Cutter
IT Journal

The Journal of
Information Technology Management

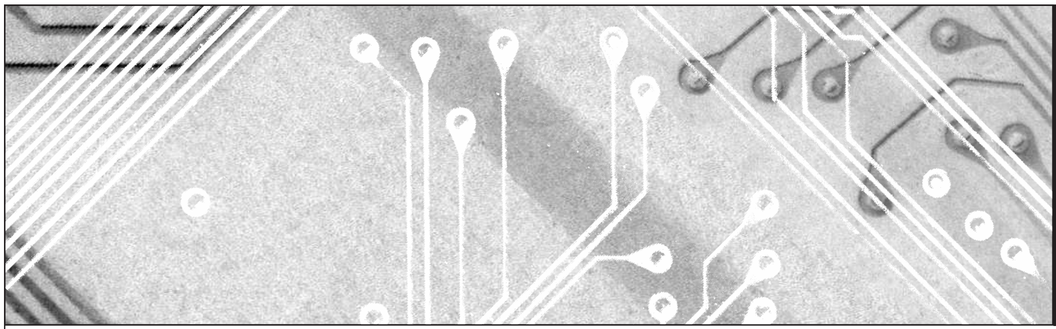
Vol. 26, No. 11
November 2013

Disciplined Agile Delivery:
The Foundation for Scaling Agile

"If you want to create an Agile organization, you can't rely on stacking the deck with your best staff members — *everyone* needs to make the transition, not just your star players."

— Scott W. Ambler,
Guest Editor

CUTTER
CONSORTIUM



Cutter IT Journal

About Cutter IT Journal

Part of Cutter Consortium's mission is to foster debate and dialogue on the business technology issues challenging enterprises today, helping organizations leverage IT for competitive advantage and business success. Cutter's philosophy is that most of the issues that managers face are complex enough to merit examination that goes beyond simple pronouncements. Founded in 1987 as *American Programmer* by Ed Yourdon, *Cutter IT Journal* is one of Cutter's key venues for debate.

The monthly *Cutter IT Journal* and its companion *Cutter IT Advisor* offer a variety of perspectives on the issues you're dealing with today. Armed with opinion, data, and advice, you'll be able to make the best decisions, employ the best practices, and choose the right strategies for your organization.

Unlike academic journals, *Cutter IT Journal* doesn't water down or delay its coverage of timely issues with lengthy peer reviews. Each month, our expert Guest Editor delivers articles by internationally known IT practitioners that include case studies, research findings, and experience-based opinion on the IT topics enterprises face today — not issues you were dealing with six months ago, or those that are so esoteric you might not ever need to learn from others' experiences. No other journal brings together so many cutting-edge thinkers or lets them speak so bluntly.

Cutter IT Journal subscribers consider the *Journal* a "consultancy in print" and liken each month's issue to the impassioned debates they participate in at the end of a day at a conference.

Every facet of IT — application integration, security, portfolio management, and testing, to name a few — plays a role in the success or failure of your organization's IT efforts. Only *Cutter IT Journal* and *Cutter IT Advisor* deliver a comprehensive treatment of these critical issues and help you make informed decisions about the strategies that can improve IT's performance.

Cutter IT Journal is unique in that it is written by IT professionals — people like you who face the same challenges and are under the same pressures to get the job done. *Cutter IT Journal* brings you frank, honest accounts of what works, what doesn't, and why.

Put your IT concerns in a business context. Discover the best ways to pitch new ideas to executive management. Ensure the success of your IT organization in an economy that encourages outsourcing and intense international competition. Avoid the common pitfalls and work smarter while under tighter constraints. You'll learn how to do all this and more when you subscribe to *Cutter IT Journal*.

Cutter IT Journal®

Cutter Business Technology Council: Rob Austin, Ron Blitstein, Tom DeMarco, Lynne Ellyn, Israel Gat, Vince Kellen, Tim Lister, Lou Mazzucchelli, Ken Orr, and Robert D. Scott

Editor Emeritus: Ed Yourdon
Publisher: Karen Fine Coburn
Group Publisher: Chris Generali
Managing Editor: Karen Pasley
Production Editor: Linda M. Dias
Client Services: service@cutter.com

Cutter IT Journal® is published 12 times a year by Cutter Information LLC, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA (Tel: +1 781 648 8700; Fax: +1 781 648 8707; Email: citjeditorial@cutter.com; Website: www.cutter.com; Twitter: @cuttertweets; Facebook: Cutter Consortium). Print ISSN: 1522-7383; online/electronic ISSN: 1554-5946.

☐ Start my print subscription to *Cutter IT Journal* ($485/year; US $585 outside North America)

Name	Title	
Company	Address	
City	State/Province	ZIP/Postal Code
Email (Be sure to include for weekly *Cutter IT Advisor*)		

Fax to +1 781 648 8707, call +1 781 648 8700, or send email to service@cutter.com. Mail to Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA.

SUBSCRIBE TODAY

Request Online License Subscription Rates

For subscription rates for online licenses, contact us at **sales@cutter.com** or **+1 781 648 8700.**

Opening Statement

by Scott W. Ambler, Guest Editor

DISCIPLINED AGILE DELIVERY: THE FOUNDATION FOR SCALING AGILE

This issue of *Cutter IT Journal* is a follow-up to June 2013's "Disciplined Agile Delivery in the Enterprise" issue. That edition of the journal covered strategies for bringing greater discipline to Agile software delivery teams, while this one focuses on how to scale disciplined Agile approaches. The term "scaling Agile" has at least two distinct definitions, both of which make complete sense. One vision focuses on adopting Agile techniques across an entire IT organization and ultimately instilling Agile behavior in the enterprise as a whole. The second vision focuses on how to apply Agile techniques in complex situations, in geographically distributed teams, or in regulatory regimes. Some people mistakenly believe these situations are outside the purview of Agile, but that's clearly not true in practice. In this issue, we explore these visions and show that both are key aspects to truly scaling Agile.

Scaling Agile Across the Enterprise

The first vision for scaling Agile concerns how to adopt Agile strategies across an entire IT department and eventually throughout the organization as a whole. This requires you to adopt Agile delivery techniques in most if not all development teams. This is much more challenging than simply cherry-picking "Agile friendly" teams, because if you want to create an Agile organization, you can't rely on stacking the deck with your best staff members — *everyone* needs to make the transition, not just your star players. When scaling Agile to the IT department, you will need to adapt all IT activities to achieve true agility (see Figure 1). Your Agile transformation efforts will need to take into account how your enterprise architecture, portfolio management, operations, data management, and many other teams work together. Furthermore, your IT department is just one group within your overall organization, so a true Agile transformation will require new behaviors within the business, too.

When helping to transform organizations to become Agile, my associates and I use something we call the P^3 Change Framework. The idea is that when making a transformation, you need to consider people, process, and product factors — a slight rewording of the people, processes, and tools strategy from the 1980s. Each of these change factors has subfactors:

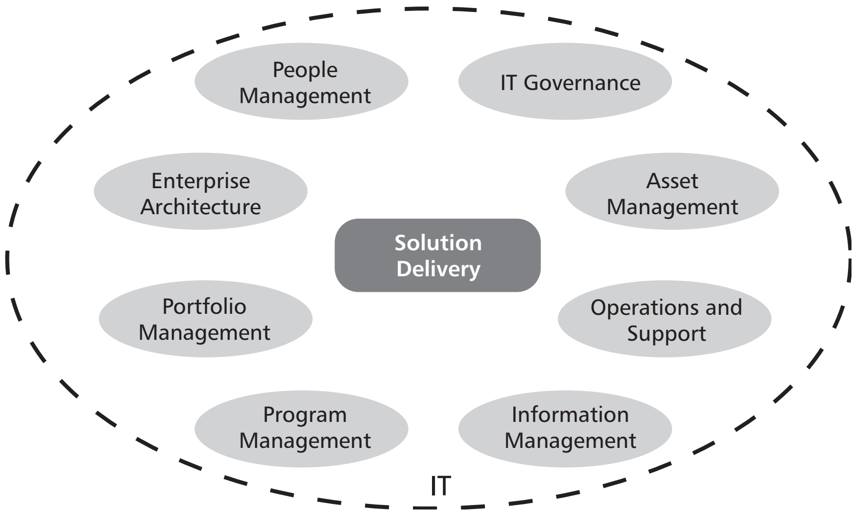


Figure 1 — Activities within an IT organization.

- **People** encompasses skills, mindset, and culture.
- **Process** encompasses practices, principles, and lifecycles.
- **Product** encompasses both tools and platforms.

Although they use slightly different terminology at times, in the two articles I've chosen for this topic, you will see how the authors found the need to address all three change factors in some way.

Terminology aside, we as an industry are sharing similar learnings from different sources, and to me that's a healthy sign of progress within the IT community.

The first of these two articles is written by Alan W. Brown, author of *Global Software Delivery: Bringing Efficiency and Agility to the Enterprise*, former IBM Rational CTO for Europe, and now a professor at the Surrey Business School in the UK. In the article, Brown argues that enterprises require greater openness and agility to be more innovative in the marketplace. He describes how the Disciplined Agile Delivery (DAD) framework provides the most realistic approach to Agile software development for enterprise-class environments. However, he warns that organizational culture and inertia will choke off the benefits of Agile unless you specifically deal with them. The article focuses on how to build an Agile organization, exploring the ways the technology, processes, and people will be affected. Regarding the people factor, Brown's experience is that you need to get the right people, promote greater collaboration between them, align responsibility and accountability, and then strive to *steer* instead of control (a key plank in DAD's approach to governance). On the technology/product side, Brown recommends that organizations adopt integrated development tools that support a business-led continuous delivery approach. Finally, he advocates such process improvements as moving from a development to a delivery focus, adopting Agile management techniques, promoting validated learning strategies, embracing innovation accounting, and adopting build-measure-learn cycles.

In our second "enterprise scaling" article, consultant Peter Herzum shares his experiences bringing Agile practices and software lifecycle automation into all of the development teams at Wolfe.com, a leader in the gift card and online prepaid domain. This Agile development strategy enabled the business itself to innovate, grow, and rapidly respond to change — in other words, to become an Agile enterprise. This article isn't specifically about the adoption of a DAD-based approach, but almost all of the Agile practices that Herzum applied at Wolfe are encapsulated within the DAD framework. These strategies include adopting a parallel independent test team to support the "standard" testing strategies employed within the delivery teams, adopting a full delivery lifecycle that explicitly addresses architecture, transition/release practices, Kanban-based strategies, and development intelligence, to name a few. While Herzum calls this "Agile 2.0," my colleagues and I have called it "Disciplined Agile." Terminology aside, we as an industry are sharing similar learnings from different sources, and to me that's a healthy sign of progress within the IT community.

Scaling Agile Delivery for Complex Situations

The second vision for scaling Agile involves ways you can tailor Agile strategies within teams to address the complexities they face "at scale" in situations that require more than a single, small, colocated team. Many people relate this version of scaling to large teams, but there's more to it than that. As Mark Lines and I describe in our article, Agile approaches are also being applied by geographically distributed teams, by teams in compliance situations, by teams that are organizationally distributed (think outsourcing), and in situations where there is significant domain complexity or technical complexity. DAD's process goal–driven strategy provides the guidance that teams require to adapt their approach to the context that they find themselves in, enabling teams to work at scale.

All IT delivery teams are governed in some way, including Agile ones. The DAD framework provides explicit advice for governing Agile teams effectively, because many organizations run into trouble with Agile when they mistakenly apply traditional governance strategies.

In fact, not updating your IT governance approach to reflect the realities of Agile solution delivery may be one of the leading causes of failed Agile adoption programs. In our fourth article, University of Bolzano researchers Saulius Astromskis, Andrea Janes, Alberto Sillitti, and Giancarlo Succi describe in detail how to enhance your governance efforts through automation. The authors focus on how to noninvasively measure what is occurring in Agile teams so that you can then identify potential improvements in your process. This sort of noninvasive measurement is called development intelligence (DI) in the DAD framework, and it supports both process improvement efforts, as described in this article, as well as governance in general. DI is a strategy in which a project or portfolio dashboard is automatically populated from data generated by tool usage. This is an important technique for enabling teams to understand what they are actually doing as opposed to what they *believe* they are doing. When implemented fully, DI provides senior management with accurate insight regarding the results of the team's work and thus allows them to govern more effectively.

One interesting thing about this article is how the team worked with tooling from multiple sources, most of which were open source. They in effect showed how it's possible to implement DI without requiring a single-vendor application lifecycle management (ALM) tooling solution. Another interesting aspect is the authors' discussion of how to analyze the data so as to visualize the existing process, thereby enabling the team to identify potential bottlenecks that need to be addressed.

Last but certainly not least is the article by Mindtree's Raja Bavani. Bavani begins by discussing the particular challenges of distributed Agile delivery and then works through 10 principles that he has found to be critical to success in these situations:

1. Methodology is driven by project teams.
2. Consistent usage of common tools improves productivity.
3. Infrastructure for communication and coordination is crucial.
4. Knowledge management is key to success.
5. Quality is multidimensional and owned by everybody.
6. Distributed Agile requires an inclusive approach.
7. Governance is the backbone of successful distributed teams.
8. Automation enables sustainable pace.
9. It is essential to streamline the payoff of technical debt.
10. Ensuring early success is a collective responsibility.

I believe that you will find this issue of Cutter IT Journal to be very informative. It is not only possible to scale Agile approaches, it is highly desirable. Enjoy!

Scott W. Ambler is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture *and* Agile Product & Project Management *practices. He is the thought leader behind the Disciplined Agile Delivery (DAD) process decision framework, Agile Model Driven Development (AMDD), the Agile Data (AD) method, and the Enterprise Unified Process (EUP), and he works with clients around the world to improve the way they develop software.*

Mr. Ambler is coauthor of several software development books, including Disciplined Agile Delivery, Agile Modeling, The Elements of UML 2.0 Style, Agile Database Techniques, *and* The Enterprise Unified Process. *He is also a Senior Contributing Editor with* Dr. Dobb's Journal. *Mr. Ambler has spoken at a wide variety of international conferences, including* Agile 20XX, Software Development, IBM Innovate, Java Expo, *and* Application Development. *He can be reached at sambler@cutter.com.*

Toward the Agile Organization: Accelerating Innovation in Software Delivery

by Alan W. Brown

THE INNOVATION CHALLENGE

Innovation is now seen as a business priority that is essential for success. For businesses around the world to have real hope of meaningful growth, the collaborative process from idea generation to solution delivery must be optimized, innovation practices enhanced to be flexible and repeatable, and leaders trained who are willing and able to lead teams in an innovation-focused interactive environment. In the past, innovation was slow and risky and left to the experts housed in the R&D department. Today, there is incredible importance placed on "democratizing innovation" by establishing practices that increase innovation speed while decreasing risk.[1]

To address these needs, the focus has moved toward two key areas: agility and openness. Consumers simultaneously require businesses to maintain a constant open dialogue to monitor feedback and reactions to product improvements and prototypes, and at the same time to engage in new forms of rapid producer-consumer partnerships such as co-creation, evolutionary design, and micro-customization. This approach helps to ensure that the business produces goods and services its customers require at the speed necessary to maintain a market advantage. Through a combination of agility and openness, the benefits of flexibility within an interactive market-driven conversation must be pursued within companies of all sizes to confront the challenges of our turbulent economic environment.[2] The goal is to deliver a rapid stream of consumer-tested ideas to position the company as a leader in the vital knowledge-driven marketplace.

However, the generation of new ideas is only the starting point. Success requires overcoming obstacles to bringing those innovations into routine practice. Illustrations of the perils of these innovation challenges abound. For example, it has been estimated that up to 80% of corporate innovations fail and only 10% of small to medium enterprises can sustain the innovation necessary to generate significant employment.[3] In particular, *speed* is of the essence in introducing innovation. Intel Corporation, for instance, claims that 90% of the revenues the firm derives on the last day of the year are attributable to products that did not even exist on the first day of that same year.[4]

Within the software world, this kind of rapid innovation has been the focus of Agile coding approaches such as Scrum, XP, and Crystal, and it has been extended into broader software delivery contexts in DSDM and blended waterfall-Agile methods (denoted as "wagile" and "water-scrum-fall"). However, it is in the Disciplined Agile Delivery (DAD) approach that we see perhaps the most comprehensive and realistic treatment of Agile software practices as they relate to the larger footprint typical of most commercial software delivery contexts.[5] The DAD process framework recognizes not only the importance of networks of cross-functional teams, it also explicitly offers support for scaling key practices across complex working environments using techniques that link software development efforts into robust software delivery contexts.

Beyond Agile software delivery, an even broader view of the Agile organization is necessary. Many businesses see their ambitions thwarted when their Agile software delivery teams become swamped by overbearing, slow-moving engineering and management practices. In any real project, substantial effort is invested in essential activities such as hiring staff, obtaining and setting up test equipment, interacting with project and program management coordinators, training sales teams on new capabilities, and so on. Without care, the gains from Agile and open software delivery become insignificant in the daily cut-and-thrust of a project, or else they are choked by the broader organizational inertia and inefficiency that surround them.[6]

FOCUS AREAS FOR THE AGILE ORGANIZATION

Let me reiterate: innovation at Internet speed is more than just generating new ideas — it is bringing new ideas into routine practice. Consequently, the focus for innovative organizations is always broader than is often assumed in Agile software development situations. Business success necessitates optimization of the collaborative process from idea generation to solution delivery. This is done by enhancing innovation practices so they are flexible and repeatable, creating organizations that adapt and respond in real time to the changes around them, and encouraging the team-working skills that are essential for overcoming obstacles inherent to any creative activity.

This ideal is referred to as the "Agile organization." It is a new way of operating based on a set of principles, practices, and tools that are emerging across a range of disciplines in systems engineering, IT, and solution delivery. It extends the principles of DAD and becomes the driving force for the new digital economy. Organizations unable to make progress toward this goal and increase capacity to innovate at Internet speed will not survive. Let's briefly examine the characteristics of an Agile organization across three familiar dimensions: technology, process, and people.

Technology

Many software-intensive businesses have struggled to transform their lifecycle approach from a *development* focus to a *delivery* focus to improve the speed and flexibility with which they operate. This subtle distinction in wording represents a dramatic change in the principles that are driving the management philosophy and the associated governance models.

As summarized in Table 1, the change in perspective from software development to software delivery affects several dimensions. A software delivery perspective focuses on the following concepts and practices:[7]

- **Continuously evolving systems.** Enterprise software undergoes a continual process of change. Traditionally, the goal is to optimize the development of the first release of the system, but the critical activities come *after* the first release. This evolution should be the focus of attention, and any Agile organization must invest in techniques that support and encourage software to evolve.
- **Blending of boundaries between development and maintenance.** Traditionally, a clear distinction is made between development and maintenance, with different teams responsible for these activities. In fact, in many cases these teams may even be in different buildings, on different continents, or managed by different companies. With an evolutionary view of software delivery, the distinction between these two activities is blurred to the point that development and maintenance are just two aspects of the same need to create and deliver value to users of the system.
- **Sequence of released capabilities with ever-increasing value.** A development view operates from the understanding that after a deep analysis, the requirements for a system are signed off and development of the system begins with the goal to "fulfill" those requirements and place the system into production. But the reality in many systems is that requirements emerge and evolve as more is discovered about the needs of the stakeholders and as

Table 1 — A Software Development Perspective vs. a Software Delivery Perspective

Software Development Perspective	Software Delivery Perspective
Distinct development phases	Continuously evolving systems
Distinct handoff from development team to maintenance team	Common process, platform, and team for development and maintenance
Distinct and sequential activities: requirements to design to code to test	Sequence of usable capabilities with ever-increasing value
Role-specific processes and tools	Collaborative platform of integrated, Web-based tools and practices
Colocated teams	Distributed, Web-based collaboration
Governance via measurement of artifact production and activity completion	Governance via measurement of incremental outcomes and progress/quality trends
Engineering discipline: track progress against static plans	Economic discipline: reduce uncertainties, manage variance, measure trends, adapt and steer

understanding of the delivery context grows. A more realistic approach to delivery views the system not as a number of major discrete releases, but as a continuous series of incremental enhancements with increasing value to the stakeholders.

- **Common platform of integrated process and tools.** The siloed delivery approach is usually supported by processes and tools that are optimized for each silo. This occurs because most organizations define processes and acquire the tools individually, function by function, with little thought for the end-to-end flow of information and artifacts. A delivery view recognizes that the interoperation of these processes and tools is vital for optimizing enterprise system value.

Too many people associate an Agile organization with a chaotic, "anything goes" attitude to governance and planning. Nothing could be further from the truth.

- **Distributed Web-based collaboration.** Teaming and teamwork are a focus within functional areas for a development approach. A delivery view defines the team more broadly, recognizing that stakeholders in software delivery may vary widely in function, geography, and organization. Technology support to include all those stakeholders in team activities is essential. While a variety of collaborative, Web-based technologies have emerged in recent years, many organizations have deployed them in an ad hoc way and invested little in any concerted approach to adopt them across their enterprise software delivery organization.

- **Economic governance tailored to risk/reward profiles.** To manage software development, most organizations use a collection of processes, measures, and governance practices that emphasize development artifacts such as the software code, requirements documents, and test scripts. A delivery view moves the focus of governance toward the business value of what is being delivered, aiming to optimize features delivered and time-to-value of delivered capabilities, increase burndown of backlogs of new requests, reduce volatility of systems, and sustain velocity of delivery teams.

- **Measurement based on business value and outcome.** Many enterprise software development organizations have prided themselves on their technical skills and the depth of their knowledge in technologies for software development and operations. These are vital to success. However, at times this emphasis on development has created the perception in the broader organization that the IT organization is constantly in search of the latest technology solution with little regard for where and how such technology investments help the business achieve its goals. A focus on software delivery gives greater emphasis to the business value of those investments and creates a more balanced view of investment.

This change in thinking radically alters the way software-intensive businesses approach their task. A delivery perspective encourages styles of software delivery that move away from early lock-down of decisions to reduce variance in software projects toward controlled discovery, experimentation, and innovation.

Process

Most software-intensive businesses focus the majority of their activities toward core processes aimed at efficiently controlling the management, upgrade, and repair of their existing systems. Typically, up to 80% of all costs are consumed in such tasks.[8] Consequently, an Agile organization must ground its work in approaches that recognize and deliver efficiency in software development and delivery. For most organizations, Lean thinking is the basis for current efficiency approaches.[9]

Innovation in software delivery is driving a number of important advances in Lean thinking as it applies to software-intensive businesses.[10] In the move toward an Agile organization, we see a focus on Lean practices with the aim of ensuring fast cycles to improve feedback and learning across the value chain. The result is an Agile innovation practice viewed as a series of experiments governed by well-defined hypotheses, a focus on the speed of testing those hypotheses, and recognition that a clear approach to measurement and management is essential to ensure that any experimental approach converges toward meaningful decision making. Some key principles are emerging, which have been well summarized by entrepreneur Eric Ries as:[11]

- **Agility is management**. Too many people associate an Agile organization with a chaotic, "anything goes" attitude to governance and planning. Nothing could be further from the truth. In any Agile organization, there must be a very strong management dimension to ensure progress is coordinated and aligned. Large numbers of small incremental changes must be made based on analytical data. That cannot occur without discipline and rigor.

- **Validated learning**. Agile decision making requires constant feedback and is consolidated through frequent reflection. An Agile organization must learn from the experiments it undertakes and validate that learning with real-world inputs.
- **Innovation accounting**. Measuring progress in conventional ways (source lines of code delivered, function points coded, etc.) offers little value to an Agile organization. Rather, the organization is optimized for rapid decision making, flexibility in adopting new capabilities, and easy adaptation to evolving feedback from early consumers. Consequently, innovation accounting focuses on establishing benchmarks for these areas and accelerating the pace of delivery based on managing those measures.
- **Build-measure-learn cycle**. The most critical lifecycle process in an Agile organization is the "build-measure-learn" cycle, and a great deal of attention is directed toward its definition and execution. Optimizing speed through this cycle ensures that everyone involved in the organization can move quickly toward better decision making based on data from the impact of earlier actions.

In fact, Ries goes further when analyzing successful software-intensive businesses that are broadly adopting an Agile approach. He observes that what distinguishes them from other organizations is they welcome and embrace change, making many adjustments in approach as they learn what works and what doesn't — changes in value proposition, customer segment, business model, partner network, and so on. Their key attribute is their ability to *pivot* when they gain feedback that is contrary to their expectations. They change directions but stay grounded in what they have learned. Furthermore, they focus on validated learning and employ a rigorous method for demonstrating progress through positive improvements in core metrics and key performance indicators (KPIs) critical to the software-intensive business.

People

Many recent studies of high-performing software-intensive businesses have highlighted the dominant role played by people and team dynamics in any software delivery success.[12] In spite of these studies, many businesses fail to take adequate account of the human elements of an improvement program and focus a majority of their attention on the more mechanical technology and process aspects.

In many regards, the Agile Manifesto[13] issued over a decade ago was explicitly aimed at placing the focus of attention on the impact of people in software-intensive businesses. Several Agile software development methods provide clear guidance on how to motivate teams to encourage greater innovation,[14] with Agile author and consultant Jim Highsmith perhaps offering the most straightforward advice on creating and motivating self-directed Agile teams:[15]

- Get the right people
- Clearly articulate the project vision, boundaries, and roles
- Encourage interaction
- Facilitate participatory decisions
- Insist on accountability
- Steer, don't control

The key attribute of Agile teams is their ability to pivot when they gain feedback that is contrary to their expectations.

Such simple guidelines present an obvious starting point for Agile organizations to improve their organizational capabilities. However, their interpretation and implementation can be much more challenging in practice if organizations don't also adopt radical management principles tuned to rapid decision making.[16] Individuals and teams operate within a broader organizational culture and context. Management consultant Steve Denning views the drive for greater organizational agility as a major challenge to traditional management practices and believes that it requires quite different approaches to how organizations set goals, support managers in achieving them, coordinate the organizational supply chain, set incentives for individuals and teams, and communicate paradigms within the organization and across the extended partner ecosystem. His view is that top-down autocratic management styles must be replaced by more radical styles based on meritocracy and shared responsibilities across team members.

At the organizational level, Agile organizations must constantly adapt to meet the demands of continuously changing business environments. Although it is tempting to believe that such adaptation can be driven through high-level corporate initiatives, the most common approach in Agile delivery involves building the visibility and credibility of leaders throughout the organization and to encourage communities led by those elected by the community itself (so-called “meritocracies”). In this way, an Agile organization builds an environment that encourages action from the bottom up and naturally develops employees who embrace and share new ways of thinking and working.

Moving an organization to an Agile mindset requires a focus on a small number of very practical areas where demonstrable, measured progress can be made.

In fact, a recent CIO-level study[17] emphasized six key findings that underscore the importance of promoting an effective organizational context for encouraging Agile thinking in individuals and teams:

1. Organizations require significant time to absorb changes and react to new norms. Typically it takes more than two years for major organizational shifts to be accepted. Attempting more rapid change in organizational structures frequently has negative impacts that overwhelm the improvements they introduce. Even as organizations work to achieve rapid innovation, they must recognize that any fundamental change cannot be implemented in a single step but will instead require a series of measured increments to become routine.
2. Where businesses are experiencing fast-paced changes in their environment, their employees’ ability to adapt to change is more significant to success than their productivity and performance. Not only do these adaptable employees enhance their own impact on the organization, but they also learn from others, seek feedback, and support their peers.
3. Agile organizations must improve communication, increase transparency, and focus on sharing information that helps employees be self-directed, be autonomous, and develop their own solutions to problems they encounter.
4. The highest-performing individuals and teams in an Agile organization create strong networks and use these networks as a primary source of communication, problem solving, and social interaction.
5. The enablement role assumed in most software-driven businesses must be revisited for Agile organizations. They must prioritize coaching, connect employees to the right networks and communities, and emphasize honest retrospection and feedback to improve all aspects of the organization.
6. The focus on agility is a cross-functional effort. It affects all aspects of the organization, including its approach to employee hiring and training, performance management models, external communications and public relations, strategic planning, and technology investment.

SUMMING UP

Agile thinking has galvanized several important ideas essential to driving innovation in software-intensive businesses. Here, two aspects have been highlighted: accelerated innovation practices focused on the rapid introduction, experimentation, and evaluation of new ideas and an Agile software delivery perspective on how software is designed, created, and placed into production.

The biggest challenge facing an Agile organization concerns a shift in thinking from top-down control, complex preplanning of all actions, and internally focused metrics for measuring progress to team-based meritocracies, flexible priority-based planning, and outcome-based metrics. For many organizations, this change is simple in theory, but very difficult to execute in practice due to cultural, financial, and operational inertia. Fundamental characteristics of an organization must be managed in realizing this change, including such scaling factors as the organization’s size, geographic distribution, exposure to external regulatory oversight, and so on. As a result, moving an organization to an Agile mindset requires a focus on a small number of very practical areas where demonstrable, measured progress can be made. Building on the DAD framework, this article has introduced the change in thinking required in software-intensive businesses and highlighted the essential areas of focus for succeeding with accelerating innovation in software delivery.

ENDNOTES

[1]von Hippel, Eric. *Democratizing Innovation*. MIT Press, 2005.

[2]Prahalad, C.K., and M.S. Krishnan. *The New Age of Innovation: Driving Co-Created Value through Global Networks*. McGraw-Hill, 2008.

[3]Levie, Jonathan, and Mark Hart. "Global Entrepreneurship Monitor: UK 2011 Monitoring Report." University of Strathclyde, 2011.

[4]Augustine, Norman R. *Is America Falling Off the Flat Earth?* National Academies Press, 2007.

[5]Ambler, Scott W., and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Solution Delivery in the Enterprise*. IBM Press, 2012.

[6]Brown, Alan W. *Global Software Delivery: Bringing Efficiency and Agility to the Enterprise*. Addison-Wesley Professional, 2012.

[7]Royce, Walker, Kurt Bittner, and Mike Perrow. *The Economics of Iterative Software Development: Steering Toward Better Business Results*. Addison-Wesley Professional, 2009.

[8]Jones, Capers. *Software Engineering Best Practices*. McGraw-Hill, 2010.

[9]Liker, Jeffrey. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. McGraw-Hill, 2004.

[10]Coplien, James O., and Gertrud Bjørnvig. *Lean Architecture for Software Development*. Wiley, 2011.

[11]Ries, Eric. *The Lean Startup: How Constant Innovation Creates Radically Successful Business*. Penguin Business, 2011.

[12]Curtis, Bill, William E. Hefley, and Sally A. Miller. *The People CMM: A Framework for Human Capital Management*. 2nd edition. Addison-Wesley Professional, 2009.

[13]Agile Manifesto (http://agilemanifesto.org).

[14]Cohn, Mike. *Succeeding with Agile: Software Development with Scrum*. Addison-Wesley Professional, 2010.

[15]Highsmith, Jim. *Agile Project Management: Creating Innovative Products*. 2nd edition. Addison-Wesley Professional, 2009.

[16]Denning, Stephen. *The Leader's Guide to Radical Management: Reinventing the Workplace for the 21st Century*. Jossey-Bass, 2010.

[17]"Building a Change-Ready IT Organization." Corporate Executive Board, September 2012.

Alan W. Brown is Professor of Entrepreneurship and Innovation in the Surrey Business School, University of Surrey (UK), where he leads activities in the area of corporate entrepreneurship and open innovation models. In addition to his teaching activities, Dr. Brown focuses on innovation in a number of practical research areas with regard to global enterprise software delivery, Agile software supply chains, and the investigation of "open commercial" software delivery models. He has formerly held a wide range of roles in industry, including Distinguished Engineer and CTO at IBM Rational, VP of Research at Sterling Software, Research Manager at Texas Instruments Software, and Head of Business Development in a Silicon Valley startup. In these roles, Dr. Brown has worked with teams around the world on software engineering strategy, process improvement, and the transition to Agile delivery approaches. He has published over 50 papers and written four books. He holds a PhD in computing science from the University of Newcastle upon Tyne (UK). He can be reached at alan.w.brown@surrey.ac.uk.

A CIO/CTO View on Adopting Agile Within an Enterprise

by Peter Herzum

From April 2011 to December 2012, I was in charge (initially as executive consultant and later as CTO) of the IT organization of Wolfe, LLC, the rapidly growing corporate owner of Giftcards.com and other brands in the online prepaid domain.

My consulting group, Herzum, Inc., was hired to introduce project-level Agile best practices (the COSM approach[1]) and software lifecycle automation (a set of Atlassian[2] products) to all of the company's software development teams. After the initial phase, the engagement evolved to supporting the group in the adoption of a disciplined, scalable, agile approach to IT that would allow the business to rapidly innovate and grow and enable IT to quickly respond to business changes. In this article, I describe the roadmap and the progression of best practices introduced, emphasizing activities that are common across agility-in-the-large transformations. (Note: In this article, I will use the lower-case term "agile" in its dictionary sense, and the upper-case term "Agile" when referring to best practices based on the Agile Manifesto [Scrums, sprints, user stories, test-driven development, continuous integration, etc.].)

INITIAL GOAL: PROJECT-LEVEL AGILE BEST PRACTICES

Initially, the company engaged Herzum to provide support for:

- Adopting specific tools (by Atlassian) to automate the software development and release lifecycles
- Becoming "Agile" — helping them select appropriate Agile best practices and coaching them in the efficient adoption of those practices
- Configuring and extending the tools to conform to the specific Agile practices selected

This initial set of activities became Phase 0 of the agile transformation of the IT organization.

At the time (April 2011), the development team was composed of 12 highly skilled developers who had extensive experience in developing e-commerce solutions based on the LAMP (Linux Apache MySQL PHP) stack but little Agile experience. The development organization was organized in a centralized development team that serviced seven different e-commerce business units.

PHASE 0: SPRINTS AND RELATED TOOL SUPPORT

Phase 0 lasted three months. In the first three weeks, we installed an initial configuration of the tools and provided Agile training and relevant tool training. Subsequently, we led (as Agile Masters) two pilot projects, began to provide overall support to other projects and activities, and progressively expanded Agile adoption to the whole development organization.

This was a typical Agile 1.0 adoption, consisting of focused sprints and related practices along with tools for project- and enhancement-based releases (releases based, week after week, on addressing "the next set of features and stories"). The phase ended when all development teams had adopted a tool-supported Agile approach to software development, using, for example, two-week release sprints (one week development/development testing, one week preproduction testing performed by a test team), story-based development, story points, Scrum/sprint boards for daily/weekly project tracking, and burndown-burnup charts.

KEY POINT

Some Agile experts advocate the elimination of separate testing teams. In my experience, high-availability e-commerce sites strongly benefit from having such teams. But all organizations — including those with a separate testing team — benefit from pushing as much testing and quality assurance as possible to the development team, making it co-responsible for the quality of their deliverables.

Phase 0 significantly shortened the time to market for all business units, created a focused weekly cadence for all activities, and enabled a weekly release-to-production process for each business unit. It also provided the basis for a disciplined, measured approach to projects and releases.

The combination of practices *and* tool support was critical during this phase. Tools included a best-in-class issue management system (JIRA), an Agile project management tool (GreenHopper, recently renamed JIRA Agile), and a wiki-based knowledge management tool (Confluence). These tools allowed company-wide visibility of all projects, which was critical for satisfying the compliance requirements of the prepaid industry.

Goals Revised: Roadmap for an Agile IT

During Phase 0, the Herzum team performed an assessment of the entire IT organization and provided short-term, medium-term, and long-term recommendations. The assessment was based on the COSM agile enterprise maturity model. Key dimensions of this model include project-level processes (project management, requirements, development, release, QA), but also enterprise-level elements, including portfolio management, IT management, enterprise architecture, software automation, collaboration, social/mobile (common to all e-commerce companies), data warehouse/business intelligence, IT management, IT organization, and IT operations. COSM takes the view that agility-in-the-large cannot be achieved without combining project-level practices with strategy-focused disciplines, adaptive architectures, and the appropriate software factory[3] tools.

The organization was experiencing significant growth. The business units (and related online applications) had grown organically and/or by acquisition, resulting in online applications that had been rapidly developed with a time-to-market emphasis. There were technical and architectural redundancies (for example, multiple shopping carts). Overall, the group was the victim of its own success, as demonstrated by the fact that the number of daily visitors had grown rapidly, requiring the online applications to provide higher levels of quality, scalability, performance, availability, and security than originally expected. The majority of the applications were due for a revamp. At the same time, in order to maintain Wolfe's leadership in the prepaid online market, there was a need to continue to rapidly implement new features.

The objectives thus shifted from adopting "basic" Agile best practices to establishing an agile IT function — one managed with the discipline required to contribute to the success of the business, effectively respond to the rapidly changing needs of the business, and scale to a growing set of teams and a growing business. This also required transformation to a set of architecturally sound, reuse-based applications that would reduce maintenance costs, address production quality issues, and much more.

Many businesses have a need for discipline that is a prerequisite for achieving explicit business objectives. For example, in the prepaid cards domain, compliance with the Level 1 Payment Card Industry Data Security Standard (PCI DSS) requires a detailed level of traceability and documented change management. The dichotomy between required discipline and agility is a common challenge of modern enterprises.

To address these many objectives, we defined an IT roadmap organized in three phases:

- **Phase 1 — Consolidation and Innovation.** This phase was designed to enable specific business, functional, and technical objectives. It involved a sweeping revamp of existing applications and consolidation of new Agile practices, as well as innovation through adoption of new business solutions, new technologies, and new architectural approaches.
- **Phase 2 — Fit for Growth.** This set of initiatives not only addressed the ongoing significant business growth (in terms of volumes and visitors) and other changes, but also the explosive growth planned for 2013.
- **Phase 3 — Sustainable Growth.**

I will only focus on those elements that are relevant for our "agility" topic. A simplified summary of the phases is presented in Table 1.

PHASE 1: CONSOLIDATION AND INNOVATION

Phase 1 lasted six months (until the end of 2011). As part of the consolidation, we addressed a new set of best practices, but also organizations, architectures, technologies, and automation (the "software factory," in COSM terms):

- **Agile 2.0.** We expanded the adoption of agile (not only Agile) best practices and supporting tools to include feature-driven management, integrated QA (testing and reviews) and release management (tool-supported continuous assembly and integration, automated release management), customer support, IT operations, and advanced project management.
- **"Factory 2.0."** The above practices were supported and automated by an integrated toolset that included tools for continuous integration and continuous delivery (Bamboo), code repository management (FishEye), review management (Crucible), and test-case management (Herzum Test Manager).

Table 1 — IT Roadmap by Phase

	Phase 0	Phase 1	Phase 2	Phase 3
Focus	Sprints and Related Tool Support	Consolidation and Innovation	Fit for Growth	Sustainable Growth
Duration	3 months	6 months	12 months	12 months
Project Processes	Agile 1.0: Weekly sprints, story-based	Agile 2.0: Continuous integration and delivery, integrated QA, Kanban for operations/customer services	Business project management, QA to include business QA, cosourcing	Optimizations and evolutions
Integrated IT Processes		IT operations, help desk, IT continuity	Strategy-focused IT, project/application portfolios, financial management	
Main Factory/ Tools Focus	Issue management, collaboration	Automated release management, QA (test case management, reviews)	IT dashboards, QA automation, proactive IT reporting	Business dashboards, proactive business reporting, cosourcing
Main Architecture Focus	Project architecture	Foundational services such as security, content management, and more	Core component/ applications, data architecture	Enterprise architecture

- **Component-based, service-oriented, reuse-centric architectures.** We established adaptive architectures, simplifying the overall application portfolio and establishing a reusable foundational set of services, as well as introducing a content management system for the 70% of the Web pages that were mostly "content."
- **Lifecycle automation and collaboration ("software factory").** We extended the automation and collaboration tools to the whole IT function and beyond.
- **Organizations.** The IT organization doubled in size and, in addition, started to make use of external teams and contractors. Existing and new agile practices, architectures, and technologies had to rapidly scale and be easily adoptable by new teams.

At the end of Phase 1, the whole of IT and most business teams had been involved in the agile transformation. At this time, "agility" included the following capabilities:

- **Business collaboration.** High levels of collaboration between IT and business teams enabled the company to rapidly define projects and provide tracking and transparency in terms of what would be released and when.
- **Integrated content changes.** These allowed the marketing team, and the business teams in general, to perform daily controlled releases for Web content changes in a controlled environment without involving IT.
- **Continuous delivery.** We achieved 95% automation of the release process from development to test, preproduction, and release to production. This removed most defects due to the release process, improved schedule predictability, and produced a high speed of release. We could, for example, release patches in a disciplined, controlled process in *minutes* and major high-quality releases on a weekly basis.
- **Integrated Kanban for IT operations and customer support.** Tool-supported Kanban allowed for easy and traceable escalation to development and directly integrated with the sprint-based application releases (when relevant).
- **Transparent IT.** The adopted best practices provided a transparent "fish bowl" approach to the whole software supply chain, from project inception to release to maintenance and operations. Both business and IT representatives could access collaboration tools and directly verify the status of each project and each detailed requirement/bug being addressed by each release. They could also drill down to specific activities of each resource and even specific code being written by individual developers.

- **Proactive reporting.** This included automatic email notification to interested parties of releases and scheduled generation (and email) of management reports.
- **Complete, metrics-based tracking.** From requirements to development to testing to release, complete metrics-based tracking was another new capability. Each requirement was automatically linked to the tests and the software code produced. This also hugely simplified compliance.

This new approach to the software supply chain, which favored transparency of development and releases, high levels of automation, and a highly collaborative environment, contributed to the group's being voted among the best places to work in the *Pittsburgh Post-Gazette*'s "Top Places to Work" poll in 2011.

The innovation part of the transformation focused on the rapid introduction of new features and initiatives, such as social login, gamification, and new search engine optimization (SEO) strategies, all of which were implemented in a matter of weeks.

Phase 1 created the prerequisites for not only an agile IT, but — more importantly — an agile business. Indeed, business benefits from Phase 1 included:

- **Market penetration.** We experienced acceleration of our market penetration through new offerings (year-to-year increases of 40%-60% in volumes), advanced techniques to improve our SEO standings, improved customer retention and conversion rates (more than double), and much more.
- **Time-to-market speed.** We could now bring new features to market within weeks of conception (at times, within days) and with significant quality.
- **Reduced software development costs.** We saw significant reduction of development and maintenance costs for our e-commerce applications.
- **Traceability, quality, and more.** The adoption of tools for unit test and application test (both test specification and execution), the extension of QA practices (both reviews and testing) to the various phases of the software supply chain, and support for low-granularity levels of traceability assured a high level of quality in our software. We also significantly improved our uptime and production performance.

PHASE 2: FIT FOR GROWTH

Phase 2 lasted 12 months (all of 2012). Most of Phase 2 addressed the introduction of new business initiatives (such as the introduction of new giftcard products, a mobile offering, and vastly improved search capabilities) and new architectural solutions (significant performance improvements, a total upgrade of the security architecture, a redesigned data architecture, etc.). The IT team had grown to 35 people. The time dedicated to IT transformation was limited, so we had to pick our priorities carefully.

The time dedicated to IT transformation was limited, so we had to pick our priorities carefully.

The key objectives were to allow IT to respond to the fast pace of the business, addressing the required extra-functional requirements (availability, scalability, performance, security, etc.), and preparing the company for the planned exponential business growth. The objectives had more to do with supporting business innovation and growth than with "Agility."

Phase 2 was driven by an aggressive set of idealized IT targets, such as "0% downtime," "100% release-to-production automation," and "0% QA-identifiable production defects." Achieving these goals required us to revamp the organization and the applications and introduce several IT best practices in conjunction with the next level of Agile practices. For this article, I will focus on the subset of Phase 2 activities related to agility.

The Phase 1 tools and approaches were extended to all parts of the business, including customer service, fulfillment, marketing, sales, and business operations. In addition, we explicitly focused on:

- **Portfolio management.** We implemented Agile solutions for project portfolio and application portfolio management and for IT management in general. This made project creation and cross-project reporting easy and enabled cross-project integration and the development of customizable dashboards.
- **IT financial management.** We integrated the budgeting and financial tracking of projects in the overall processes.

- **Technologies refresh.** We adopted newer technologies and solutions that would simplify development and maintenance. For example, we replaced the Zend framework with Symfony, and the Subversion version control system with Git, and further optimized our release management and change tracking.
- **Cosourcing.** Adopting a transparent cosourcing model allowed for flexible growth of the IT resourcing model in a period of accelerated development.
- **Architecture 2.0 — core.** We redesigned a set of core components for reuse and scalability. This was the most important element in achieving new levels of agility.
- **Improved collaboration tools.** We further expanded our collaboration tools (e.g., adopting HipChat), expanding and optimizing support for continuous delivery and collaboration.

Agile best practices came and went. What stayed were the new integrated disciplines, the architectures, and the integrated collaboration tools that enabled responsiveness to the business.

At the end of Phase 2, so I could fully focus on my consulting group, I helped select a new CTO. My replacement and I overlapped for several months, and in that time it was fascinating (and a great lesson!) to see a new, experienced CTO take ownership of "my" team in a friendly, managed transition. Metrics and reports that I considered key were ignored. Technologies that I did not consider important became high priority. Practices that I would require every day stopped being used. And yet, IT continued to improve.

AGILITY IN THE ENTERPRISE

After a profound, company-wide, architecture-driven, tool-supported IT transformation, we had an agile IT organization. Apart from reaching at least Level 2 on *all* dimensions of the COSM agile maturity model, we could now claim:

- **Highly productive development teams.** We were now able to bring to market new features across multiple online business units on a weekly basis, and complete new solutions within four to eight weeks, in a fully traced/tracked, transparent set of development projects.
- **High levels of release and delivery automation.** This included QAs, as well as metrics to tune and control the process at the portfolio level.
- **"Agile" as a pervasive concept across IT and the business.** We used the same Agile tools (the Atlassian tools) across the business, and several business processes were reflected in the JIRA workflow engine, providing high visibility and reporting automation. Compliance and financial management were built into the processes. The whole business (customer service, fulfillment, sales, marketing, etc.) was able to escalate specific technical issues to the development team and see them rapidly resolved.
- **Adaptive architectures.** We used common technologies, common foundation services, and common core components across all applications, hugely reducing maintenance and extension costs.

The emphasis of some of our initial practices changed. For example, we had to address sprint fatigue: five to eight full releases to production *each week* by a team of 30 individuals is an incredible pace to keep up over a period of 20 months. Self-organizing teams were shortcutting Agile practices, with no particular negative impact: teams knew what everybody was doing, so even the 15-minute Scrum meetings were considered a waste of time. Teams with good velocity didn't care about measuring it. They chose to implement more features rather than spending time in story-point estimation and tracking.

The business had dashboards to verify the status of projects and complete access to the details of each project, feature, story, test, and defect and how these items related to each other. These dashboards were progressively tuned, since executives were focused on business drivers: How many giftcards have we sold today? How many visitors did we have today?

So some Agile best practices came and went. What stayed were the new integrated disciplines, the architectures, and the integrated collaboration tools that enabled responsiveness to the business.

CONCLUSIONS

Industry-wide adoption of Agile practices and tools at the project level is increasing at an exponential rate. Enterprises are learning to balance these project-level practices with the required IT-wide disciplines. Agility-in-the-large requires an integrated approach combining project-level and IT-wide practices, with (at the very least) adaptive architectures and software factories. The appropriate tools are a critical success factor in agility-in-the-large IT transformation.

Obviously, the IT transformation is never the primary objective: businesses must focus first of all on business success. Business objectives always trump Agile objectives. The roadmap to an agility-in-the-large transformation must deal with carefully selected priorities.

Installing new practices is relatively simple. Adopting integrated approaches that combine the disciplines required by IT for business alignment, business responsiveness, and (for example) self-organizing development teams is harder, and it significantly benefits from a heavy emphasis on adaptive architectures and software factories. When possible, transforming application portfolios to component-based, service-oriented architectures is a critical success factor, but transforming applications takes longer than transforming teams.

A word of caution: throughout this article, I differentiated between use of the term "agile" as a never-ending quest for nimbleness and the ability to adapt to change, and the use of the term "Agile" to indicate the set of best practices popularized by the Agile Manifesto and subsequent trends. Is "Agile" always agile? Not really. If a CEO needs to change sprint scope on a regular basis, Agile is not agile if it does not make this easy. If the VP of Sales needs an accurate "project percentage complete," to say "Agile does not measure percentages" is not agile. An agile organization can deal with all business requests. "Agile" will continue to expand and storm the industry if we stay away from dogmatic adoptions.

Our industry now has vast experience in reaching and continuously optimizing high levels of agility through the adoption and alignment of IT best practices (enterprise architectures, IT governance, IT strategy, application/project portfolio management, IT service management, cosourcing, and many others) and project-level Agile practices. Successful *agile enterprises* often have a healthy mix of enterprise discipline and project-level Agile practices, where the two levels of best practices integrate with and augment each other for IT and business success. This is clearly the current frontier of Agile adoption.

ENDNOTES

[1]COSM is a fourth-generation, disciplined, scalable agile approach to software manufacturing. For more information, see www.herzum.com/cosm.

[2]Atlassian (www.atlassian.com).

[3]The COSM software factory is a set of tools, technologies, processes, procedures, standards, and how to's that govern software development from requirements to deployment.

Peter Herzum is a serial entrepreneur, an internationally renowned IT and Agile expert, and a successful CIO. For his work at Wolfe, he was named one of the Top 100 CIOs of 2012 *by* CIO *magazine and was a finalist for the* 2012 CIO of the Year *award from the Pittsburgh Technology Council. He is the author of the industry bestseller* Business Component Factory*, which has been translated into multiple languages and adopted as a textbook in university courses and continuing education programs worldwide.*

As the founder and CTO of the Herzum Group, Mr. Herzum has extensive experience in large-scale Agile adoptions, enterprise and IT strategy, enterprise architectures, advanced and innovative technologies, and management and ALM methodologies specifically for software products and large enterprises. He has over 25 years' experience in managing, architecting, and consulting for large enterprises using innovative technologies. He is a frequent speaker at international conferences on topics such as Agile practices, enterprise and software architectures for large enterprises, IT innovation, and IT strategy. Mr. Herzum is particularly passionate about helping organizations use IT for business success, transforming themselves into agile, disciplined, lean, automated producers of innovative software solutions. And, of course, he is passionate about piano jazz. He can be reached at peter@herzum.com.

What It Means to Scale Agile Solution Delivery

by Mark Lines and Scott W. Ambler

Although many Agile teams are small, say 10 or fewer people, and either colocated or at least near-located, the majority of Agile teams work in more complex situations. For example, some teams are several dozen people in size and sometimes larger. Some teams are geographically distributed. Some are taking on very challenging problems. It's easy to observe that organizations are actively applying Agile strategies in a range of situations and are tailoring these strategies to suit their needs. We're learning that a context-sensitive approach to IT solution delivery is much more effective than a single, common approach prescribed to all teams. Individual teams find themselves in unique situations and therefore need the freedom to act accordingly.

This article describes:

- Context-dependent scaling factors
- How Disciplined Agile Delivery (DAD) provides a foundation from which to scale
- Why being goal-driven is the key to scaling
- The four critical goals for scaling

CONTEXT COUNTS: SCALING FACTORS IN AGILE SOFTWARE DELIVERY

What scaling factors should we consider when tailoring our approach to Agile solution delivery? Several years ago, while working with IBM customers around the world to adopt and scale Agile, Scott developed the Agile Scaling Model (ASM) to help answer this question. In parallel to this work on the ASM, Philippe Kruchten was working on something he terms "situational agility," the heart of which constitutes eight factors often referred to as the "Octopus model."[1] In late 2012, we began thinking about how to combine and evolve these two frameworks into one, originally calling the result the Process Context Framework (PCF). We moved away from that name because the framework was clearly applicable to more than just software process, and hence we adopted the name Software Development Context Framework (SDCF), which also includes people, process, and tools.[2]

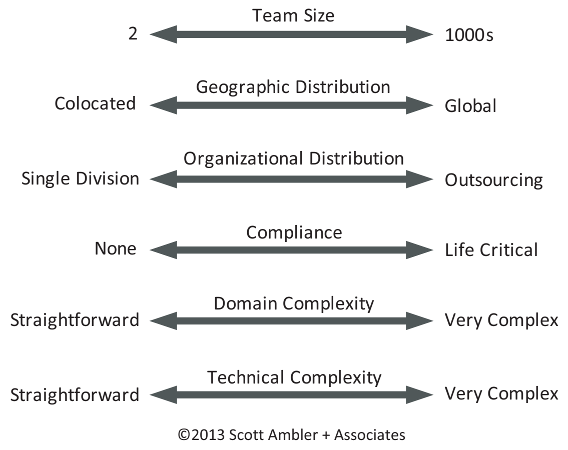


Figure 1 — Scaling factors faced by Agile teams.

Figure 1 summarizes the six scaling factors of the SDCF, indicating the range of each factor. On the left-hand side is the "simple" extreme, and on the right-hand side is the "challenging" extreme. Any given team will find itself somewhere on the spectrum for all six scaling factors, hopefully closer to the simple extreme on the left than the challenging extreme on the right. In various IT surveys over the years (most recently in the 2012 Agility at Scale survey[3]), we have found evidence that organizations are applying Agile at all levels of scale, so let there be no doubt that organizations are attempting to scale Agile.

Now that we understand the scaling factors faced by Agile teams and that these factors necessitate adapting our approach for more complex endeavors, we're in a position to discuss the realities of scaling Agile strategies within a team. Often organizations begin experimenting with Agile by adopting methods such as Scrum with a few ideas from XP, usually running a few pilot projects to learn about the fundamentals of Agile. A common realization is that teams need to go far beyond Scrum (a strategy that Scrum proponents refer to as "Scrum And"), and they start adding in techniques from other sources to address the issues Scrum doesn't. Until recently, the only recourse that organizations had was to formulate their own process — often still calling it Scrum — an endeavor that typically proved time-consuming, expensive, and difficult when

Agile process expertise wasn't available. But now organizations have the option of using the DAD framework, a hybrid approach that has already done a lot of the hard thinking about how all these Agile techniques fit together and when (and when not) to use each one. With DAD's goal-driven approach, it becomes easy to tailor the framework to address the scaling factors appropriately.

DISCIPLINED AGILE DELIVERY (DAD) AS A FOUNDATION FOR SCALING

As we've said, many organizations start their Agile journey by adopting Scrum because it describes a good strategy for leading Agile software teams. Scrum, however, 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 what issues they need to consider.

To address these challenges, the DAD process decision framework provides a more cohesive approach to Agile solution delivery.[4] To be more exact, here's our definition of DAD:

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

DAD provides a foundation from which to scale because it:

- **Addresses the full delivery lifecycle.** DAD recognizes that projects go through some startup activities as well as the work involved in transitioning the solution to stakeholders.
- **Is a hybrid of good ideas.** DAD adopts proven practices from methods such as Scrum, XP, Agile Modeling, and Kanban, to name a few. DAD shows how practices for management, testing, architecture, programming, continuous integration, deployment, and other practices fit together.
- **Is goal-driven.** We have found that process-related goals are fairly consistent across most types of projects. However, context is inevitably different, so DAD provides easy-to-consume guidance for how to adapt to the unique situations that you will face.

Of all these aspects of DAD, our experience is that adopting a goal-driven approach is the one that most organizations have missed in their attempts to successfully scale Agile solution delivery. Let's explore this further.

DRIVEN BY PROCESS GOALS: THE KEY TO SCALING

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 thereby one neck to wring), and other such ideas will get the job done. But we all know that some of these "rules" are meant to be broken. There are actually many reasonable methods for managing requirements change, there are various means of coordinating within a team besides a daily stand-up meeting, 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 in which it is appropriate. DAD describes a straightforward, easy-to-consume strategy that is goal-driven. It has a visual component (process goal diagrams that summarize the fundamental process decision points) and a textual component (goals tables that describe the factors to be considered, various options for each factor, and the tradeoffs between them).

Yes, Scrum's product backlog is one way to address changing stakeholder needs, but it isn't the only option, nor is it the best option in many situations.

Thanks to this goal-driven approach, DAD avoids being prescriptive and is therefore 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 that during construction you have the goal of addressing changing stakeholder needs. It also indicates that there are several factors related to successfully accomplishing a goal that warrant consideration and several techniques/practices that you should consider adopting to achieve that goal. DAD goes further and describes the advantages and disadvantages of each technique and the situations it is best suited for. Yes, Scrum's product backlog 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 2 shows the goals that are consistent with any project regardless of type, whether it is custom development or implementing a package.

FOUR CRITICAL GOALS FOR SCALING

Pareto Saves the Day

The goals *Explore Initial Scope*, *Identify Initial Technical Strategy*, *Coordinate Activities*, and *Move Closer to Deployable Release* seem to bear the brunt of an organization's process tailoring efforts when working at scale. It really does seem to be one of those Pareto situations where 20% (of the goals, in this case) accounts for 80% of the work. The process tailoring decisions that you make regarding these goals will vary greatly based on the various scaling factors we depicted in Figure 1.

Applying Goal Diagrams to Address Scaling Factors

Figure 3 provides an example of a DAD process goal diagram. To address the *Explore Initial Scope* process goal, we need to consider various issues or process factors in order to work most effectively in the context that we find ourselves in. While mainstream Agile methods trivialize the need to do any planning prior to coding, DAD recognizes that most projects do indeed spend some time exploring the scope of the project. As you can see from the figure, there are a number of choices for each factor. As with all process goal diagrams, Figure 3 doesn't provide an exhaustive list of options, although it does provide a pretty good start. Some choices, such as work item list, are bolded and italicized, which indicates that they are good places to start for a typical DAD team. Some issues show an arrow beside the options, which indicates that the choices at the top are generally the most effective and the better alternatives to strive for. A team will make hundreds of process decisions, and these diagrams can be used to ensure that the team considers the various options. An example of a decision might be what View Types could we use to depict scope? In this case, DAD recommends starting with a combination of usage modeling, domain modeling, and nonfunctional requirements. For usage modeling, user stories are the most popular Agile approach, but you could also create use case diagrams or personas as needed.

The process goal diagram for *Explore Initial Scope* gives us some ideas to consider when eliciting the requirements of the solution. For small projects, simply creating a work item list of stories and other work might be sufficient. However, even for small projects, ignoring other strategies such as various modeling techniques could mean that we risk missing functionality or, worse, creating work items for the wrong functionality. As the complexity of the domain increases, or a regulatory requirement demands it, it becomes necessary to vary the View Types as well as the Level of Detail. The geographic and organizational distribution typical in outsourcing scenarios will also lead to different levels of documentation requirements. Technical complexity is a scaling factor that will affect the scope identification approach. For example, air traffic control systems will

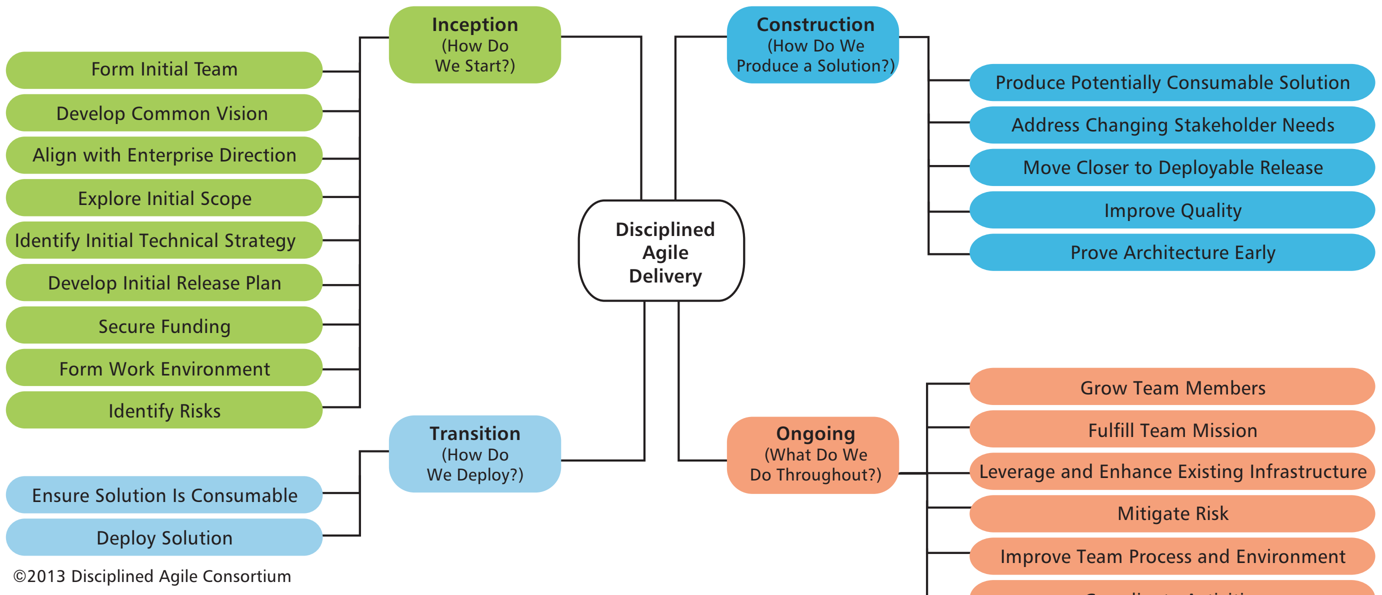

Inception (How Do We Start?)
Form Initial Team
Develop Common Vision
Align with Enterprise Direction
Explore Initial Scope
Identify Initial Technical Strategy
Develop Initial Release Plan
Secure Funding
Form Work Environment
Identify Risks
Construction (How Do We Produce a Solution?)
Produce Potentially Consumable Solution
Address Changing Stakeholder Needs
Move Closer to Deployable Release
Improve Quality
Prove Architecture Early
Disciplined Agile Delivery
Transition (How Do We Deploy?)
Ensure Solution Is Consumable
Deploy Solution
Ongoing (What Do We Do Throughout?)
Grow Team Members
Fulfill Team Mission
Leverage and Enhance Existing Infrastructure
Mitigate Risk
Improve Team Process and Environment
Coordinate Activities
©2013 Disciplined Agile Consortium

Figure 2 — The process goals of DAD.

require more than a Scrum product backlog of stories to adequately capture the scope of the system.

Figure 4 depicts the process goal diagram for *Identify Initial Technical Strategy*. For a simple website application, a minimalist approach to documenting the architecture would likely be appropriate. Informal modeling sessions around a whiteboard might be all that is required. However, for a technically complex system, your nonfunctional requirements might include the need to address concerns such as performance, load,

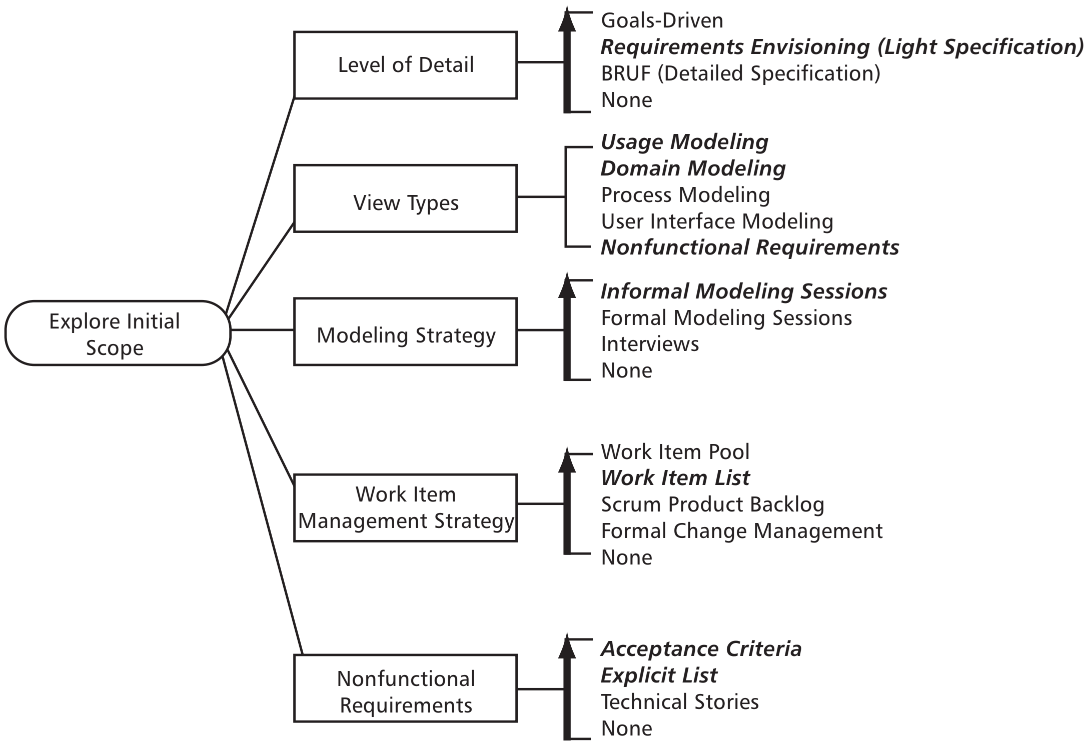


Figure 3 — Process goal diagram: Explore Initial Scope.

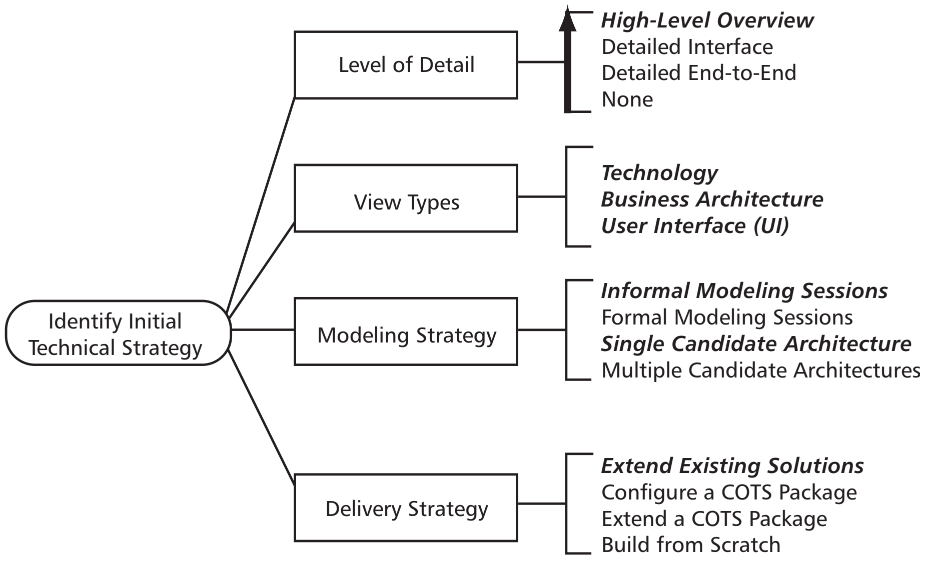


Figure 4 — Process goal diagram: Identify Initial Technical Strategy.

fault tolerance, security, and other architectural challenges. Perhaps multiple candidate architectures need to be considered. In cases like this, more View Types will be required, at a greater Level of Detail. For geographically or organizationally dispersed teams, sharing information about the architecture will also need to be done in a more formal manner. An interesting consideration when identifying a Delivery Strategy is whether you are going to build from scratch, extend existing solutions, or perhaps purchase and integrate a package. You can in fact use an Agile approach to each of these scenarios, although the way that you describe your technical approach will need to be adapted.

The process goal diagram for *Coordinate Activities* is shown in Figure 5. This process goal 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 for coordinating 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. You are also likely to employ Share Information strategies. If you have 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 somehow coordinate releases with other delivery teams and with your operations team). Third, several issues address scaling factors. For example, large teams (often called "programs") will find that they need to adopt strategies called out by Coordinate Within Program. Teams that are geographically or organizationally distributed will need to consider strategies from Coordinate Between Locations. They may find that they need to have people who act as boundary spanners (i.e., serve as key liaisons at each site) and even invest in enabling some people to gather physically at critical times. 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.

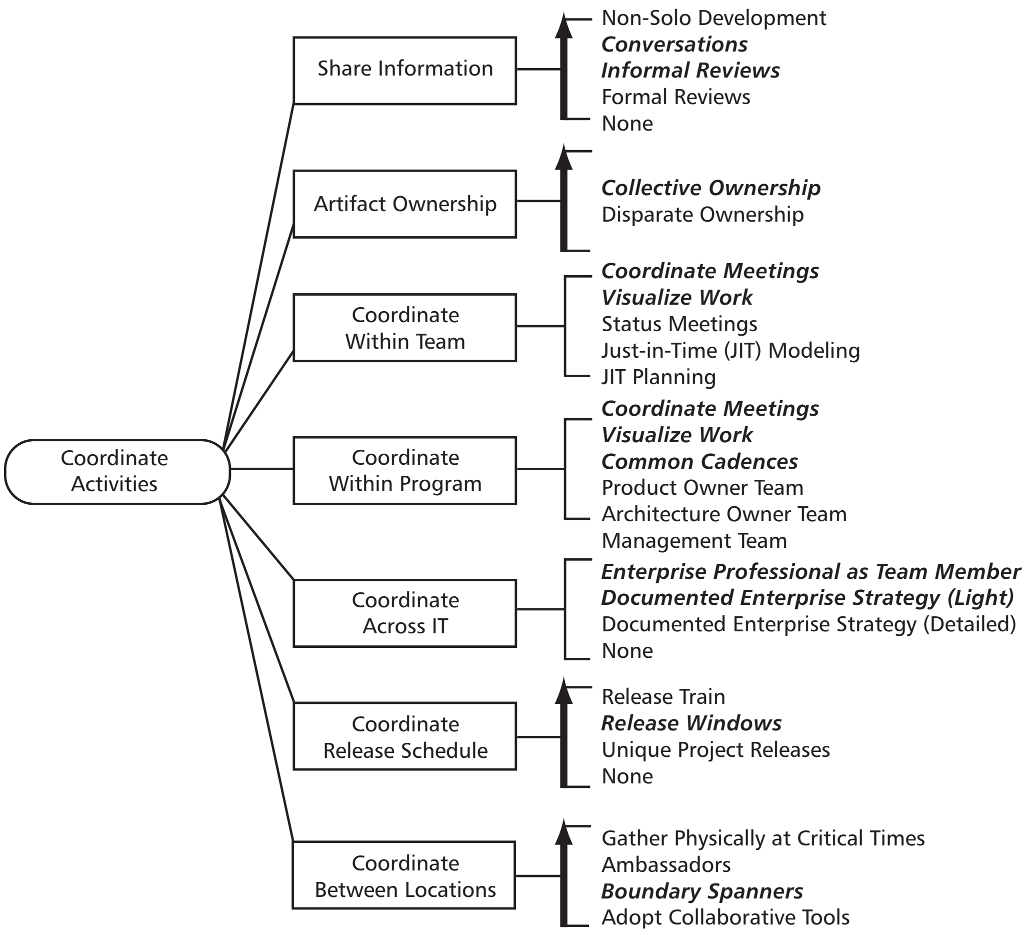


Figure 5 — Process goal diagram: Coordinate Activities.

Lastly, for large-scale efforts where multiple teams need to deliver functionality in parallel, it might make sense to Coordinate Release Schedule using a release train.

Figure 6 provides some ongoing guidance to consider when fulfilling the goal *Move Closer to Deployable Release*. For small projects, it is relatively easy to have working and tested software at all times, as well as to make frequent releases to stakeholders if basic Agile technical practices such as continuous integration and continuous deployment are in place. However, as projects face various scaling factors, this becomes more difficult. With regard to Deployment Strategy, many companies implementing Agile like to boast about their ability to deploy new functionality to customers on a daily basis. In many situations, this is only possible due to the simplistic or localized impact of the change made to the solution. In addition, this typically only appears in environments with minimal compliance requirements and in organizations that are risk tolerant. However, for the majority of enterprises that are implementing Agile at scale, the ability to do daily deployments of material functionality is not a practical reality. Highly regulated organizations are generally quite risk-averse, and they therefore need to devote more effort toward Validation and Verification activities. For instance, they may find it necessary to conduct formal reviews as well as static and dynamic analysis of the work products. As an additional measure of quality assurance, parallel independent testing is often a good approach to Validation. As domain and technical complexity increases, the need for sophisticated Validation practices also becomes more acute. For complex solutions, manual regression testing is impractical. Automated testing practices can increase confidence that changing or adding new functionality will not break existing functionality and can give the team the confidence to rework aspects of the system without breaking something else.

While we have given some examples of how these diagrams can be very useful for providing guidance on what choices to make in various scaling scenarios, it is worthwhile to restate that there are textual tables behind each diagram to provide some more detailed advice for how to apply DAD's process decision framework. Of the 22 DAD goals listed in Figure 2, the four we've discussed here will address roughly 80% of the tailoring required to address the scaling factors called

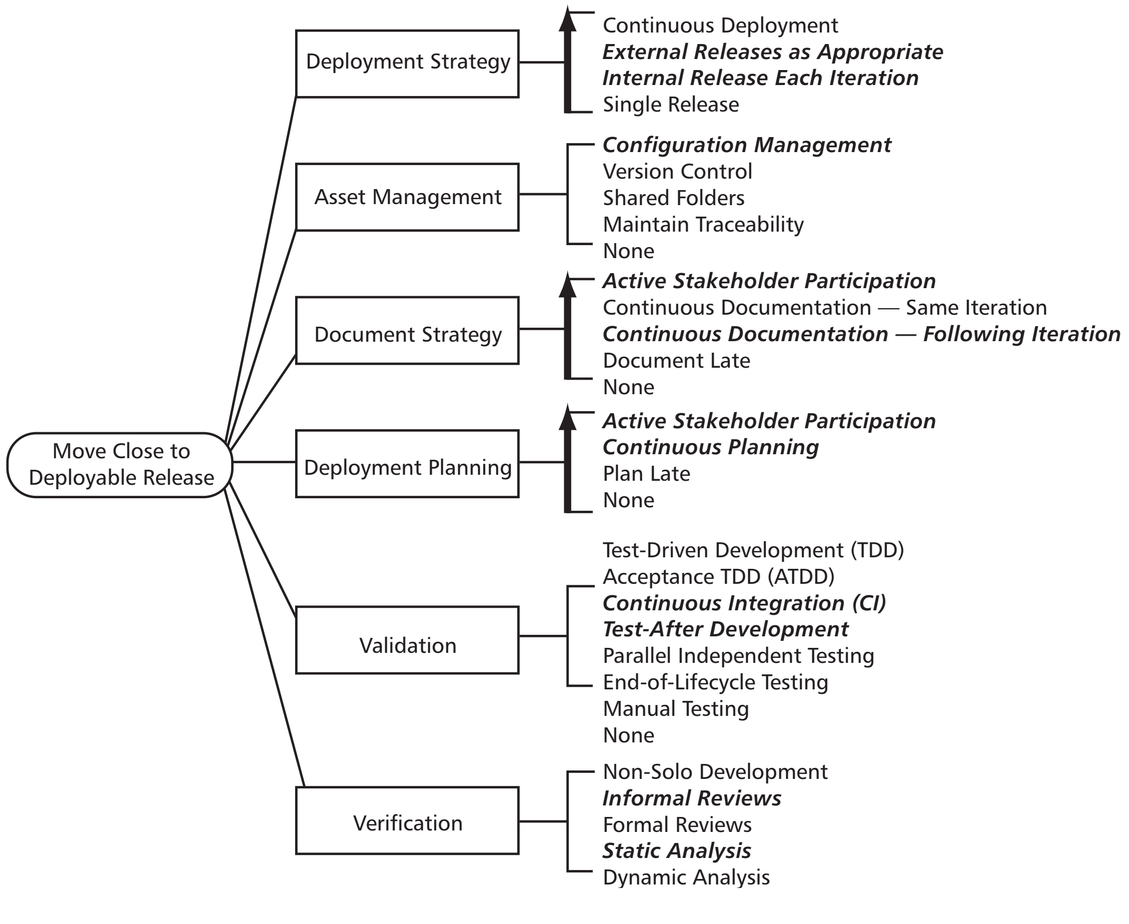


Figure 6 — Process goal diagram: Move Closer to a Deployable Release.

out in Figure 1. The other 18 goals will still be affected, but to a much lesser extent.

PARTING THOUGHTS

There are several fundamental advantages to taking a goal-driven approach to Agile solution delivery. First, it makes your process options obvious. The 22 process goals, in combination with the more detailed process goal diagrams, make the range of available Agile practices very clear. Second, DAD's goal-driven approach supports process tailoring by making your process decisions explicit. Third, scaling Agile delivery strategies is enabled by delineating the strengths and weaknesses of each practice. (This advice is currently captured as textual tables in the DAD book[5] and programmatically in several software process tools.) Fourth, a goals-based approach clarifies 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 a delivery team is facing.

DAD seems to be resonating with organizations around the world that are looking for a pragmatic approach to scaling Agile in the enterprise. This stuff is hard.[6] Presenting various strategies for achieving scale in a straightforward manner — with advantages, disadvantages, and considerations for each approach — can make the challenges less daunting.

ENDNOTES

[1]Kruchten, Philippe. "Contextualizing Agile Software Development." *Journal of Software: Evolution and Process*, Vol. 25, No. 4, April 2013, pp. 351-361.

[2]Ambler, Scott W. "Scaling Agile: The Software Development Context Framework." Disciplined Agile Delivery, 15 March 2013 (http://disciplinedagiledelivery.wordpress.com/2013/03/15/sdcf).

[3]Ambler, Scott W. "Agility at Scale Survey: Results from the Summer 2012 DDJ State of the IT Union Survey." Ambysoft, 2012 (www.ambysoft.com/surveys/stateOfITUnion201209.html).

[4]Ambler, Scott W., and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Development in the Enterprise*. IBM Press, 2012.

[5]Ambler and Lines (see 4).

[6]For more information about DAD and the four process goals described in this article, visit DisciplinedAgileDelivery.com. The Disciplined Agile Consortium (DisciplinedAgileConsortium.org) maintains a directory of certified DAD practitioners and instructors.

Mark Lines is Managing Partner at Scott W. Ambler + Associates. With Scott Ambler, he is coauthor of Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. *Mr. Lines is a "disciplined" Agile coach, helping organizations all over the world transform from traditional to Agile methods. He also helps customize Agile governance practices to accelerate complex projects in large enterprises. Mr. Lines delivers workshops on Disciplined Agile Delivery (DAD) as well as other Agile topics. He writes for many publications, is a frequent speaker at industry conferences, and blogs about DAD at www.DisciplinedAgileDelivery.com. He can be reached at mark@scottwambler.com.*

Scott W. Ambler is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture *and* Agile Product & Project Management *practices. He is the thought leader behind the Disciplined Agile Delivery (DAD) process decision framework, Agile Model Driven Development (AMDD), the Agile Data (AD) method, and the Enterprise Unified Process (EUP), and he works with clients around the world to improve the way they develop software.*

Mr. Ambler is coauthor of several software development books, including Disciplined Agile Delivery, Agile Modeling, The Elements of UML 2.0 Style, Agile Database Techniques, *and* The Enterprise Unified Process. *He is also a Senior Contributing Editor with* Dr. Dobb's Journal. *Mr. Ambler has spoken at a wide variety of international conferences, including* Agile 20XX, Software Development, IBM Innovate, Java Expo, *and* Application Development. *He can be reached at sambler@cutter.com.*

Supporting Governance in Disciplined Agile Delivery Using Noninvasive Measurement and Process Mining

by Saulius Astromskis, Andrea Janes, Alberto Sillitti, and Giancarlo Succi

Customers not only expect developers to provide working code, but also to contribute toward a working IT system. Developers need to cooperate closely with the team that handles IT operations in order to coordinate and integrate software development, technology operations, and quality assurance.[1] To accomplish this, approaches like the Disciplined Agile Delivery (DAD) framework have been proposed.

The DAD process decision framework differentiates itself from other Agile methods in that it provides an end-to-end solution delivery lifecycle (i.e., a description of how a team can proceed from the inception of an IT solution to its release).[2] It incorporates recognized software development methods such as Scrum, Kanban, Agile Modeling, and Extreme Programming and extends these "construction-focused" methods with approaches that ensure the delivery of a solution and not just working code. DAD considers governance to be of major importance, and it incorporates explicit guidelines to implement governance into the process.

The DAD framework also incorporates ideas coming from DevOps, a movement that addresses the problem that developers and system administrators basically have opposing interests. Stakeholders expect developers to constantly increase the provided value, to exploit new technologies, to add new features — in other words, to *improve* software. In contrast, stakeholders expect administrators to deploy software and keep it working. Naturally, administrators tend to resist change, as it threatens the stability of the system.

This is the result of a traditional division of labor following the principle *divide et impera* ("divide and rule"): one team develops, another team tests, and another team deploys. In such companies, the work of developers is "just thrown over the wall" to the system administrators, and they have to deploy the changes. If a problem arises, system administrators adjust or write their own deployment scripts, modify configuration files, adapt the installation environment, and so on to reflect the production environment, which might be different from the development or QA environment. Often they duplicate the work already done in previous environments, and since they cannot build on the experience gained there, they may introduce or uncover new bugs.[3] If the deployment fails, then a blame game begins, in which system administrators blame developers for writing bad software and developers blame system administrators for not being able to install it.

One goal of DAD is to efficiently guide the communication between different departments and groups, and by doing so, to facilitate the process of achieving common goals more efficiently. We discuss in this article how we can support this process by visualizing the steps that are actually carried out and comparing them with the steps that *should* be carried out. We will be able to see the sequence in which these activities are executed and how the effort is distributed among the executed activities. This knowledge can later be used to find opportunities to collaborate, improve, or change.

GOVERNANCE IN DISCIPLINED AGILE DELIVERY

DAD divides the whole solution lifecycle into three main phases:

1. Inception
2. Construction
3. Transition

Figure 1 shows the activities foreseen for the Construction phase. The horizontal axis of the diagram represents the timeline of the implementation, divided into five phases. The activities inside each phase contribute to four goals, shown on the vertical axis.

Figure 1 shows the activities recommended by DAD. Activities for which we automate the measurement are marked with a circle; the other activities are marked with a diamond. Below we will discuss how to automate the measurement of activities marked with a circle.

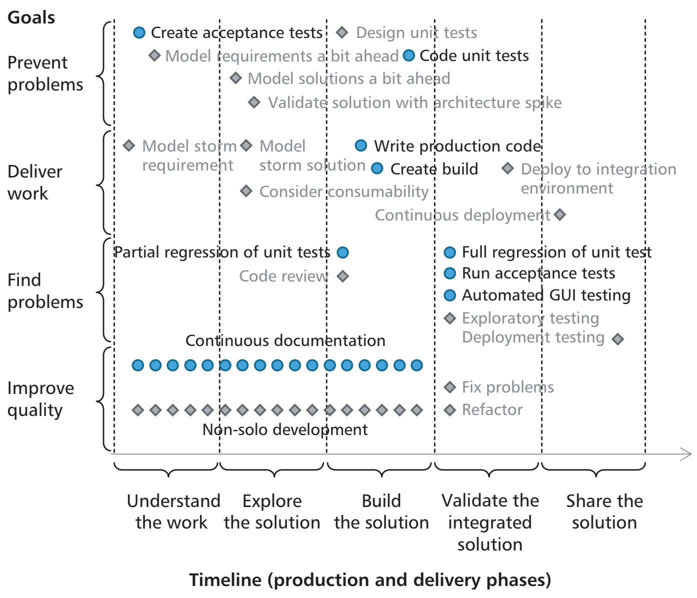


Figure 1 — Phases, goals, and activities to deliver a work item. (Adapted from Ambler and Lines.)

To efficiently manage the communication between different departments and groups, DAD makes the following recommendations, which it subsumes under the term "Information technology governance":[4]

- Establish chains of responsibility, authority, and communication in support of the overall enterprise's goals
- Establish measurements, policies, standards, and control mechanisms to enable people to carry out their roles and responsibilities effectively

In this article, we focus on this governance objective, particularly on how to establish measurements and control mechanisms using process mining. We will show how to evaluate:

- Whether the process steps shown in Figure 1 are executed (i.e., process mining)
- Whether the sequence of steps follows the recommended one (i.e., process conformance checking)

COMBINING NONINVASIVE MEASUREMENT AND PROCESS MINING TO MAKE THE PROCESS VISIBLE

In this section, we will explain how to combine noninvasive measurement and process mining[5] to support governance within Disciplined Agile Delivery. We use the process depicted in Figure 1 for our example.

An Architecture to Collect Noninvasive Measurements

To observe the workflow performed by the developers, we use measurement probes (more on these below) to collect data about the interactions of the developers with the development tools the team uses to implement a work item.

Let us assume we are at the beginning of a new iteration, in which the development team begins implementing a set of work items. The project is developing a Web application in Java using Apache Tomcat as a platform. The tools used in the development process are:

- Twist (from Thoughtworks) for acceptance testing
- Visual Paradigm for modeling
- Eclipse IDE for coding and unit testing
- Jenkins (from Jenkins CI, formerly Hudson Labs) for continuous integration
- Sikuli Script for GUI testing
- Microsoft Office for documentation
- Bugzilla for task management

The work items selected for the current iteration are requests for new features in the system. The development team implements each work item using the activities depicted in Figure 1.

To collect data about which applications developers use, when they use them, and what they do with them, we have developed an application (we call it "Trace") that tracks which window the developer is focused on. With this application (using the API offered by Windows), we are able to log whenever the focus changes from one window to another. If the focus changes, the application logs the following data:

- The time and date of the focus change
- The name of the application that was used
- The title of the window (this occurs only for those applications in which the title of the window is the name of the document being edited)

The team uses Bugzilla to manage the status of a work item. An add-on[6] written for Bugzilla reports to the server the following data as a work item is changed:

- The time and date of the event
- The ID of the work item
- The event type (added, modified, removed)
- The status of the work item (new, assigned, released, etc.)

Moreover, the team uses two additional measurement probes: an add-in[7] for Microsoft Office and a plug-in[8] for Eclipse. Whenever a developer changes the focus from one document or one piece of code to another, these probes report:

- The time and date of the focus change
- In Microsoft Word: the name of the document
- In Eclipse: the name of the project, the namespace, the name of the class, the current method signature, and the current action (edit, debug, execute)

The data from all four measurement probes (Trace, Bugzilla, Microsoft Office, and Eclipse) is sent to the server and fed into a process-mining algorithm. A possible architecture of the measurement system described here is depicted in Figure 2. As shown, the data is collected on the team members' computers (laptops or workstations) and is sent to a message queue.

We use a message queue to minimize the workload of the client computers. Developers do not want their machines to slow down because some measurement program is running in the background. By employing a message queue (we use Apache ActiveMQ), we can minimize the time the client machine is busy uploading data. The data is then read from the message queue, processed, and inserted into the data warehouse. Due to the large amount of data we collect, we use a NoSQL database, Apache Cassandra. The data is then extracted from the data warehouse and processed to display it on a dashboard.[9] An example of the data our system collects is shown in Table 1.

The format of the collected data is the same for other phases, too, except that during the development phase, more detailed data (also project names, namespaces, class names, and method signatures) is collected.

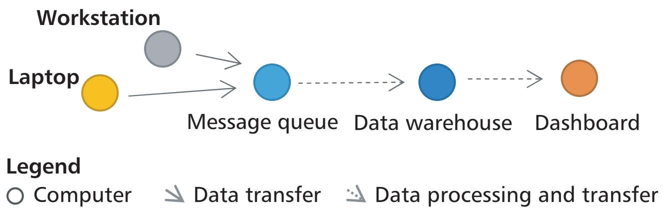


Figure 2 — Data flow within the measurement framework.

How to Analyze the Data Using Process Mining

Before we can use the data in Table 1, it has to be preprocessed. We have to map the collected events to those activities that we want to study. In our case, these are the activities of Figure 1. In this example, we look only at the activities marked with a circle. To assign activities to events, we define logical rules, as in Table 2.

After applying the rules to the events, we can pass them to a process-mining algorithm that mines a process model. This process model describes what is really happening during development and can be used to compare that to what should happen. Thus we compare the real process to the ideal process. The mined process shows the identified activities, their sequence, and timing information. We exclude events that do not match any of the rules.

To extract the visual process model from the event log, we use an algorithm called the heuristics miner,[10] as this algorithm is suited for real-world data.[11] The heuristics miner algorithm is able to deal with noisy data, ignoring rare events and activities, and therefore to generate simple models that represent only the most common behavior.

The process model we obtained in our case is depicted in Figure 3.

This visualization of the mined process model reveals the most common transitions between the activities of

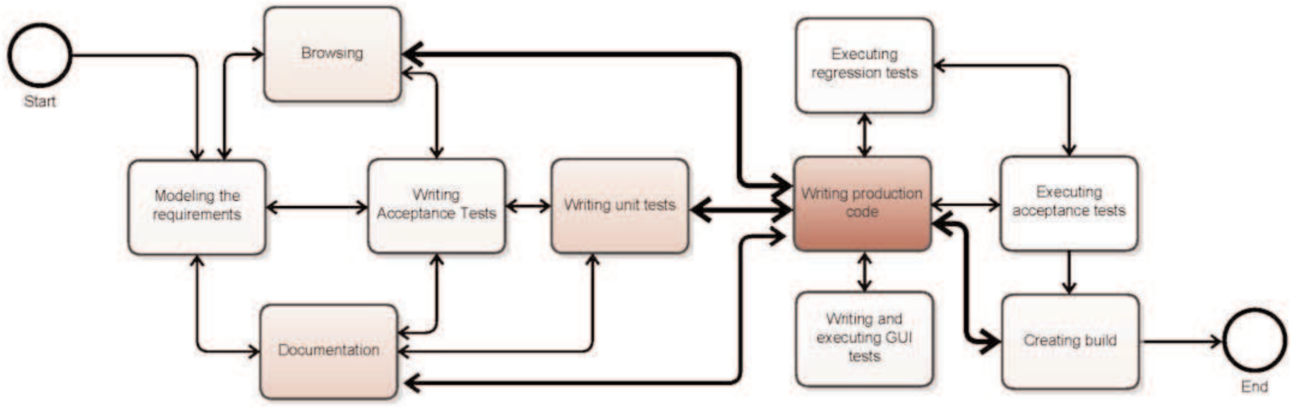
Table 1 — Excerpt of the Data Collected During the Execution of the Iteration*

User	Application	File	Source	Start	End	Action
1	Twist	projects/p8/at1scn	1	9:19:01	9:21:15	
1	Chrome		1	9:35:07	9:37:23	
1	MS Word	projects/p4/Req02.doc	1	10:01:51	10:12:51	
1	Eclipse	ArchSpike/web/login.jsp	2	10:35:11	10:44:24	Edit
1	MS Word	projects/p1/Req02.doc	1	11:05:54	11:15:32	
1	Eclipse	ArchSpike/web/login.java	2	11:15:32	11:19:08	Execute
…	…	…	…	…	…	

*This excerpt shows the data coming from one developer in the "Understand the work" phase, in which he or she uses the Twist tool to create the acceptance tests and Microsoft Word to write the more detailed specification of the requirements.

Table 2 — Some Rules for Assigning Activities to Events

Activity	Rules
Create acceptance tests	Application = Twist
Create GUI tests	Application = Sikuli
Model the requirements	Application = MS Word and File is "req*.doc"
Model the solution	Application = Visual Paradigm
Model storm the solution	Application = Eclipse and Project is "arch_spike_WI*" and Action is "Edit"
Write documentation	Application = Microsoft Word
Code unit tests	Application = Eclipse and File is "test*.java" and Action is "Edit"
Write production code (business logic)	Application = Eclipse and File is not "test*.java" and File is not "*.jsp"
Write code (user interface)	Application = Eclipse and File is "*.jsp" and Action is "Edit"
Create build	Application = Eclipse and File is "*.ant" and Action is "Edit"
Execute unit tests	Application = Eclipse and File is "test*.java" and Action is "Execute"



Start
Browsing
Executing regression tests
Modeling the requirements
Writing Acceptance Tests
Writing unit tests
Writing production code
Executing acceptance tests
Documentation
Writing and executing GUI tests
Creating build
End

Figure 3 — Mined process model of a work item implementation as defined in Figure 1.

the events we collected. The thickness of a line represents the frequency of transitions; the intensity of the shading of an activity represents the relative effort compared to the overall effort.

This model is interpreted as follows:

- **An activity is present in the model.** The data contains a significant amount of events assigned (through a rule) to the activity shown in the model.
- **An activity is not present in the model.** The data *does not* contain a significant amount of events assigned (through a rule) to the activity shown in the model. This could also mean that the rules do not assign the correct activity to the collected events.
- **One activity is connected with another activity.** The data contains a significant, *low* amount of events in which one activity B follows another activity A.
- **One activity is connected with another activity through a thick line.** The data contains a significant, *high* amount of events in which one activity B follows another activity A.
- **The line connecting two activities has an arrow only on one side.** The data contains only transitions from the first activity to the second but *not* vice versa.
- **The line connecting two activities has arrows on both sides.** The data contains transitions from the first activity to the second *and* vice versa.
- **An activity is shaded with a light color.** Considering the total effort, the amount of time spent in this activity is relatively *low*.
- **An activity is shaded with a dark color.** Considering the total effort, the amount of time spent in this activity is relatively *high*.

CONCLUSION

The process model can be mined for the whole team or for each individual. To avoid a negative impact on the motivation of the collaborators — they might feel spied on by the measurement probes — we recommend aggregating the data on the team level and looking at what the team is doing well and where it needs to improve. (That said, a developer might want to extract the process model of only his or her data to study how he or she is developing.) The mined process model helps to identify:

- The activities that the team carries out
- The sequence of activities the team carries out
- The distribution of effort between the activities
- Missing activities
- The process conformance of the team (in general and for specific work items)
- The typical process per work item type
- The degree of conformance to a defined process

Furthermore, the mined process model can be used for sprint retrospectives, to show stakeholders how the sprint was actually executed and to brainstorm about possible ways to improve the next one. Such an approach can help both the team and management to understand how the team spends its time and whether it is doing the right things. The team can also use the obtained process model to measure improvement; that is, to evaluate the decisions taken before the sprint by observing whether those decisions led to process improvement.

Software is intangible, and so are the processes used to develop it. This fact creates a challenge when it comes to governance within a software development organization. It is difficult to objectively measure and evaluate the performance of the processes in place and to check whether they conform to the ones imposed by management. The noninvasive measurement and process mining approach we have described in this article makes it easier to visualize and analyze the actual processes in an organization. The results can be used by both development teams themselves and by management to properly guide the delivery of products.

ENDNOTES

[1]Spinellis, Diomidis. "Don't Install Software by Hand." *IEEE Software*, Vol. 29, No. 4, July 2012.

[2]Ambler, Scott W., and Mark Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, 2012.

[3]Edwards, Damon. "What is DevOps?" dev2ops, 23 February 2010 (http://dev2ops.org/2010/02/what-is-devops).

[4]Ambler and Lines (see 2).

[5]The basics of noninvasive measurement and process mining are covered in: Astromskis, Saulius, Andrea Janes, Alberto Sillitti, and Giancarlo Succi. "Implementing Organization-Wide Gemba Using Noninvasive Process Mining." *Cutter IT Journal*, Vol. 26, No. 4, 2013.

[6]The Bugzilla development team uses the term "add-on" to describe an extension to Bugzilla.

[7]Microsoft uses the term "add-in" to describe an extension to Microsoft Office.

[8]The Eclipse development team uses the term "plug-in" to describe an extension to Eclipse.

[9]Janes Andrea, Alberto Sillitti, and Giancarlo Succi. "Effective Dashboard Design." *Cutter IT Journal*, Vol. 26, No. 1, 2013.

[10]De Weerdt, Jochen, Manu De Backer, Jan Vanthienen, and Bart Baesens. "A Multi-Dimensional Quality Assessment of State-of-the-Art Process Discovery Algorithms Using Real-Life Event Logs." *Information Systems*, Vol. 37, No. 7, November 2012.

[11]De Weerdt et al. (see 10).

Saulius Astromskis is a PhD student at the Free University of Bolzano-Bozen (Italy). His research interests include Lean software development, noninvasive software measurement, and software development process mining. Mr. Astromskis holds a master's of science degree in informatics from Vilnius University, Lithuania. He can be reached at saulius.astromskis@unibz.it.

Andrea Janes is Assistant Professor of Software Engineering at the Free University of Bolzano-Bozen. His research interests include Lean software development, value-based software engineering, and empirical software engineering. Mr. Janes holds a master's of science degree in business informatics from the Technical University of Vienna and is pursuing a PhD in informatics at the University of Klagenfurt (Austria). He can be reached at ajanes@unibz.it.

Alberto Sillitti is Associate Professor of Computer Science at the Free University of Bolzano-Bozen. He has been involved in several EU-funded projects related to open source software, services architectures, and Agile methods in which he applies noninvasive measurement approaches. Additional research areas include mobile and Web services. Dr. Sillitti has served as a program committee member of several international conferences and as program chair of OSS 2007, XP2010, *and* XP2011. *He is the author of more than 80 papers for international conferences and journals. Dr. Sillitti holds a PhD in electrical and computer engineering from the University of Genoa (Italy). He can be reached at asillitti@unibz.it.*

Giancarlo Succi is a Senior Consultant with Cutter Consortium's Agile Product & Project Management *practice. He is also Professor of Software Engineering and Director of the Center for Applied Software Engineering at the Free University of Bolzano-Bozen. Dr. Succi's research areas include Agile methodologies, open source development, empirical software engineering, software product lines, software reuse, and software engineering over the Internet. He is the author of four books and more than 300 papers published in international conferences proceedings and journals. He can be reached at gsucci@cutter.com.*

10 Principles for Success in Distributed Agile Delivery

by Raja Bavani

The Agile movement is all about delivering business value in short iterations at a sustainable pace, adapting to changing business needs. Agile software development focuses on early delivery of working software and considers working software the primary measure of progress. It creates an environment that responds to change by being flexible and nimble. It discourages the creation of extensive documents that do not add any value. In simple terms, adherence to the Agile Manifesto and the principles behind it is the foundation of Agile delivery.

Since 2001, various Agile software development and project delivery methods (XP, Scrum, DSDM, etc.) have gained widespread popularity. Originally, such methods were primarily developed for software projects executed at a single location. Today, with many adopters and practitioners across the globe, Agile methods are showing promising results in multisite projects, too. Offshore delivery models have been successful in application maintenance and enhancement projects for more than two decades. In the case of development projects, iterative lifecycle approaches are more widespread and acceptable than the classical waterfall approach in delivering results and ensuring customer satisfaction. Thinking beyond the original 12 Agile principles declared by the authors of the Agile Manifesto is essential to the success of distributed Agile projects. In this article, I present 10 principles of distributed Agile. These 10 principles, together with the 12 Agile Manifesto principles, provide the necessary foundation for geographically distributed Agile teams.

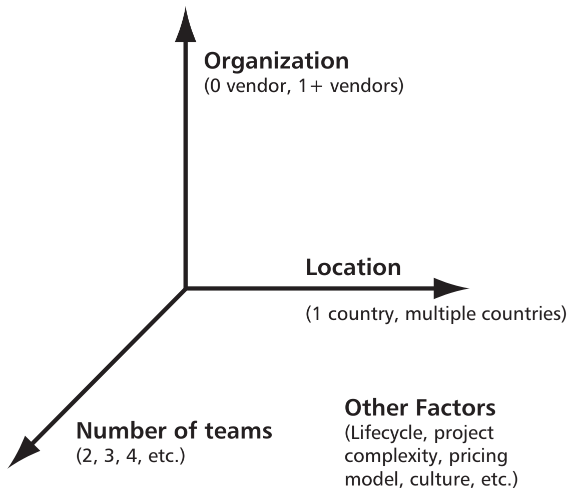

Organization
(0 vendor, 1+ vendors)
Location
(1 country, multiple countries)
Number of teams
(2, 3, 4, etc.)
Other Factors
(Lifecycle, project complexity, pricing model, culture, etc.)

Figure 1 — Distribution of teams.

Distributed Agile software development/testing simply involves applying Agile principles and practices to software projects executed by distributed teams. These teams could be on two or more floors of the same building or in different buildings, cities, or countries and across time zones. Distributed Agile teams require more discipline than colocated Agile teams. They need to be disciplined enough to decide what is "just enough" in order to move swiftly and avoid waste. For this, their approach needs to be based on time-tested principles.

GLOBAL SOFTWARE ENGINEERING AND DISTRIBUTED AGILE

Global software engineering (GSE) entails software engineering projects executed by virtual teams from different time zones and diverse cultures. Over the past decade, GSE has become popular due to several factors such as optimal costs, availability of a skilled pool of resources, and globalization trends such as mergers and acquisitions. Distribution of teams becomes a complex issue as an increasing number of teams from different organizations participate in a project from different locations or sites, as shown in Figure 1. Distributed teams also face such challenges as project complexity, disparate distribution of lifecycle activities, different pricing models (which can lead to different ways of measuring outcomes), and cultural incongruities.

In theory, distributed Agile is nothing but the application of Agile methods (such as XP, Scrum, and DSDM) in GSE projects. In practice, as we will see, such methods need to be customized, or a hybrid method implemented, to suit the project context.

THE CHALLENGES OF DISTRIBUTED AGILE

The challenges of distributed Agile[1] can be divided into three broad categories:

1. Communication and coordination
2. Time zone differences
3. People, culture, and leadership style

1. Communication and Coordination

Geographically distributed teams do not get the opportunity to have face-to-face meetings or informal hallway or water-cooler discussions. In contrast, colocated team members have the advantage of interacting with each other and with the onsite customer on a daily basis. This helps them understand and refine requirements in a timely manner. Distributed teams have to depend on a set of tools and processes in order to communicate and coordinate with each other. Also, unlike colocated teams, distributed teams need to practice just-enough documentation in order to create and retain knowledge across teams. This requires discipline and rigor.

2. Time Zone Differences

Time zone differences will obviously impact distributed teams. This impact can be either positive or negative depending on the work culture and relationship among team members. For example, two teams from different time zones with five hours of overlap may feel very comfortable working with each other, as they can communicate in real time with relatively little effort. However, the more they utilize their time in communication and coordination, the less time they will have for other engineering activities. At the other extreme, if there are two teams with only one hour of overlap, they may have to work extra hours to achieve adequate communication and coordination unless they are exceptionally efficient and smart in using their one-hour overlap. The way distributed teams manage this issue depends on the next challenge of distributed Agile: people, culture, and leadership style.

3. People, Culture, and Leadership Style

People from different geographies or countries have discernible traits related to how they communicate, what they really mean by the things they convey, or how they collaborate with team members. For example, when someone in the US says "yes," he or she means a firm "yes," whereas if it comes from someone in India, most likely it means "Let me try and see" (i.e., not a firm "yes"). Also, the culture of the organization plays a significant role when it comes to behavior at work. Leadership style can be command-and-control or collaborative in nature. In one project situation, the master location may be driving the other locations. In another, there may be a collaborative partnership among all locations. Leadership style determines the way a team is taken care of and nurtured in exceptional situations. When things go wrong, some leaders look at the apparent results and take rapid action, whereas a seasoned leader would collect all the facts and find the root causes before arriving at a decision.

Organizations that are new to Agile or going through Agile transformation find it very difficult to cope with these challenges, whereas organizations that promote Agile and have an established Agile culture tend to manage these challenges confidently and execute successful projects. Geographically distributed teams need to understand and appreciate cultural differences and work together to ensure harmony and rapport among all teams. Successful teams consciously adhere to certain principles, and it is this principle-centered approach that helps them face the challenges of distributed Agile and deliver the best results.

When geographically distributed teams are driven by a cookbook methodology, they struggle to deliver meaningful solutions.

10 PRINCIPLES OF DISTRIBUTED AGILE DELIVERY

Through my experience working with project teams and in interactions with industry experts, I have garnered and applied the following 10 principles in distributed Agile delivery contexts. The results have been positive throughout.

1. Methodology Is Driven by Project Teams

Agile software development in a distributed environment does not mean step-by-step implementation of any specific Agile methodology (e.g., Scrum) with high expectations of on-time, high-quality delivery. When geographically distributed teams are driven by a cookbook methodology, they struggle to deliver meaningful solutions. This approach can hinder the team members' ability to listen to one another and respect new ideas, thus making it harder to customize the methodology to suit the project context. An orchestra of expert musicians may be equipped with impeccable scores and world-class instruments, but if the musicians lack people skills or cannot collaborate, they will fail to deliver the best performance in a given situation. In a geographically distributed project, the best way to avoid a similar fate is to ensure that the methodology is driven by teams!

This means collaboration among distributed teams to identify processes that follow Agile principles and to put those processes together in a methodology that works for them. Distributed Agile projects suffer when a methodology adopted by one subteam is allowed to drive the rest of the team. Successful distributed Agile projects happen when collaborative teams work together to define a methodology that suits the overall project context. The definition of such a methodology occurs by means of open communication and ongoing minor adjustments to make things work. Furthermore, a methodology that works for one distributed ecosystem may not work for another distributed ecosystem. The reason is that while the basic tenets of a methodology may remain intact, the implementation details will vary across ecosystems.[2]

This distributed Agile principle focuses on two dimensions: (1) a collaborative approach to tailoring a methodology to fit a context, and (2) ongoing efforts to continuously improve the methodology in order to sustain throughput and quality. While focusing on these two dimensions, distributed project teams must think beyond standalone "named" Agile methods that work for small, colocated teams and instead learn from disciplined frameworks such as the Unified Process. Enterprise projects require a more rigorous approach because of stage gates and entry criteria for budget approvals and regulatory compliance. Understanding the Disciplined Agile Delivery framework[3] and tailoring[4] it to suit enterprise projects is a way to adhere to this principle. Obviously, when there is a need for a methodology expert to help you define what is right for your project (which is the case in most situations), it is worth considering an expert coach to assist you in the first two or three projects.

2. Consistent Usage of Common Tools Improves Productivity

Tool selection plays a vital role in distributed Agile projects. For example, effective tools for managing user stories as well as related communication and coordination mechanisms contribute positively to distributed Agile projects. Indecisive approaches to tool implementation or introduction of new tools during project execution will hamper effective management of user stories. It is strongly recommended that distributed Agile teams consider a Web-based tool that supports specification of user stories as well as facilitates collaboration among team members. Not having the right tool for managing users stories will increase the risk of requirements being misunderstood. This will have a direct impact on software quality as well as team productivity.

Team members in distributed teams must also have access to a standardized set of tools for engineering activities such as design, coding, static analysis, unit testing, build automation, test automation, defect tracking, and so on. Moreover, they need to use such tools consistently in order to realize the benefits. Disparate tools result in compatibility issues and hinder team productivity.

Ensuring that geographically distributed teams have access to common tools right from the first iteration is critical to success. When distributed teams accept and adhere to this principle, they identify tools and make such tools available to all team members. Otherwise the principle becomes no more than an aspirational statement that is only partially fulfilled, resulting in either nonavailability of tools or inconsistent usage of them.

Process frameworks such as DAD recommend a phased approach to software delivery in which the first phase (the Inception phase in DAD) focuses on laying a firm foundation in order to ensure iterative construction. One of the recommended key activities in this phase is to identify common tools, facilitate training, and implement techniques to avoid suboptimal usage of tools. This is because consistent and optimal usage of common tools improves team velocity.

3. Infrastructure for Communication and Coordination Is Crucial

Team members of geographically distributed teams depend on phone calls, chat, email and videoconferencing for communication. Also, they depend on Web-based tools for Agile project management, issue tracking, defect tracking, and so on. It is crucial to have an infrastructure that supports distributed development in order to relieve teams from technical issues related to communication and coordination.

The Agile Manifesto values "Individuals and interactions over processes and tools." Geographically distributed teams need the right infrastructure, which includes tools and protocols, in order to communicate and coordinate. Therefore it must be an ongoing goal for teams to leverage and enhance their existing infrastructure, and DAD includes this goal.

4. Knowledge Management Is Key to Success

Assimilation, creation, dissemination, and regular upkeep of knowledge related to technology as well as the domain elements of a project are critical to success in distributed projects. In distributed Agile projects, knowledge management (KM) becomes even more important because of Agile's emphasis on delivering meaningful solutions over short iterations while responding to changes coming from business users.

Knowledge management starts with learning and ends with efficient learning enablement. While Agile methods promote learning through such means as retrospectives, product demos, and the like, geographically distributed teams require additional formal mechanisms to foster knowledge management. What can distributed teams do to initiate and sustain KM? They can implement a set of KM practices. One such practice is to hold regular knowledge-sharing sessions that promote both technical and domain learning. Another practice is to create "knowledge nuggets" in the form of concise documents or wiki pages. These can benefit remote teams as well as new members of a local team.

Lack of focus on KM causes serious issues during unforeseen situations such as attrition. Individuals who join the team as replacements for departing team members struggle to understand the project domain and technology. They consume significant time from veteran team members, resulting in a loss of team productivity. A systematic and consistent focus on KM improves the ability to assimilate new team members in order to expand teams and manage attrition effectively.

5. Quality Is Multidimensional and Owned by Everybody

Quality can be seen in terms of intrinsic (or internal) quality and external quality. External quality is an attribute that relates to the end-user experience. External quality can be assessed and improved through black-box testing and defect prevention. Internal quality is invisible to the end users but visible to various groups in the development team such as designers, developers, maintainers, and technical reviewers. Internal quality can be assessed through reviews and static analysis. Internal quality can be improved by means of defect prevention as well as defect detection followed by analysis and fixing of bugs and other code quality issues.

Quality can be improved from different dimensions or streams of activities such as requirements inspection, design reviews, functional testing, performance testing, security testing, compliance testing, exploratory testing, and so forth. Agile teams understand this multidimensional aspect of quality and value a whole-team approach. Obviously, the set of metrics or measures used to understand the progress of projects needs to be multidimensional. Also, in a distributed project situation, team members from every location have to demonstrate a relentless focus on quality.

This leads us to the fact that distributed teams cannot thrive when there are double standards. A typical example of a double standard is when one team does not practice automated unit testing but expects another team to stay up to date on maintaining unit test scripts. This is detrimental to teamwork as well as the quality of deliverables. Again, quality is multidimensional and owned by everybody.

Distributed teams cannot thrive when there are double standards.

6. Distributed Agile Requires an Inclusive Approach

More than allocating functional modules or user stories across sites, distributed Agile teams need to consider an inclusive approach in order to nurture distributed ecosystems. There are several practices that align with this principle. For example, facilitating a "base camp" at the beginning of the project at a central location is the first step in ensuring inclusion and putting the right foot forward.[5]

Setting up the base camp involves forming a seed team with at least one team lead, one or two technical leads, and a handful of engineers. Typically, members of this seed team are selected from distributed locations. They come together and, depending on the size of the project, spend four to eight weeks at the location from which the project initiates. The objectives of setting up this base camp are visioning,[6] understanding the high-level scope, deciding on team structure and composition, identifying tools and setting up the environment, architecture envisioning, and so on.

Establishing a base camp offers several benefits. On the project execution front, it provides an opportunity to achieve adequate clarity on the technical environment, tools, and key engineering processes. On the people front, it enables rapport building that promotes efficient resolution of issues and conflicts during the project. In DAD, the Inception phase provides distributed teams with a similar opportunity and includes clear guidelines and goals.

The next practice is to allocate funds for team members to travel across sites at regular intervals and to facilitate such travel plans. Implementing distributed test drives or reviews, distributed retrospectives, and distributed root cause analyses is also a way to nurture inclusion.

Aiming for instantaneous results from the first iteration of a distributed Agile project is an unrealistic expectation.

7. Governance Is the Backbone of Successful Distributed Teams

Distributed teams cannot function without a collaborative governance mechanism. In the case of projects executed across multiple geographic locations and time zones with employees of the project sponsor organization, external vendors, and independent contractors, the complexity of governance increases multifold. Hence it is absolutely essential to form a governance team that comprises representatives from all locations and works together as a single body in order to run distributed projects successfully. Governance has been one of the key success factors in distributed projects.

Even though every project needs well-defined milestones and goals, it is critical to define success criteria at the governance level. This helps distributed Agile governance teams understand project success in terms of a common set of parameters. Without this step, governance teams tend to focus on transactional issues and miss the big picture.

While it is imperative to have a long-term view of the future, it is equally important to focus on early wins. One way to accomplish this is to define success parameters beyond tested code and avoid extensive low-value documentation, which is subject to costly rework. To make this happen, distributed Agile governance teams must have a strong, visible commitment to the success of projects. Having a one-year roadmap and identifying milestones or events that can be measured every quarter is a way to promote early success and mitigate risks.

8. Automation Enables Sustainable Pace

Automation of engineering tasks such as build creation, test data creation, unit test execution, regression testing, test result analysis, and the like is necessary to avoid manual effort spent on routine tasks. With automation, team members get adequate time to focus on critical tasks that need manual intervention. The significance of automation is greater in distributed teams than in colocated teams. This is because a lack of automation initiatives in one location will impact the quality and schedule of dependent locations.

9. It Is Essential to Streamline the Payoff of Technical Debt

Distributed teams need to be aware, aligned, and organized in managing technical debt in order to deliver maintainable, robust software.[7] In his blog post "The Financial Implications of Technical Debt,"[8] Jim Highsmith articulates that the financial impact of technical debt goes beyond the cost of fixing the debt; technical debt also has a big impact on the ROI of any project because it delays benefits and makes the project costlier to implement.

There is no better way to start a project than by making sure that certain key activities are carried out in a disciplined manner in order to avoid the accumulation of technical debt. Methodologies such as DSDM and DAD include such practices. Good examples from DAD are architecture envisioning in the Inception phase and regular focus on modeling on an as-needed basis in the Construction phase. Architecture envisioning is an essential practice for avoiding technical debt to begin with. Architecture prototyping is another technique that reveals technical risks up front and hence enables the early payoff of certain types of technical debt, which otherwise would involve lot of rework at a later stage of the project.

10. Ensuring Early Success Is a Collective Responsibility

Aiming for instantaneous results from the first iteration of a distributed Agile project is an unrealistic expectation. Nevertheless, achieving early success is both vitally important and the collaborative responsibility of project teams as well as governance teams.

During the initial stages of a distributed Agile project, the progress of iterations is very significant, and it happens in the form of issue resolution, continuous improvement, and the formulation or revision of policies among remote teams. The best way to start the first iteration is by including user stories that are simple to implement and not necessarily critical to the business. This will enable the teams to accomplish the goals of the first iteration with relative ease. Performing iteration-end process reviews along with retrospectives during the first four to six iterations will help ensure positive progression and hence early success.

A lack of focus on achieving early success can lead to severe issues, misunderstandings, and an absence of confidence in the project delivery model. On the other hand, consistent focus on ensuring early success in distributed Agile projects introduces positive reinforcement in project teams, motivates team members, and boosts performance. As I noted above, ensuring early success is a collective responsibility.

MOVING FORWARD

Over the past 10 years, industry experts and methodology gurus have spoken and written about their experiences with distributed Agile and shared their practices. We are entering an era in which enterprises — including large companies in banking, insurance, retail, and other sectors — are adopting Agile in a significant way. This means an increasing adoption of Agile practices by geographically distributed teams. In order to nurture a community of practice in distributed Agile and promote knowledge sharing, a group of like-minded professionals (including myself) founded the Global Distributed Agile Consortium in 2013 and started sharing our thoughts through the Global Distributed Agile Consortium Blog.[9] We foresee a large number of geographically distributed teams joining the Agile bandwagon over the next decade. The 10 principles discussed in this article align with process frameworks, such as DAD, which can be tailored to suit distributed ecosystems. I am confident that adhering to these principles and considering a principle-centered approach to Agile software development will enable geographically distributed teams to deliver results.

ENDNOTES

[1]Woodward, Elizabeth, Steffan Surdeck, and Matthew Ganis. *A Practical Guide to Distributed Scrum*. IBM Press, 2010.

[2]Hoda, Rashina, Philippe Kruchten, James Noble, and Stuart Marshall. "Agility in Context." *Proceedings of OOPSLA '10*. ACM, 2010.

[3]Disciplined Agile Delivery (www.disciplinedagiledelivery.com).

[4]Boehm, Barry, and Richard Turner. "Observations on Balancing Discipline and Agility." *Proceedings of the Agile Development Conference (ADC '03)*. IEEE Computer Society, 2003.

[5]Bavani, Raja. "Critical Success Factors in Distributed Agile for Outsourced Product Development." *Proceedings of the International Conference in Software Engineering (CONSEG-09)*. Computer Society of India, 2009.

[6]Highsmith, Jim. "What Is Agile Software Development?" *CrossTalk*, October 2002.

[7]Bavani, Raja. "Distributed Agile, Agile Testing, and Technical Debt." *IEEE Software*, Vol. 29, No. 6, November 2012.

[8]Highsmith, Jim. "The Financial Implications of Technical Debt." Jim Highsmith (website), 19 October 2010 (http://jimhighsmith.com/the-financial-implications-of-technical-debt).

[9]*Global Distributed Agile Consortium Blog* (http://blog.distributedagile.org).

Raja Bavani is Chief Architect of Mindtree and plays the role of Agile Evangelist. Mr. Bavani has more than 20 years' experience in the IT industry and has presented papers at international conferences on topics related to code quality, distributed Agile development, customer value management, and software estimation. His areas of interest include global delivery models, Agile software development, requirements engineering, software architecture, software reuse, customer value management, knowledge management, and IT outsourcing. Mr. Bavani is a member of the IEEE and the IEEE Computer Society and regularly interfaces with educational institutions, offers guest lectures, and writes for technical conferences. Mr. Bavani can be reached at raja_bavani@mindtree.com; blog: www.blogs.mindtree.com/author/raja-bavani.

Cutter
IT Journal

About Cutter Consortium

Cutter Consortium is a truly unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and Agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts, experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats, including content via online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

The Cutter Business Technology Council

The Cutter Business Technology Council was established by Cutter Consortium to help spot emerging trends in IT, digital technology, and the marketplace. Its members are IT specialists whose ideas have become important building blocks of today's wide-band, digitally connected, global economy. This brain trust includes:

- Rob Austin
- Ron Blitstein
- Tom DeMarco
- Lynne Ellyn
- Israel Gat
- Vince Kellen
- Tim Lister
- Lou Mazzucchelli
- Ken Orr
- Robert D. Scott