Agility at Scale: Become as Agile as You Can Be

By Scott W. Ambler
Chief Methodologist/Agile & SOA, IBM Rational

Agile software development is an evolutionary, highly collaborative, quality-focused approach to software development where potentially shippable working software is produced on a regular basis. Agile software development processes include Scrum, Extreme Programming (XP), Open Unified Process (OpenUP) and Agile Data (AD), to name a few. Although agile approaches are often equated to the development of Web-based applications, in reality they're also being applied to mobile applications, fat-client applications, business intelligence (BI) systems and even mainframe applications. They're being applied by a range of organizations, including but not limited to e-commerce companies, financial companies, manufacturers, retailers and government agencies.

Agile software development techniques have taken the industry by storm, with 69 percent of organizations reporting that they had one or more agile projects under way [1]. Agile is becoming widespread because it works — organizations are finding that agile project teams, when compared to traditional project teams, enjoy higher success rates, deliver higher quality, have greater levels of stakeholder satisfaction, provide better return on investment (ROI) and deliver systems to market sooner [2].

This eBook begins with an overview of the Agile Process Maturity Model (APMM), a framework we use to provide context to the plethora of agile methodologies available today. It then describes each of the three maturity levels in detail — agile, disciplined agile and agility at scale — showing how they build on one another. Then it works through an example of the development of an online bartering system and finishes with some advice for successful agile adoption.

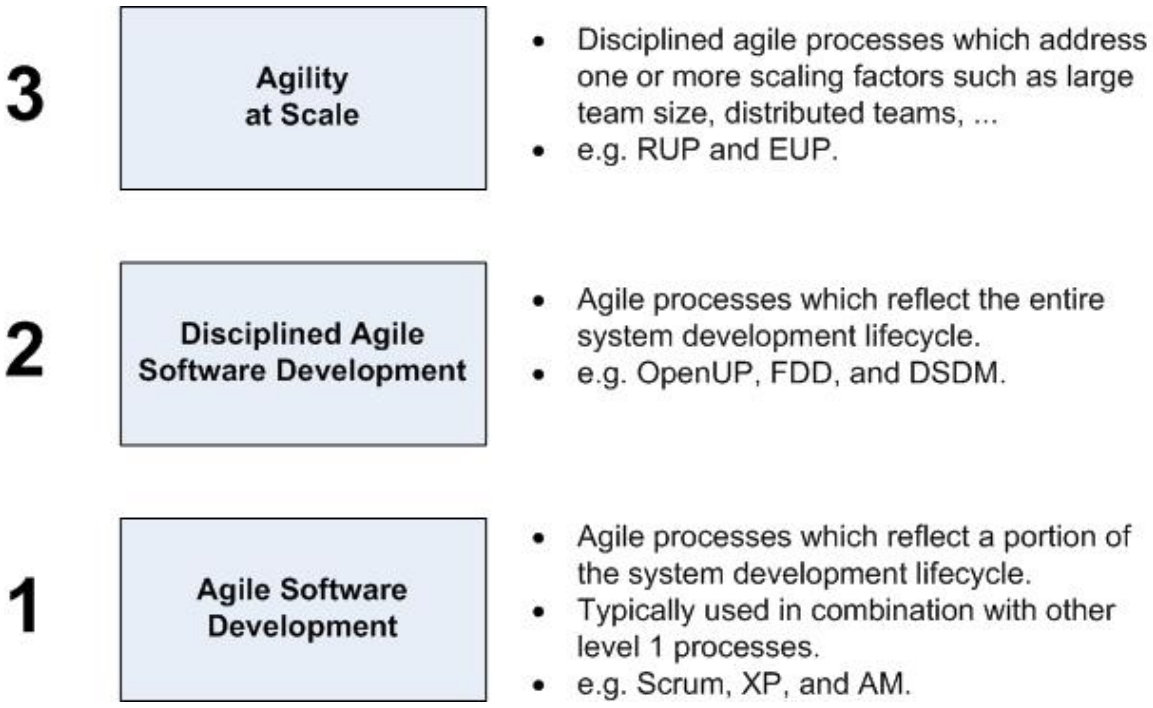
The Agile Process Maturity Model (APMM)

Within the software process realm, "maturity" is a loaded term, not the least because of the Software Engineering Institute (SEI)'s Capability Maturity Model Integrated (CMMI) [3]. A lot of good work has been done to show that agile and CMMI can be applied together [4], and I look forward to seeing that strategy come to fruition. However, whereas the goal of the CMMI is to provide a framework for software process improvement, the goal of the Agile Process Maturity Model (APMM) is much more modest — it merely strives to define a framework that can be used to put the myriad agile processes into context.

Figure 1 shows an overview of the APMM, depicting how its three levels build upon each other. The levels are:

1. Level 1 includes agile processes that address only a portion of the development life cycle. Examples include Scrum and my own Agile Modeling (AM).
2. Level 2 processes go further by covering the full agile system delivery life cycle (SDLC). This includes disciplined agile software delivery processes, such as Dynamic System Development Method (DSDM) and the Open Unified Process (OpenUP).
3. Level 3 addresses disciplined agile delivery processes applied at scale, which include tailored forms of level 2 processes as well as Enterprise Unified Process (EUP). The scaling factors which an agile team may face to a lesser or greater extent includes team size, physical distribution, and regulatory compliance to name a few.

Figure 1. The agile process maturity model



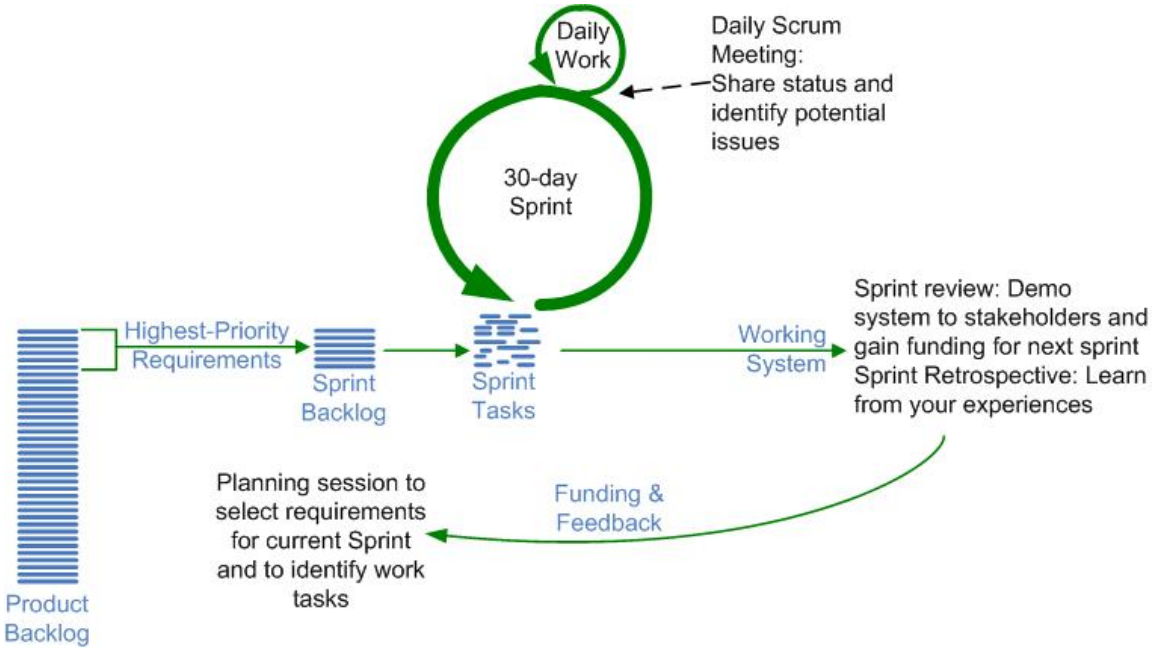
The following sections describe each level in greater detail.

Level 1: Agile Software Development

Level 1 agile methods address a portion of the software development life cycle. They conform to the values and principles of the Agile Manifesto [5, 6 written in 2001 by a group of 17 software development experts, and are typically described as a cohesive collection of practices. Examples of Level 1 agile processes include:

- **Scrum**. The focus of Scrum is project leadership and requirements management. Scrum defines a high-level life cycle for construction iterations (what Scrum calls “sprints”), see Figure 2, and several practices such as a daily stand-up “Scrum” meeting, product owner, product backlog, iteration/sprint planning and potentially shippable software.
- **Extreme Programming (XP)**. XP is a collection of practices for software construction, include refactoring, test-first design, pair programming, on-site-customer, continuous integration, whole team and collective ownership.
- **Agile Modeling (AM)**. AM is a collection of practices for light-weight modeling and documentation, including requirements envisioning, executable specifications, active stakeholder participation, prioritized requirements, and prove it with code.
- **Agile Data (AD)**. AD is a collection of practices for database development, including agile data modeling, database testing, database refactoring, and continuous database integration.

Figure 2. Scrum construction life cycle



Level 2: Disciplined Agile Software Delivery

Level 2 agile processes extend Level 1 to address the full system delivery life cycle (SDLC). As the criteria suggest (see sidebar: Are You Really Agile?), they also tend to “dial up” certain aspects of agile development, such as testing, measurement and process improvement. Disciplined agile software delivery is an evolutionary (iterative and incremental) approach that regularly produces high-quality software in a cost-effective and timely manner via a risk- and value-driven

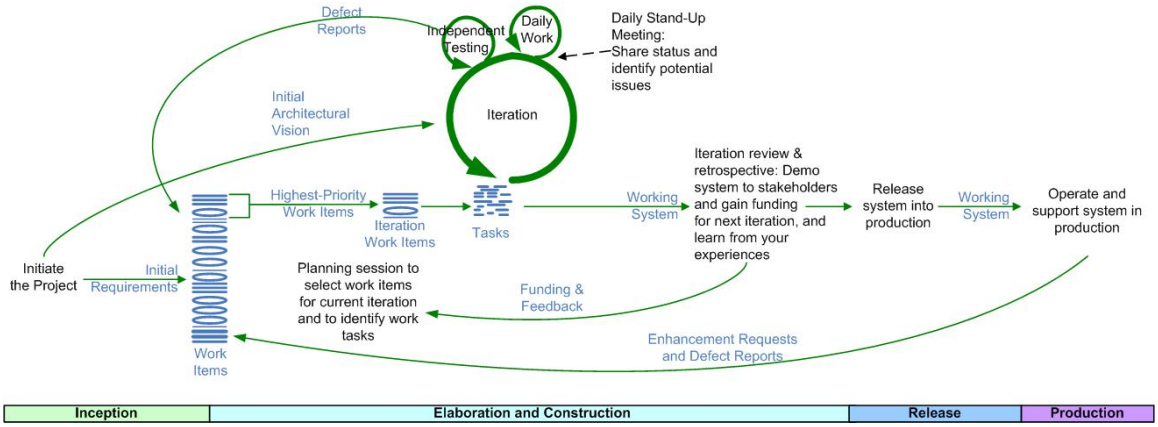
life cycle. It is performed in a highly collaborative and self-organizing manner, with active stakeholder participation to ensure that the team understands and addresses the changing needs of its stakeholders. Disciplined agile delivery teams provide repeatable results by adopting just the right amount of ceremony for the situation they face.

Examples of Level 2 agile processes include:

- **Rational Unified Process (RUP)**. RUP is a comprehensive process framework for iterative software delivery that can be instantiated anywhere from a very agile form to a very traditional form as your situation warrants [5]. RUP practices include risk-value life cycle, whole team, test-driven development (TDD), business process sketching and continuous integration.
- **Open Unified Process (OpenUP)**. OpenUP, the definition of which is available via open source, combines and extends practices from Scrum, XP, AM and RUP for co-located agile teams that are building business applications. OpenUP practices include whole team, daily stand-up meeting, prioritized work items, risk-driven life cycle, TDD, active stakeholder participation and continuous integration.
- **Dynamic System Development Method (DSDM)**. DSDM is an agile delivery process originally based on Rapid Application Development (RAD), which is often used to develop user-interface intensive applications. DSDM practices include prototyping, test throughout the life cycle, reversible changes and feasibility study.
- **Feature-Driven Development (FDD)**. FDD is a model-driven, short-iteration, agile software delivery process. FDD practices include individual class ownership, domain object modeling, development by feature, feature teams and regular builds.

Figure 3 depicts a high-level view of a full agile SDLC [6], which expands upon the Scrum construction life cycle in several important ways. First, it includes an explicit project inception phase where you do some initial modeling, start putting together your team, and gain initial project funding. Second, it includes parallel independent testing to ensure that defects don’t fall through the cracks, particularly problems associated with integration or nonfunctional requirements. Third, it extends the product backlog concept to include not only functional requirements but also defects and other work items such as providing feedback on work from other teams, taking training courses, and so on. Fourth, it includes explicit transition/release and production phases.

Figure 3. Agile system-development life cycle



Sidebar: Are You Really Agile?

A common problem in many organizations is that undisciplined "ad hoc" teams will claim to be agile, often simply because they're not writing any documentation and have read an article or two about how cool agile is. Unfortunately, those ad hoc teams often run into trouble and give actual agile teams a bad name. I suggest the following criteria to determine whether a team is agile:

1. **Working software**. Agile teams produce working software on a regular basis.
2. **Regression testing**. Agile teams do, at a minimum, continuous developer regression testing. Disciplined agile teams take a Test-Driven Development (TDD) approach.
3. **Active stakeholder participation**. Agile teams work closely with their stakeholders, ideally on a daily basis.
4. **Organization**. Agile teams are self-organizing, and disciplined agile teams work within an appropriate governance framework.
5. **Improvement**. Agile teams regularly reflect on, and disciplined teams also measure, how they work together, and then act to improve on their findings in a timely manner.

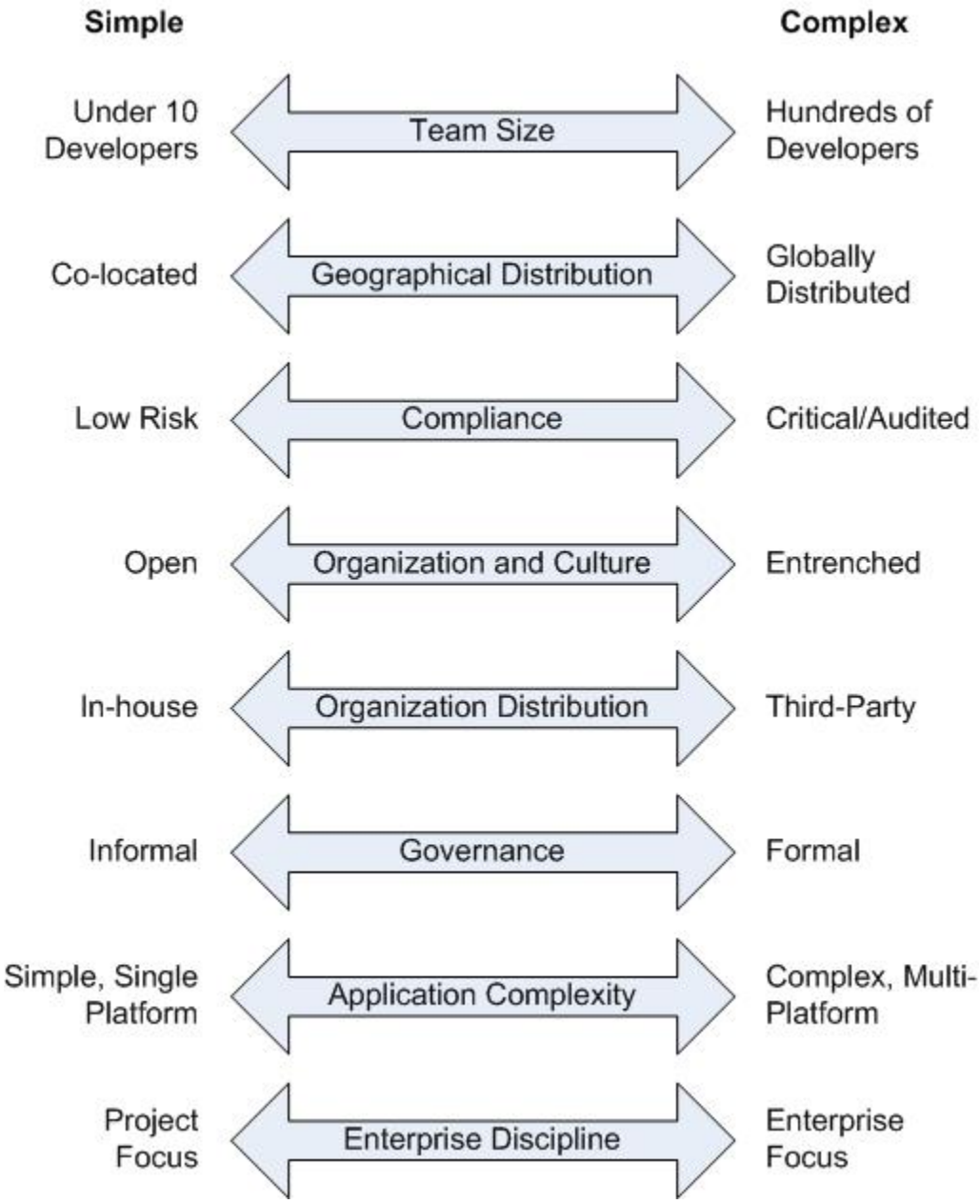
Several of the terms in the above criteria are underlined to indicate where your strategy needs to be flexible. For example, some agile teams will produce working software every two weeks, whereas others may be in a more complex situation and may do so only every two months. Different situations require different strategies, implying that one process size does not fit all.

Level 3: Agility at Scale

In the early days of agile, the applications where agile development was applied were smaller in scope and relatively straightforward. Today, organizations apply agile strategies to a broader set of projects. This is what Level 3 of the APMM is all about —explicitly addressing the complexities that disciplined agile delivery

teams face in the real world. Figure 4 gives an overview of the eight scaling factors of agile development.

Figure 4. Potential scaling factors for software development



Each factor has a range of complexities, and each team will have a different combination and therefore will need a process, team structure and tooling environment tailored to meet its unique situation. Level 1 agile processes on the APMM work best when basically all factors are at the left-hand side (the low-complexity side), although they can potentially be tailored to address greater complexity with strategies from higher-level processes. Level 2 agile processes

typically assume that one or more of the factors are slightly to the right, and Level 3 agile processes have one or more factors significantly to the right (the high-complexity side).

When it comes to tooling, many Level 1, and even some Level 2 teams, will find that they can make do with open source tools. But, when they find themselves in Level 3 situations, they soon discover that they need to adopt more sophisticated tools. To succeed at scaling agile, you will need tools that integrate easily, are sufficiently instrumented to provide the metrics required for effective governance, support distributed development, enhance collaboration between disparate team members, and automate as much of the work as possible to comply with regulations.

Agile Online Bartering

The best way to understand agile software development at scale is by example. SWA International, a fictitious company, wanted to extend its existing e-commerce offerings by adding bartering functionality. With the current economic times as they are, the company found that fewer people were online purchasing products from them. Offline, SWA had noticed that some of its customers were trying to barter, offering their services to help pay down their existing debt to SWA. The company felt the time was ripe for online bartering.

SWA put together a team that would eventually grow to 25 people, composed of two
subteams, one in Toronto and the other in San Francisco, plus several people working from their home offices in other cities. The company chose to follow OpenUP [7], a Level 2 disciplined agile process with two-week iterations. The first iteration was spent doing initial-requirements envisioning, with four senior business staff, four senior developers on the project, two operations staff members and the chief technology officer (CTO).

The goals were not only to understand the scope of the new system but also to come to stakeholder concurrence regarding what that scope was, thereby reducing business risk. In parallel, the CTO and senior developers did high-level architecture envisioning to identify the subsystems and their interfaces, critical information required to split the work appropriately between the two subteams — distributed agile teams prefer to organize themselves around the architecture instead of around a job function (e.g., having the developers in one location, the testers in another, the modelers in another and so on). One member of the business staff was an expert in regulatory issues, an important issue because SWA was entering a new line of business and publicly traded in the U.S. (hence, Sarbanes-Oxley compliance was an issue). Two of the businesspeople became product owners, one for each subteam, for the rest of the project. The product owner is the person who prioritizes the requirements and provides detailed information about the business to developers.

The second iteration focused on developing an end-to-end working skeleton of the system to prove that the architecture actually works. To simplify this effort, the work was performed in Toronto, the headquarters for SWA International, with three members of the San Francisco team involved. The team built a simple Web page that allowed an end user to select an item from a list and then make a bartering offer for it. The offer was then persisted by the system, and an e-mail notice was sent to the owner of the item for which the offer had been made. The team ran load tests showing that the system worked with 500 concurrent users for a period of 10 hours — a critical performance requirement for the system. At the end of the iteration, they demonstrated the system to the original stakeholders who had been involved in setting the requirements envisioning for the previous iteration, proving that the architecture for the system worked. At the end of the first month, they had reduced both their business and technical risks substantially.

For the next four construction iterations, the work was split between the Toronto and San Francisco teams, with the Toronto team focusing on the financial processing (SWA takes a fee for each swap), tax calculation and remittance, and management reporting, while the San Francisco team focused on the trading functionality. Both teams needed to coordinate their work on a daily basis, with the architecture owner on each team serving as the primary contact point for technical issues (often putting together the right technical people on each team to resolve an issue), and the product owners coordinating the requirements. A person living in Maui was the “independent test team,” integrating the entire system every day and performing exploratory testing on it to look for bugs the developers had missed. Even though the development team was doing comprehensive regression testing of its own, many problems still got past them, so the defects the independent tester found were reported back to the development teams via their shared bug-tracking system. The first version of the system was delivered into production after three months, and subsequent releases every six weeks (three iterations) after that.

Become as Agile as You Can Be

Many organizations have been successful at adopting agile software development approaches in part because the greatest focus until now has been on pilot projects or on a handful of projects within an organization. However, successful process improvement across an entire organization can prove difficult to implement in practice, often simply because by casting a wider net you run into a wider range of challenges. I’ve found that the following strategies can help increase your chances of success at improving your software process.

1. **Recognize that the true goal is to improve**. The reality is that nobody is going to give you a little gold star for being agile. They might, however, reward you for becoming more effective at system delivery. Agile

techniques can often help with this, but we need to remember that there are still some pretty good ideas out there in the traditional community, too.

2. **Have a plan**. For your process-improvement efforts to be successful, you should first determine what your goals are, what your current situation is and what challenges you face.
3. **Gain some experience**. Adopt agile approaches on one or more medium-risk pilot project(s) to gain organizational experience as well as build expertise among your staff. It's important to expect to run into a few problems because pilot projects never go perfectly.
4. **Explicitly manage your process-improvement efforts**. A common agile strategy is for a team to reflect regularly on its approach so as to identify potential improvements, and then to act on those improvements. Teams that explicitly track their progress at adopting improvements are more successful than those that don't [8].
5. **Invest in your staff**. You need to train, educate and mentor your staff in agile philosophies, processes, practices and tooling. Focus on the people involved with the pilots at first and train them on a just-in-time (JIT) basis. Don't forget senior management, project management and anyone interfacing with the pilot team — because these individuals need to change the way that they work, too.

Parting Thoughts

Many organizations have succeeded at applying agile at scale, and you can, too. If you keep your wits about you and stay away from some of the rhetoric of the Level 1 folks, you should be okay. Minimally, you want a disciplined agile system delivery approach (Level 2) that addresses the full life cycle, not just parts of it. Remember that you will often find yourself in a scaling situation (Level 3), and that different teams will experience differing scaling factors, and that's okay because it is fairly straightforward to scale agile strategies with effective practices and tooling. With a realistic approach to process improvement and with a bit of help from the outside, you can increase your return on investment (ROI), quality, stakeholder satisfaction and time to value through agility at scale.

References and Suggested Resources

1. Dr. Dobb's Journal's 2008 Adoption Survey. www.ambysoft.com/surveys/agileFebruary2008.html
2. Dr. Dobb's Journal's 2008 Project Success Survey. www.ambysoft.com/surveys/success2008.html
3. Capability Maturity Model Integrated Home Page. www.sei.cmu.edu/cmmi/
4. Glazer, H. , Dalton, J. , Anderson, D.J., Konrad, M., Shrum, S. (2008). CMMI® or Agile: Why Not Embrace Both! www.sei.cmu.edu/publications/documents/08.reports/08tn003.html

5. Kroll, P. and MacIsaac, B. (2006). Agility and Discipline Made Easy: Practices from OpenUP and RUP. Boston: Addison-Wesley.
6. Ambler, S.W. (2005). The Agile System Development Lifecycle (SDLC). www.ambysoft.com/essays/agileLifecycle.html
7. The Open Unified Process (OpenUP). www.eclipse.org/epf/downloads/openup/openup_downloads.php
8. Kroll, P. and Krebs, W. (2008). Introducing IBM Rational Self Check for Software Teams. www.ibm.com/developerworks/rational/library/edge/08/may08/kroll_krebs/index.html
9. IBM Agile Development — Rational Home Page. www.ibm.com/rational/agile/
10. The Agile Manifesto www.agilemanifesto.org
11. The Principles Behind the Agile Manifesto www.agilemanifesto.org/principles.html

About the Author

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