skill to tailor it down appropriately, often resulting in extra work being performed. On the other hand many Scrum teams had the opposite problem with not knowing how to tailor it up appropriately, resulting in significant effort spent reinventing or relearning techniques to address the myriad issues that Scrum doesn't cover. Either way, a lot of waste could have been avoided if only there was an option between these two extremes.

To address this challenge the DAD process framework is goals-driven, as summarized in Table 1. There are of course many ways which these goals can be addressed, so simply indicating the goals is of little value. Our experience is that this goals-driven, suggestive approach provides just enough guidance for solution delivery teams while being sufficiently flexible so that teams can tailor the process to address the context of the situation that they find themselves in.

Table 1 doesn't provide a full listing of the goals your team will address. There are several personal goals of individuals, such as specific learning goals, the desire for interesting work, for compensation, and for public recognition of their work. There are also specific stakeholder goals that will be unique to your project.

Let's overview the DAD phases to better understand the contents of the DAD process framework.[3]

The inception phase

Before jumping into building or buying a solution, it is worth spending some time identifying the objectives for the project. Traditional methods invest a large amount of effort and time planning their projects up front. Agile approaches suggest that too much detail up front is not worthwhile since little is known about what is truly required as well as achievable within the time and budget constraints. Mainstream agile methods suggest that very little effort be invested in upfront planning. Their mantra can be loosely interpreted as "let's just get started and we will determine where we are going as we go". To be fair, some agile methods have a short planning iteration, called "Sprint 0" in Scrum, and the "Planning Game" in Extreme Programming (XP) that on average takes 3.9 weeks.[4] In DAD, we recognize the need to point the ship in the right direction before going full-speed ahead—typically between a few days and a few weeks—to initiate the project. Figure 3 overviews the potential activities that may occur during Inception. This phase ends when the team has developed a vision for the release that the stakeholders agree to and has obtained support for the rest of the project (or at least the next stage of it).

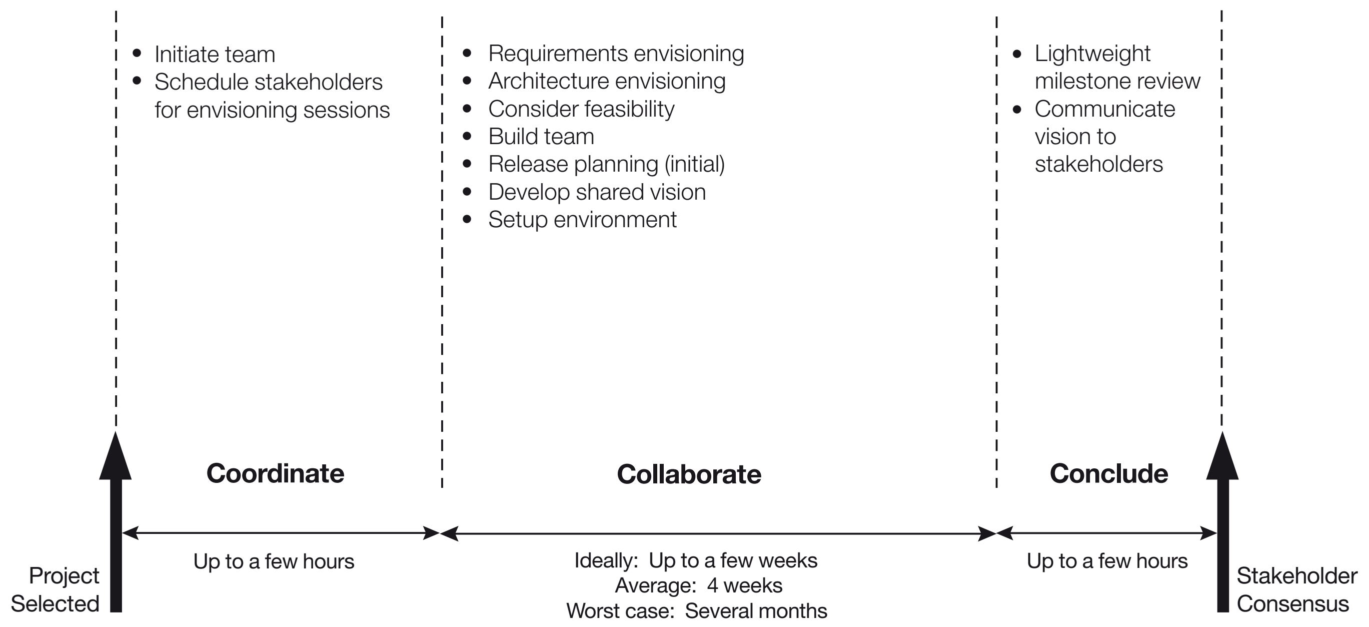
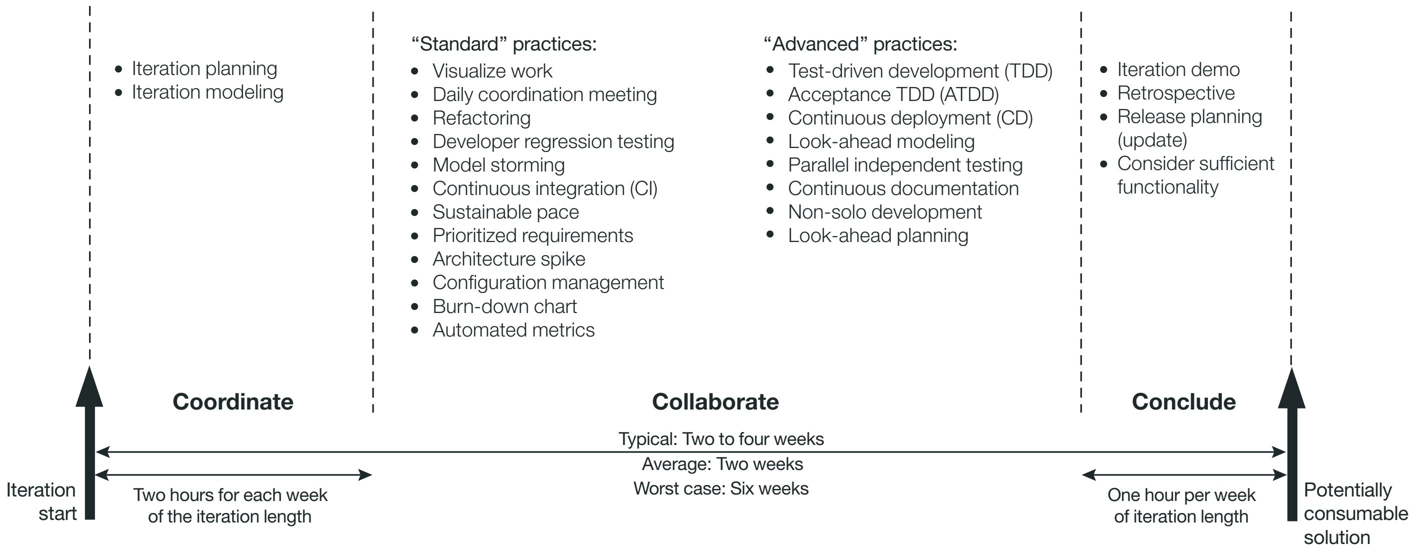

• Initiate team
• Schedule stakeholders for envisioning sessions
• Requirements envisioning
• Architecture envisioning
• Consider feasibility
• Build team
• Release planning (initial)
• Develop shared vision
• Setup environment
• Lightweight milestone review
• Communicate vision to stakeholders
Coordinate
Collaborate
Conclude
Project Selected
Up to a few hours
Ideally: Up to a few weeks
Average: 4 weeks
Worst case: Several months
Up to a few hours
Stakeholder Consensus

Figure 3: Inception phase overview

The construction phase

The construction phase in DAD is the period of time during which the required functionality is built. The timeline is split up into a number of time-boxed iterations. These iterations, the potential activities of which are overviewed in Figure 4, should be the same duration for a given project—typically one week to four weeks, with two and four weeks being the most common—and typically do not overlap. At the end of each iteration a demonstrable increment of a potentially consumable solution has been produced and regression tested. The construction phase ends where there is sufficient functionality to justify the cost of transition, sometimes referred to as minimally marketable release (MMR), and which the stakeholders believe is acceptable to them.

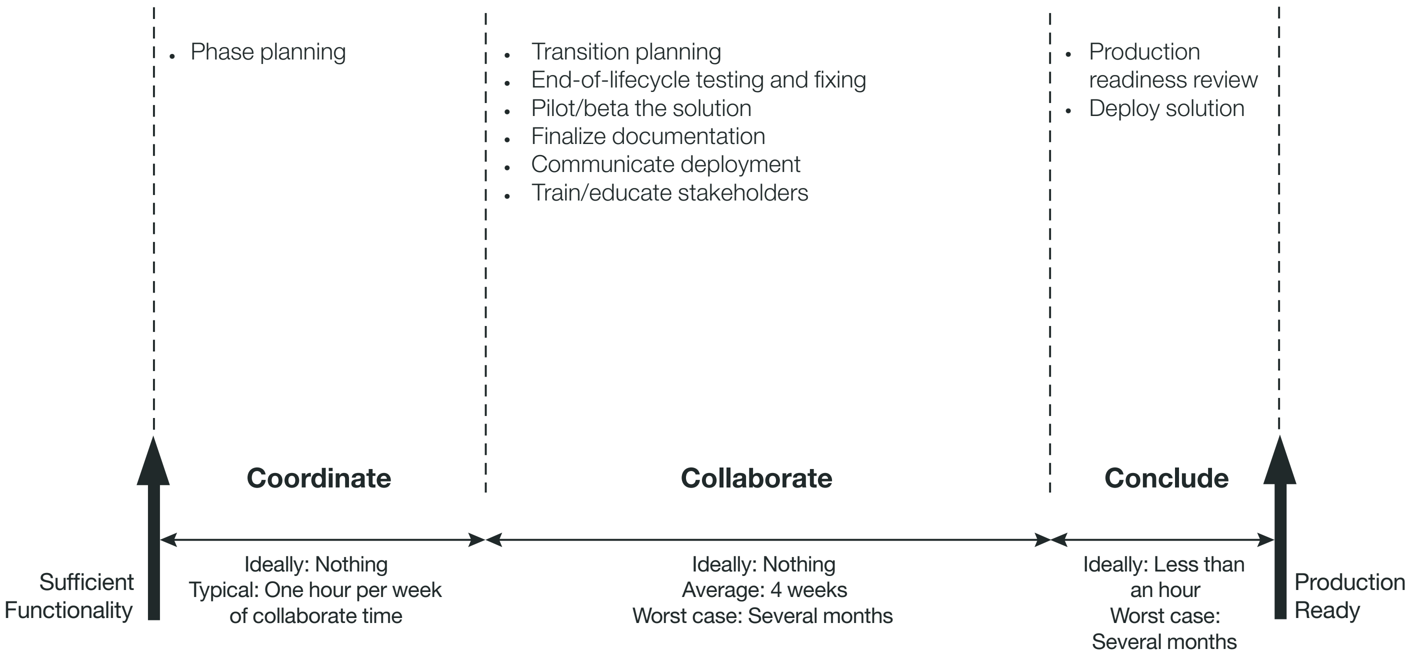


• Iteration planning
• Iteration modeling
"Standard" practices:
• Visualize work
• Daily coordination meeting
• Refactoring
• Developer regression testing
• Model storming
• Continuous integration (CI)
• Sustainable pace
• Prioritized requirements
• Architecture spike
• Configuration management
• Burn-down chart
• Automated metrics
"Advanced" practices:
• Test-driven development (TDD)
• Acceptance TDD (ATDD)
• Continuous deployment (CD)
• Look-ahead modeling
• Parallel independent testing
• Continuous documentation
• Non-solo development
• Look-ahead planning
• Iteration demo
• Retrospective
• Release planning (update)
• Consider sufficient functionality
Coordinate
Collaborate
Conclude
Typical: Two to four weeks
Average: Two weeks
Worst case: Six weeks
Iteration start
Two hours for each week of the iteration length
One hour per week of iteration length
Potentially consumable solution

Figure 4: Construction iteration overview

The transition phase

The transition phase focuses on delivering the system into production (or into the marketplace in the case of a consumer product). As you can see in Figure 5 there is more to transition than merely copying some files onto a server. The time and effort spent transitioning varies from project to project. Shrink-wrapped software entails the manufacturing and distribution of software and documentation. Internal systems are generally simpler to deploy than external systems. High visibility systems may require extensive beta testing by small groups before release to the larger population. The release of a brand new system may entail hardware purchase and setup, while updating an existing system may entail data conversions and extensive coordination with the user community. Every project is different. The transition phase ends when the stakeholders are ready and the system is fully deployed.


• Phase planning
• Transition planning
• End-of-lifecycle testing and fixing
• Pilot/beta the solution
• Finalize documentation
• Communicate deployment
• Train/educate stakeholders
• Production readiness review
• Deploy solution
Coordinate
Collaborate
Conclude
Sufficient Functionality
Ideally: Nothing
Typical: One hour per week of collaborate time
Ideally: Nothing
Average: 4 weeks
Worst case: Several months
Ideally: Less than an hour
Worst case: Several months
Production Ready

Transition phase overview

Enterprise aware

DAD teams work within your organization's enterprise ecosystem, as do other teams, and explicitly try to take advantage of the opportunities presented to them—to borrow an environmental cliché, disciplined agilists act locally and think globally. This includes working closely with the following teams and individuals: enterprise technical architects, and reuse engineers to leverage and enhance[5] the existing and "to be" technical infrastructure; enterprise business architects and portfolio managers to fit into the overall business ecosystem; senior managers who should be governing the various teams appropriately; data administrators to access and improve existing data sources; and IT development support people to understand and follow enterprise IT guidance (such as coding, user interface, security, and data conventions to name a few). In other words, DAD teams should adopt what Mark refers to as a "whole enterprise" mindset.

With the exception of start-up companies, agile delivery teams don't work in a vacuum. There are often existing systems currently in production, and minimally your solution shouldn't impact them although your solution should leverage existing functionality and data available in production. There are often 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. There may be a common vision which your organization is working towards, a vision which your team should contribute to. There will be a governance strategy in place, although it may not be obvious to you, which hopefully enhances what your team is doing.

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 interesting for you. Unfortunately this isn't always the case. Some IT "professionals" will choose to work with new technologies, and even implement them in the solutions that they produce, not because those technologies are what's most appropriate for the projects at hand but because they help to improve their resume. Or they'll choose to build something from scratch, or use new development tools, or create new data sources, when perfectly good ones already exist within the organization. We can and should do better, and we can do so by:

1. **Leveraging enterprise assets:** There are many enterprise assets, or at least there should be, which you can use and evolve. This includes common development guidelines, such as coding standards, data conventions, security guidelines, and user interface standards. But enterprise assets are far more than standards. If your organization uses a disciplined architecture-centric approach to building enterprise software, there will be a growing library of service-based components to reuse and improve upon for the benefit of all current and future solutions. DAD teams strive to work to a common infrastructure, for example using the enterprise-approved technologies and data sources whenever possible, and better yet they work to the "to be" vision for your infrastructure. To do this DAD teams will collaborate with enterprise professionals—including enterprise architects, enterprise business modelers, data administrators, operations staff, and reuse engineers—throughout the life cycle and particularly during Inception during envisioning efforts. Leveraging enterprise assets increases consistency and thereby ease of maintenance, decreases development costs and time, and decreases operational costs.

2. **Enhance your organizational ecosystem:** The solution being delivered by a DAD team should minimally fit into the existing organizational ecosystem[6]—the business processes and systems supporting them. Better yet, it should enhance that ecosystem. To do this, the first step is to leverage existing enterprise assets wherever possible as described earlier. DAD teams will work with operations and support staff closely throughout the life cycle, particularly the closer you get to releasing into production, to ensure that they understand the current state and direction of the organizational ecosystem. DAD teams will often be supported by an independent test team that will do production integration testing (among other things) to ensure that your solution works within the target production environment which it will face at deployment time.

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 the daily coordination meeting and listen in; this is a strategy promoted by the Scrum community. Although we highly recommend this, 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. In fact Scott found exactly this in the 2010 "How Agile Are You?" survey. Another approach, one we've seen to be incredibly effective, is for DAD teams to use instrumented and integrated tooling, particularly Jazz™-enabled products such as IBM Rational Team Concert™ software, which generates metrics in real time that can be displayed on project dashboards. You can see an example of such a dashboard for the Jazz team itself at jazz.net, a team following an open commercial strategy.

A third strategy is to follow a risk-driven life cycle, discussed in the next section, with explicit milestones which provide consistent and coherent feedback as to the project status to interested parties.

Risk and value-driven

The DAD process framework adopts what is called a risk-value life cycle; effectively, this is a light-weight version of the strategy promoted by the Unified Process (UP). DAD teams strive to address common project risks, such as coming to stakeholder consensus around the vision and proving the architecture, early in the life cycle. DAD also includes explicit checks for continued project viability, whether sufficient functionality has been produced, and whether the solution is production ready. It is also value-driven, a strategy which reduces delivery risk, in that DAD teams produce potentially consumable solutions on a regular basis.

It has been said, "Attack the risks before they attack you." This is a philosophy that is consistent with the DAD approach. The DAD risk-value driven life cycle is an extension of the value-driven life cycle common to methods such as Scrum and XP. With a value driven life cycle you produce potentially shippable software with each iteration, or more accurately (from a DAD perspective) a potentially consumable solution with each iteration. The features delivered represent those in the requirements backlog that are of highest value from the perspective of the stakeholders. With a risk-value driven life cycle you also consider features related to risk as high priority items, not just high-value features. With this in mind we explicitly address risks which are common to IT delivery projects as soon as we possibly can. To be fair, value-driven life cycles address three important risks: 1) the risk of not delivering at all, 2) the risk of delivering the wrong functionality, and 3) political risks resulting from lack of visibility into what the team is producing. Addressing these risks is a great start, but it's not the full risk mitigation picture.

First and foremost, DAD includes and extends standard strategies of agile development methods to reduce common IT delivery risks:

1. **Potentially consumable solutions:** DAD teams produce potentially consumable solutions with every construction iteration, extending Scrum's strategy of potentially shippable software to address usability concerns (the consumability aspect) and the wider issue of producing solutions and not just software. This reduces delivery risk because the stakeholders are given the option to have the solution delivered into production when it makes sense to do so.

2. **Iteration demos:** At the end of each construction iteration the team should demo what they have built to their key stakeholders. The primary goal is to obtain feedback from the stakeholders and thereby improve the solution they're producing, decreasing functionality risk. A secondary goal is to indicate the health of the project by showing their completed work, thereby decreasing political risk (assuming the team is working successfully).

3. **Active stakeholder participation:** The basic idea is that not only should stakeholders, or their representatives (i.e. product owners), provide information and make decisions in a timely manner but they can also be actively involved in the development effort itself. For example, stakeholders can often be actively involved in modeling when inclusive tools such as paper and whiteboards are used. Active stakeholder involvement through the entire iteration, and not just at demos, helps to reduce both delivery and functionality risk due to the greater opportunities to provide feedback to the team.

The mainstream agile strategies to addressing risk on IT delivery projects are a good start, but only a start. DAD also adopts explicit, light-weight milestones to further reduce risk. At each of these milestones an explicit assessment as to the viability of the project is made by key stakeholders and a decision as to whether the project should proceed is made. These milestones, indicated on the DAD life cycle depicted in Figure 2, are:

1. **Stakeholder consensus:** Held at the end of the Inception phase, the goal of this milestone is to ensure that the project stakeholders have come to a reasonable consensus as to the vision of the release. By coming to this agreement we reduce both functionality and delivery risk substantially even though very little investment has been made to date in the development of a working solution. You should expect to cancel ten to fifteen percent of your projects at this milestone.

2. **Proven architecture:** In the early construction phase iterations we are concerned with reducing most risk and uncertainty related to the project. Risk can be related to many things such as requirements uncertainty, team productivity, business risk, and schedule risk. However, at this point in time much of the risk on an IT delivery project is typically related to technology, specifically at the architecture level. Although the high-level architecture models created during the Inception phase are helpful for thinking through the architecture, the only way to be truly sure that the architecture can support the requirements is by proving it with working code. This is a vertical slice through the software and hardware tiers that touches all points of the architecture from end to end. In the UP this is referred to as "architectural coverage" and in XP as a "steel thread" or "tracer bullet." By writing software to prove out the architecture, DAD teams greatly reduce a large source of technical risk and uncertainty by discovering and then addressing any deficiencies in their architecture early in the project.

3. **Continued viability:** In Scrum the idea is that at the end of each sprint (iteration) your stakeholders consider the viability of your project. In theory this is a great idea, but in practice it rarely seems to happen. The cause of this problem is varied—perhaps the stakeholders being asked to make this decision have too much political stake in the project to back out of it unless things get really bad and perhaps psychologically people don't notice that a project gets into trouble in the small periods of time typical of agile iterations. The implication is that you need to have purposeful milestone reviews where the viability of the project is explicitly considered. We suggest that you do so once a quarter, implying that this milestone may occur several times during the construction phase of a longer release or not at all for a shorter one.

4. **Sufficient functionality:** The construction phase milestone is reached when enough functionality has been completed to justify the expense of transitioning the solution into production. The solution must meet the acceptance criteria agreed to earlier in the project, or be close enough that it is likely any critical quality issues will be addressed during the transition phase.

5. **Production ready:** At the end of the transition phase your key stakeholders will need to determine if the solution should be released into production. At this milestone, the business stakeholders are satisfied with and accept the system, the people responsible for operating the system once it is in production are satisfied with the relevant procedures and documentation, and the people responsible for supporting the system once it is in production are satisfied with the relevant procedures and documentation.

Concluding thoughts

The good news is that evidence clearly shows that agile methods deliver superior results compared to traditional approaches and that the majority of organizations are either using agile techniques or plan to in the near future. The bad news is that the mainstream agile methods—including Scrum, Extreme Programming (XP), and Agile Modeling (AM)—provide only a part of the overall picture for IT solution delivery. Disciplined Agile Delivery (DAD) is a hybrid process framework that pulls together common practices and strategies from these methods, and more, to address the full delivery life cycle. DAD puts people first, recognizing that individuals and the way that they work together are the primary determinants of success on IT projects. DAD is enterprise aware, motivating teams to leverage and enhance their existing organizational ecosystem, to follow enterprise development guidelines, and to work with enterprise administration teams. The DAD life cycle includes explicit milestones to reduce project risk and increase external visibility of key issues to support appropriate governance activities by senior management.

For more information

For more detailed discussions about several of the topics covered in this paper, refer to:

- **The Agile Manifesto:** The 4 values of the Agile Manifesto are posted at http://www.agilemanifesto.org/ and the twelve principles behind it at http://www.agilemanifesto.org/principles.html.
- **Agile surveys:** Throughout the paper we referenced several surveys. *The Agile Journal* Survey is posted at http://www.agilejournal.com/. The results from the *Dr. Dobb's Journal* (DDJ) and Ambysoft surveys are posted at http://www.ambysoft.com/surveys/, including the original source data, questions as they were asked, as well as slide decks summarizing Scott Ambler's analysis.
- People first. Alistair Cockburn paper, "Characterizing people as nonlinear, first-order components in software development" at http://alistair.cockburn.us/Characterizing+people+as+non-linear%2c+first-order+components+in+software+development argues that people are the primary determinant of success on IT projects. In "Generalizing Specialists: Improving Your IT Skills" at http://www.agilemodeling.com/essays/generalizingSpecialists.htm Scott argues for the need to move away from building teams of overly specialized people.
- **The Agile Scaling Model (ASM):** The ASM is described in detail in the IBM white paper "The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments" at ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF
- **Lean:** For more information about lean software development, Mary and Tom Poppendieck's *Implementing Lean Software Development: From Concept to Cash* (Addison Wesley, 2007) is the best place to start.
- **Hybrid processes:** In "SDLC 3.0: Beyond a Tacit Understanding of Agile" (Fourth Medium Press, 2010) Mark Kennaley summarizes the history of the software process movement and argues for the need for hybrid processes which combine the best ideas from the various process movements over the past few decades.

Additionally, financing solutions from IBM Global Financing can enable effective cash management, protection from technology obsolescence, improved total cost of ownership and return on investment. Also, our Global Asset Recovery Services help address environmental concerns with new, more energy-efficient solutions. For more information on IBM Global Financing, visit: **ibm.com**/financing

About the authors

Scott W. Ambler is Chief Methodologist for Agile and Lean with IBM Rational, working with IBM customers around the world to help them to improve their software processes. In addition to Disciplined Agile Delivery (DAD), he is the founder of the Agile Modeling (AM), Agile Data (AD), Agile Unified Process (AUP), and Enterprise Unified Process (EUP) methodologies and creator of the Agile Scaling Model (ASM). Scott is the (co-) author of 19 books, including *Refactoring Databases*, *Agile Modeling*, *Agile Database Techniques*, *The Object Primer 3rd Edition*, and *The Enterprise Unified Process*. Scott is a senior contributing editor with *Dr. Dobb's Journal*. His personal home page is www.ambysoft.com

Mark Lines cofounded UPMentors in 2007. He is a "disciplined" agile coach helping organizations improve on all aspects of software development. He is passionate about reducing the huge waste found in many IT organizations, and demonstrates hands-on approaches to speeding execution and improving quality with agile and lean techniques. Mark is a frequent speaker and writes for many software publications. His company website is www.UPMentors.com. Mark can be reached at Mark@UPMentors.com

[5] Disciplined agile teams strive to reduce the level of technical debt in your enterprise by adopting the philosophy of mature campers and hikers around the world—Leave it better than how you found it.

[6] We apologize for the application of "management speak", but organizational ecosystem is an accurate term.

[1] We apologize for the confusion of using the same term for two different but related concepts—disciplined agile delivery (all lower case) to refer to the category and Disciplined Agile Delivery (all upper case) to refer to the process framework.

[2] Granted, in this version we're using the term iteration instead of sprint, and work items instead of product backlog.

[3] For those of you familiar with UP we've adopted its phase names, although we've combined the UP's Elaboration and Construction phases. We've kept the fundamental focus of Elaboration, which is to explore and then prove the architecture with working code, in the form of the proven architecture milestone. Removing Elaboration as a separate phase helps streamline the DAD life cycle.

[4] Results of a 2009 Ambysoft survey.

RAW14261-USEN-00