Agile Configuration Management for Large Organizations

By Peter Schuh

Updated: August, 2006

Executive Summary:

No development manager would argue that finding better and faster ways to deliver high-quality products to market is a worthwhile goal. In their quest for speed and efficiency, many project teams have effectively employed Agile practices and techniques because they create a development mindset that is focused on low-defect, high-quality software that can be delivered in small and frequent releases.

In my experience, small companies have been more willing to embrace an Agile approach than large enterprises. Arguably, part of this is because large companies tend to be slower and more resistant to change, but there is also a perception that Agile is too risky or that there are insufficient quality controls to make Agile Development viable for large, complex projects with distributed teams.

In reality, the opposite is true. Large companies need to embrace Agile Development principles precisely because of their large size and complexity. Left to their own devices, massive development efforts become slow and silo'd. Defects appear more frequently due to disconnected toolsets, teams and reporting structures, and vast geographies that make communication and coordination difficult and time-consuming. Practically speaking, the right hand doesn't know what the left hand is doing. Given these challenges, large development teams must make a conscious effort to establish a foundation of configuration management that will enable an automated, flexible, integrated and connected approach.

The keystones that make up this Agile CM foundation are a robust source control system, an automated and repeatable build system, a test automation framework, and a reliable deployment approach. Most importantly, all of these components should be integrated so projects can progress seamlessly and rapidly through the development cycle. Finally, information must be shared across these various teams to ensure that miscommunications and poor handoffs do not create project delays.

There is a growing wealth of data that demonstrates the profound benefits that an Agile process can bring to the large enterprise. However, few people would argue that the large company must be more deliberate in its adoption of Agile techniques. Automated systems are essential for complex, high-volume, distributed development environments to be successful. When determining the systems that will serve as the foundation for an Agile initiative, companies must be careful that the overall architecture facilitates flexibility, scalability, security, and self-documentation to ensure the success of their Agile initiative over the long term.

Large companies that must be concerned with issues such as integration of disparate environments, scalability, traceability and security may find it more economical and team efficient to consider using off-the-shelf products to assist in the implementation of an Agile CM approach rather than building and maintaining their own system. One product, IBM Rational Build Forge, can be used to rapidly enable an Agile approach in large development organizations. Build Forge establishes a framework that allows builds and releases to be automated and executed consistently, even in globally distributed development environments. It integrates the code-build-test-deploy functions so projects can flow smoothly from one group to another, and captures information across the development cycle to help make informed management decisions. It also provides a self-service capability that allows team members to execute builds, view results, and diagnose errors on their own, removing bottlenecks from the process. Their solution has been validated in some of the largest and most complex development environments, and has delivered significant productivity and quality gains.

Introduction:
The Case for Agile Practices in Large Development Organizations

Early in my consulting career, I was fortunate to be placed on a project that was experimenting with a new approach called Extreme Programming. The project environment was a rather typical candidate, twenty people with limited complexity and platform requirements. Overall, the project was a success. The new approach helped us to deliver on time and with fewer defects. After that project, I found myself in a variety of situations that were more challenging to an Agile approach, including a large project team (one hundred plus) and fixed cost work. A common theme in all of these experiences is that they were all single project teams, even though some were significantly larger than others. Despite these experiences, I did not fully appreciate the benefit that Agile Development can bring to an organization – specifically, the Agile practices and techniques related to configuration management – until I spent almost two years working with several project teams in a prominent Fortune 100 company.

When I arrived as a consultant to this company, I found an entire development organization where there was no routine use of source control, much less any automated builds or automated unit testing. It took several months of hard work just to set one project team on a more Agile course, with Continuous Integration, short iterations, and various other Agile practices and techniques. When that project was successful, team members helped guide other teams towards an Agile approach. After eighteen months, we had six projects following the major Agile configuration management practices. Each project team had its own codebase, but everyone was sharing components, tests, and build processes. The programmers were checking in code multiple times a day, adding automated tests just as quickly, and recompiling even after writing only a dozen lines of code. Project teams were running their own automated test suites, as well as those of other systems, multiple times a day. A significant portion of the organization was beginning to benefit from more robust code bases, more timely deliveries, and, ultimately, better end result products.

Since that time, I've been approached by many development teams within large organizations wanting to know if Agile is right for them. They've bought into the misnomer that Agile is disorganized, chaotic, and risky. Nothing could be further from the truth. My own experience has proven that Agile practices and techniques can provide a dependable and flexible configuration management environment that is vital for the large organization to sustain competitiveness and meet their quality objectives.

In this paper, I will present some of the basic building blocks of Agile configuration management, detail how these practices may be used to benefit large development organizations, and discuss how one particular product, IBM Rational Build Forge, may be used to rapidly enable an Agile approach in a large development organization.

Before We Start: Clarifying Terms

The profession of software development is both very good and very bad about its use of terminology. That is, we all do a great job of coming up with terms. Unfortunately, we typically do a lousy job at agreeing on what they mean. Therefore, before continuing, we should take a few sentences to clarify the terms that we intend to use in this discussion.

Starting with the easiest first, a ***large development organization*** can take a number of different forms. It may, for example, be a large project, numbering a hundred or more individuals, that has been decomposed into a number of subsystems and subteams for planning and development purposes. It may, also, be a development organization that consists of numerous, interconnected systems and project teams. More generically, it is probably any development organization that chooses to describe itself with the term "enterprise." Large development organizations mean multiple teams and multiple codelines. Most often, these large organizations possess a variety of systems in different stages of development, production, or near-retirement; any number of databases, flat-file repositories, and other data sources, a flurry of different project schedules and mandates, and a throng of interested parties with varying needs and agendas. Quite often these large organizations are on the verge of—or even are the definition of—complexity.

At a high level, ***Agile Development*** is very easy to define. It is a term used to describe any development approach (typically in the guise of a known methodology or employed by an actual project team) that adheres to the values of the Manifesto for Agile Software Development[1]. These values, in brief, focus on individuals and interactions, working software, customer collaboration, and acknowledge change as an unavoidable and even valuable component of software development. This high-level definition, however, only describes what Agile teams value, not what they do. When I speak of what Agile teams do, I mean the practices and techniques that Agile teams follow, such as Continuous Integration, automated unit testing, and short iterations. In the Agile community there are large, ongoing debates over whether a team that follows Agile practices but rejects Agile values can carry the moniker *Agile*. The values are important, because they provide guidance on the appropriate implementation of Agile practices and techniques. However, for the purposes of this paper, I will sidestep this debate and use term *Agile* to identify those practices and techniques that Agile teams use.

And then there is ***configuration management***; a concept that can be as challenging to describe as *quality* and that has traditionally held a number of different definitions[2]. Everyone seems to agree that configuration management covers the identification of items within a system and the controlled change of both specific items and the system as a whole. A very narrow definition of configuration management may be satisfied by the implementation and proper use of any popular source control system. Meanwhile, a very loose definition might toss in nearly the entire project team and all its artifacts, including all code and activities meant to insure the correct operation of

[1] The Agile manifesto can be found at *http://www.Agilemanifesto.org*

[2] For an informative and sometimes amusing read, see the collection of definitions for configuration management begun by Bard Appleton on the CM Crossroads Wiki, at: *http://www.cmcrossroads.com/cgi-bin/cmwiki/bin/view.cgi/CM/SoftwareConfigurationManagement*

any part of the system, all change control activity, and even the tracking of any alterations in the day to day procedures of the team. For this paper, we'll take a somewhat middle-of-the-road approach to configuration management, by including any work done by programmers to organize the parts of the system, know the state of the system at any time, manage its evolution, and ensure the continued and proper function of the system throughout the development process.

The Need for Agile Practices in Large Companies:

Now that we've measured up our ingredients for this discussion, let's see how they mix together. First, while small projects may get away with spotty and informal configuration management practices, most readers will likely agree that a formalized configuration management approach is required for large development organizations. I make this statement, which six years ago might have been considered overly bold, based on my observations of the inherent issues faced by large-scale development efforts. When dozens (if not hundreds) of product components are in play, and you're dealing with hundreds (if not thousands) of developers, the probability for chaos, slow development cycles, and poor product quality is extremely high. Large systems simply become too complex too fast to be sustained with manual systems. In these organizations, automation, process control, change management, and team coordination are necessities to keep development on track,

Second, let's discuss the mix between Agile Development and configuration management. When Agile Development was a new and growing topic of interest among software development professionals who were looking to break with the old habits of slipped schedules, cost overruns and failed projects, no one was talking about the Agile approach to configuration management. Agile, however, does have quite a bit to say about what are good configuration management practices because Agile teams need sturdy and flexible codebases in order to be responsive to ever-changing business environments and customer needs. One way Agile teams do this is by requiring that the code is frequently integrated (typically several times a day) across the entire project. Another key principle of the Agile mindset brings testing in as a critical element of effective configuration management. On many Agile teams, all new code is covered by an automated unit test, and all unit tests are run every time a build is performed. A broken unit test is taken as seriously as a compilation error. As in any good configuration management process, Agile teams want to know the health of all their codelines. Furthermore, they work hard to keep the code from ever drifting far from a release-worthy state.

Finally, there is Agile Development and the large development organization. Any large organization really can benefit from the incorporation of some aspects of Agile Development. Granted, there are unique challenges for large development organizations, such as those related to communication and coordination between individuals and teams in addition to the raw logistics activity associated with multiple projects, systems and data sources. But these are problems that large organizations will face regardless of whether or not they are taking their queues from an Agile approach.

What does Agile Development have to offer the large organization? First, Agile can increase team efficiency by automating tasks that reduce the likelihood of human error and enable teams to do more work with fewer resources. Second, Agile can help large organizations improve quality and deal more effectively with change by accelerating the speed of feedback loops to development members so problems can be resolved more quickly. Third, Agile can encourage richer and more timely communication by replacing large (and quickly outdated) requirements documents with iterative planning, analysis and development activities that can be documented automatically by the systems themselves as code is being designed and written.

Finally, it is important to note that an Agile CM approach can be implemented at either the project or organizational level. There is no need to initially turn an organization on its head, since individual projects may be treated as test labs and idea incubators. Meanwhile, when Agile CM practices are implemented at the organizational level, the organization must be mindful of to allow each project team a sufficient level of flexibility and autonomy to implement the solutions that best fit its individual needs.

Agile Configuration Management Practices[3]

Streamlined processes and automation are the foundation of an Agile CM approach. Each activity (from checking in code to fixing a broken test) should be easy to perform and provide quick feedback to both the individual programmer and the entire team. Furthermore, Agile teams attempt to make these activities self-documenting. For example, an automated build need only be documented in its execution scripts. One can easily count the benefits of a collection of well-written automated build script over a manual process accompanied by a constantly out-of-date how-to document in Microsoft Word.

The practices that compose Agile CM have been identified for their usefulness in a wide variety of project environments – whether large or small, simple or complex. We will discuss the practices, themselves, in this section, and apply these capabilities to the specific needs of large organizations in the following section.

Source Control

This is the oft-forgotten critical component of Agile configuration management. Not forgotten because Agile teams do not use source control (they do). It's forgotten because most Agile teams assume that every project has a source control system *and* that every project uses it correctly. The average source control system comes with a host of goodies, such as versioning, rollback, tagging, and merge assistance. Even more important, however, source control provides a reliable *place of record* for all the code lines of a project team or development organization. This only happens, however, when every programmer is checking code into the system on a frequent basis. When I say this, I mean at least once a day. When this happens, a project always knows where to find the current system in its entirety. It is not scattered across several development workstations, or possibly a handful of tarballs located somewhere on a shared server. The current system (or something no more than a few hours old) is always what checks out of the source control system.

[3] Portions of this section have been adapted from my book, *Integrating Agile Development in the Real World*, where these practices all are described in much greater detail.

To reiterate, just because a project or organization has a source control system does not mean that that system will support an Agile CM approach. At one client where I managed several teams, the two hundred person development organization used a proprietary tool for organization-wide source control. But the system had a critical flaw: it took hours to perform a single check-in! Because of this, teams only checked in their code when they had to—prior to releasing to production. A common source control system can be of great benefit to a large organization, as I explain later, but this is the case only when individual programmers and teams can actually check code in and out in a time-efficient manner.

I'll make one final note on source control. The team should not merely version the code that it is writing; it must also version the process (or scripts) it uses to compile and test that code. This way, if the team ever needs to roll back the code, it will also be able to roll back the build and test processes required to make that code useful and useable.

Build Automation

An automated build is the first step a team can take towards assessing the stability of their current software or system. Furthermore, an automated build reduces the time programmers spend on unnecessary tasks and removes a bottleneck (namely, the team's reliance on a lone buildmaster or independent build team) from the development process, thereby enabling the team to respond faster to change.

The goal of this practice is to reduce the build process to a quick push-of-a-button activity that any programmer on the team can perform. This activity should include all the code related to the system, regardless of what component or interface a programmer is working on. At the same time, the system must compile quickly. Faster workstations, incremental compilation, and alternative compilers are all strategies that may be used to keep compile times short. For the individual programmer, the ability to quickly build the system while writing new code has a number of benefits. First, it helps programmers verify the correctness of the assumptions made while coding – for example, to verify that certain external APIs work as expected. Second, routine builds of the code guards against issues that may otherwise silently arise from recent check-ins by other programmers. Finally, it identifies unknown dependencies that may reside in "far off" portions of the system that rarely surface through local builds.

An automated build that is useable by the entire team will reduce the time that team spends chasing down compilation and convergence issues. Programmers no longer need to wait hours or perform a set of arduous tasks to confirm that newly written code compiles. Instead, moments after a programmer has written his code, he will know whether his new code will integrate with everything that has been written before. This means compilation and integration errors will most often become apparent when they are introduced into the system, and can therefore be dealt with quickly and easily.

Finally, automated builds become more important the larger the system gets. Large projects that entail or connect with lots of other systems on a variety of platforms, especially, require some real strategic thinking to put together a build process that makes sense. And obtaining short compile times in these environments can be a challenge. Later in the paper we'll discuss some approaches to handling builds of larger systems—that is, when it simply is not practical for a single team to compile and test the entire system on a regular basis.

Automated Migration and Deployment

This is the next logical step following an automated build. The point of automating migration and deployment activities is to streamline and increase the predictability of promoting builds from development through testing environments and into production. Too often, a myriad of problems arise the first time a project has to coax its way into system testing, user acceptance testing, and then fumble its way into production. With automated migration, teams can regularly perform dry runs that deploy the code into a clean "production class" environment, where automated unit and system-level tests may be executed. By testing in a near-production environment throughout the development process, the team will identify environment, integration and even performance issues long before they make their way to system testing. This also makes the team much more familiar with the actual process of deploying to production. Finally, human error is largely removed from the equations, when the majority of the production deployment process is automated.

Another important point about migration and deployment is that these activities are typically managed by a dedicated team that may not even report into the development organization. Integrating these processes into the project team's everyday activity creates more effective and timely handoffs between teams. Admittedly, this can be a challenging activity for large enterprises because of their sheer size, reporting structures, and tendency to spread departments across multiple geographies. We'll discuss some more advanced solutions to this problem below.

Test Automation

Automated tests are the first step a project can take toward knowing whether the current system is release-ready. Automated tests should be written as an accompaniment to all new code in the system. Additionally, when old, untested code is modified, programmers should write new tests for that code. Finally, when a defect is discovered in acceptance testing or production, a test should be written to demonstrate the defect and the new test should be incorporated into the overall test suite to prevent the defect from happening in the future. When all the unit tests in the system pass successfully, a programmer should have a high level of confidence that his code is functioning properly and that he has done no harm to other portions of the system.

From an Agile point-of-view, automated unit-level tests are an extension of the build process. Every time a programmer runs the build process as has a successful code compile, a run of all the applicable unit tests should follow. In an Agile CM environment, a broken unit test is treated with the same seriousness as a broken build. This way, small problems do not fester into big ones. And big problems (such as an entire section of the application that is no longer functional) do not linger silently in the background until one week before the release date. Instead, the team agrees to address such problems as they arise, and the ever-growing suite of unit tests increases the likelihood that errors will be accurately detected a higher percentage of the time.

One thing I'll add about both unit- and system-level tests. Smart projects and organizations will spend some effort on *test data management*. This is the process of making test data easy to create, easy to modify, easy to maintain (while the system's data structures evolve), and easy to restore (as in, before every test). A huge suite of automated tests that relies on brittle and arbitrarily-devised data structures can quickly be brought to its knees when a major change is made to the data model or even, in some cases, when the development database is wiped clean.

Continuous Integration

Continuous Integration ties together the proper use of source control, an automated build process, and a trustworthy set of automated tests to provide the team with a high level of confidence in both the stability and proper functioning of the system under development. In a Continuous Integration environment, programmers are writing code, running the build and tests on their own workstations, and checking in multiple times a day. To keep everyone on the team honest, there is typically a stand-alone build machine (or group of machines) that compiles the entire system and runs all the tests in a clean environment that closely resembles the production configuration. This activity can be triggered either manually or automatically – either at fixed intervals or incrementally as programmers check in code. If, for whatever reason, the code does not pass the build and tests in this clean environment, the team is typically alerted less than an hour after the check-in. In many circumstances, Agile teams will mandate that no one on the team is allowed to check in additional code until the situation is resolved.

The benefit of Continuous Integration is that it instills a development discipline where individuals are discouraged from checking in poor quality code, and the team is committed to resolving errors immediately when they occur. Since code check-ins occur multiple times a day, programmers in a Continuous Integration environment rarely or never make changes to the code that will cause them to be more than a few hours from a stable build. Additionally, there's no checking in half-baked code, because the team's build machines will catch it within hours, not days or weeks. Therefore, programmers have to think through designs, and even test some ideas out, before they cut into the code. This means that programmers in a Continuous Integration environment are much less likely to perform their experiments directly within the system code, to half-write a piece of functionality and forget to complete it, or take the system apart and leave it all spread out on the garage floor. When errors are detected shortly after the code is written, resolution times are typically much faster than if they were identified days or weeks later. Over time, this practice results in higher quality code and more rapid release cycles.

Scaling Agile CM for Large Systems

Large organizations experience many of the same CM problems that small and medium-sized projects do, with a small heap of additional frustrations related to their multi-system, multi-project, multi-initiative nature. For example, while a small project may uncover a defect or experience an integration problem that may take the entire team hours or days to fix, integration issues and defects discovered across systems can cost large organizations weeks of time. Similarly, issues related to source control and versioning on a small project are trivial compared to the collective migraine that results when a system rollback or multiple-application redeployment is performed in a large organization that has no dependable configuration management approach in place. Due to sheer scale and complexity, proper configuration management practices are essential for large organizations.

The Agile approach to configuration management can help large organizations address configuration management issues while remaining more flexible to the changing needs of customers, evolving business climates, and ever-advancing technologies. Additionally, an Agile

CM approach can help new projects get up and running faster by reusing build processes from existing systems and projects.

In this section, I will discuss three topics related to using Agile practices in the large development organization. First, we'll discuss how an Agile CM process can be implemented flexibly to benefit both the individual project team and the large development organization. Second, how an Agile CM approach can be used on distributed projects in multi-site organizations. Finally, how the organization may increase economies of scale and accelerate software delivery through careful and deliberate integration of applications and processes within their development lifecycle.

Effective Team Coordination: Sharing Codebases and Chaining Builds

In everyday project environments there is often a need for teams to share code, depend on common libraries, and even share one another's build process. One project, for example, may need to incorporate the build and test activities of other projects. This could include obtaining updated versions of shared libraries to confirm that changes in other projects have not adversely affected the team's code, or validate that changes to the team's codebase will not adversely affect the functioning of another project's code. Furthermore, projects may share essential resources which either project may have the ability to update. Such resources may include common classes or other tools such as test harnesses and test data generators. These situations are common in large organizations, and they often happen without the knowledge or assistance of the organization itself.

An Agile approach provides the opportunity for development teams to collaborate more efficiently and engage in ongoing communication so projects progress more smoothly. By providing rapid feedback on build success/failure, developers are able to detect and resolve problems when they are easiest to fix.

To be effective within a large development organization, Agile practices must be implemented at the individual team level but should be sponsored and supported by a corporate lead configuration management best practices initiative. When implementing an organization-wide Agile CM approach, the individual team must take responsibility for several things. First, it must follow a reliable Agile CM implementation based on the practices discussed in the previous section. Second, it must make its processes available to other teams. Third, and when appropriate, it must include the build process of systems both up- and down-stream into its own build and testing activities. This final step does not need to be done by programmers during their daily activities, but it should be performed by an automated process that runs as often as possible (ideally between once-a-day and once-a-week). And, when an issue with another system does arise, this issue must be taken seriously and resolved quickly.

To help implement an Agile CM approach across all teams, the corporate CM organization will also have distinct responsibilities. These tasks may belong to a shared services entity (often called Engineering Services Groups). This organization may provide a common, and user-friendly, toolset or platform which all teams can employ to complete and share their source control, build and testing activities. This platform may include components such as source control, build, and testing systems. Furthermore, the organization should provide support and guidance for teams in the implementation and continued use of Agile CM practices, and may provide a set of reusable CM processes or best practice recommendations to bring consistency and reliability across project teams. Finally, the organization must ensure that every team still

has sufficient control of its own CM process. This may seem like a bit of a balancing act, but it is necessary for effective software delivery. Ultimately, it is still the individual team that is building the software and it is the job of the organization to help each project to maximize their success.

Supporting Distributed Teams and Organizations

It is true that many Agile methodologies were not created with distributed teams in mind, but this is an ever more common characteristic of large organizations, and one that simply cannot be ignored by any movement that aspires to alter the software development industry. Despite a lack of specific focus, an Agile CM approach can be remarkably useful to projects and organizations working within a distributed team environment.

To benefit from an Agile CM approach, distributed projects and organizations must leverage the solid implementation of several Agile CM practices, especially routine use of source control, Continuous Integration and automated testing. The importance of frequent check-ins and the maintenance of a stable build cannot be overstated, because teams split across time-zones and continents need to be confident that they will have access to a complete and operational version of the system on a daily basis. When something is broken or appears out of date, often, there will be no one available at a team's sister site to lend a hand.

In order to maintain an Agile CM solution for a distributed project, everything must be checked into source control, including build scripts and local environment settings. Any change that occurs at one site should automatically replicate over to the other development sites. This is needed because of the complexities associated with distributed teams and their typically large systems. Specifically, once a system in development begins to exhibit quirky behavior at one site and not another, days can be lost at both sites before the source of the issue is discovered to be some setting on the server or virtual machine that no one would have ever thought capable of causing such trouble.

Additionally everything related to the database needs to be replicated and shared. This may be accomplished by scripting all changes to the database and checking them into source control.[4] It might also be accomplished through some form of database replication. Finally, the project must account for any connected or third-party systems against which development activity must be performed. In this case, each site must have access to either the same or identical systems.

There are two general approaches that may be taken to implement a distributed and Agile configuration management environment. The first is built around a single development environment that is constantly accessible by all development teams. This environment would include—at least—a single source control system, all databases and connected systems, and the ability to perform continuous integrations. This solution can work very well for teams that work within nearby time zones and have dependable online access. The second approach is built on the concept of stand-alone development sites. Here, each team has a completely independent and identical development environment, including source control, databases, additional system installations, and continuous integration setup. A daily replication schedule must be put in place and adhered to in order to keep code, data and environment changes across all development sites in sync. As much as possible, synchronization activity should be automated. Furthermore, automated tests must be routinely written and executed. If daily replication and thorough testing

[4] For more information on setting up and managing a database in this fashion, consult my paper "Agility and the Database." This can be found at *http://www.peterschuh.com/articles.html*

are not performed (that is, if things are allowed to fall out of sync) the organization may find itself with a convergence nightmare on its hands. Finally, projects and organizations can also pursue a middle-of-the road solution—that is, where some portion of the Agile CM environment is centralized while the rest is maintained at the individual sites. For example, an organization may have common source control and build systems, but maintain local instances of the database and other third-party systems across its different development sites.

Scalability Through Flexible Integration of Tools and Process

If sufficient thought and preparation is put into the creation of good build processes and automation it can become a powerful development asset, and this infrastructure can (and should) be leveraged across multiple projects. A typical inefficiency that occurs in large companies stems from each project team creating a new build system for each software project. The result – multiple custom build applications to maintain with dedicated hardware resources and configuration management staff. This prevents large organizations from gaining economies of scale from the pooling of resources, staff, and best practices knowledge.

If an organization plans to implement Agile practices with any scale (meaning simultaneous code-build-test-deploy cycles across multiple teams, projects, and/or operating platforms), then serious thought should go into how these systems will communicate and interact to create a smooth code-build-test-deploy cycle. If cross-team, cross-system integration is not factored into the overall development strategy, teams often find that development progress is slowed by gaps, wait periods, and miscommunication between the functional silos. Without an infrastructure that tracks and aggregates information from each phase of the cycle, teams can find it difficult to determine the true health and state of a release.

Integration should include workflow automation as well as information sharing (or at least extraction) from your core development systems mentioned in the previous section. Workflow automation includes the order and orchestration of tasks, and should involve a rules-based capability to alter task execution and notification based on the success or failure of preceding steps. When determining your integration approach, teams should look to industry standard approaches (XML, etc) in order to architect a flexible solution that can adapt to changing needs and development applications.

A Solution to Consider: IBM Rational Build Forge

Historically, most build systems have been proprietary – that is, created, maintained, and supported by in-house development teams. This was due, in part, to the lack of viable alternatives (either commercial or open source) but also because compilation and integration are often an afterthought in the development process. Historically, the build "process" (implying more than just a successful compile, but rather a series of tasks that are required to deliver the final executable) has been viewed as a necessarily evil rather than an essential tool to let the team know whether they are on track. As a result, most in-house systems – while sometimes quite sophisticated – are typically an ad-hoc collection of scripts and e-mail communications that are highly error prone and time-consuming to maintain. Today, many companies are realizing that, like source control, defect tracking, and testing solutions, the build is an essential link in the overall process that must have a stable, reliable foundation.

This is not to say that internal systems cannot be effective, but they are typically very costly to develop, support, and maintain. This is especially true with large companies, where millions of dollars are expended annually to execute and maintain hundreds of builds for their large array of products. Open source alternatives that provide basic build functionality have recently become available, but these systems (such as Cruise Control) are project-minded and do not take into account many of the needs of a large enterprise system, such as:

- Supporting and linking multiple coexisting systems and project teams
- Sharing common components and build processes across multiple teams
- Supporting the security requirements of a large organization (for example, allowing employees and contractors to connect securely, as well as providing relevant information to each of these groups)

Unlike several years ago, today there are commercial build solutions that may provide a more cost-effective alternative for a stable, reliable, build and release system. While I have not set out to evaluate all of the products available, I have encountered one that I believe deserves a closer look – IBM Rational Build Forge. First known as the independent Austin Texas based company, BuildForge, Inc. it was acquired by IBM in May, 2006 and is now offered as part of Rational's suite of development products. Although Build Forge integrates out-of-the-box with Rational's Software Development Platform, it also supports and integrates with a variety of non-Rational tools and environments. Build Forge provides a foundation of automation, documentation, integration, and communication capabilities that are crucial to a successful Agile CM implementation in enterprise environments.

I will highlight some of Build Forge's capabilities that apply to successful Agile Development in the following paragraphs; however readers can refer to the Appendix at the end of this paper for a more detailed list of Build Forge functions and their usefulness in Agile environments.

Consistent Build and Release Process Management

The Build Forge system provides a framework for push-of-a-button build execution that I mentioned earlier in this paper. Complex build processes can be broken down into a series of discrete tasks (component builds, unit tests, file transfers, etc.) to eliminate the single point of failure that typically exists with large build scripts. As a result, builds execute in an unattended mode, and failures are identified as soon as they occur. The product has a web-based management console that serves as a centralized location for all build and release projects. Once in the system, build projects are run consistently each time, eliminating many of the errors that are associated with manual processes. Through this centralization, processes can be standardized and shared easily across different teams and projects, reducing setup times and establishing reusable best practices. This helps teams gain greater efficiencies and scalability, allowing them to accomplish more work with their existing resources.

End-to-End Lifecycle Support

Build Forge offers a unique approach for companies who wish to integrate their development cycle – from coding through build, test, and release. Large corporations typically use a variety of tools across their development teams, often involving several languages and IDE's, multiple source control systems, testing, and defect tracking products. As discussed earlier, in order to effectively streamline processes, there needs to be automated handoffs between these systems. Build Forge provides development teams with a high degree of flexibility by offering an open command line interface versus a proprietary scripting language. This is well suited for large organizations that have significant investments in existing scripts and dependency management tools, allowing these scripts to be plugged into the Build Forge system without requiring extensive migration work. Build Forge has created numerous application-specific adaptors that provide out-of-the-box integrations with popular source control, defect tracking, and test automation systems to help teams reduce implementation times. They also offer a developer's API that allows teams to create custom integrations to additional third party or proprietary tools. These integrations reduce implementation times and enable teams to achieve Continuous Integration more quickly by providing pre-built functionality that can trigger builds based on source repository activity and automatically promote successful builds into the test and release phases.

Scalable Build and Release Operations

For many large development organizations, Agile practices may appear unattainable simply by the sheer volume of projects, systems, and data sources that must be managed. Part of this burden is lessened through automation – the remainder must be addressed through more sophisticated methods. There are several important aspects here that Build Forge has addressed. First and foremost, Build Forge supports a diverse set of "Tier 1" platforms that are requirements for most large companies – including variations of UNIX, Linux, Windows, and Macintosh. More importantly, Build Forge enables companies to better manage their build and release workload across these platforms by placing hardware into "pools", and allowing projects to be assigned dynamically based on a server's availability and processing power. To achieve further performance gains, Build Forge allows non-dependent project tasks to be run in parallel using their "threading" capabilities. These strategies not only create efficiencies in hardware utilization, they allow development organizations to take their Agile strategies to the next level

by reducing build times significantly (as much as 4-20x based on examples I've seen) and dramatically increasing the number of development turns the team can achieve on a daily basis. Specific results are discussed in a later section.

Traceability Through Self-Documenting Systems

As mentioned earlier, a build and release processes should be self-documenting to avoid creating documentation manually that will quickly grow out of date. It should also include sufficient tracking so errors can be identified and resolved quickly. Inherent within Build Forge's framework is a self-documenting capability that captures all build and release activity into a detailed bill-of-materials. By tracking the complete process – from source checkout through the build, test, and release phases – the bill-of-materials provides a thorough manifest of the required tasks, systems, and individuals required to complete the development cycle. This data can provide important insights into process bottlenecks and quality issues that may allow Agile teams to tune processes and identify areas for improvement as they go. Furthermore, the product's integration with various source control, test, and defect tracking systems creates a detailed and accurate audit trail of release components, source changes, test results and defects resolved to help development, build and QA teams reproduce and resolve errors quickly.

Self-Service Increases Team Productivity

Lack of communication can kill any development initiative, especially an Agile one. Breaking down the walls between developers and the build team is one of the biggest steps you can take to become a more Agile organization.

How many times has your development team experienced the following situation – code works correctly on the developer's system only to break in the nightly integration build? What ensues is a frenzy of finger pointing and the build team searches frantically through log files to detect the errors and communicate back to the team so it can be resolved. In the meantime, members of development and QA sit idle – their progress stalled.

This is caused by two dynamics within the development process – inconsistent system configurations and inadequate communication between group members. Build Forge has taken specific steps to break down these barriers by providing capabilities for the development team to run pre-established build and release processes on their own, both before and after source check-ins. Using Build Forge's IDE plug-ins for Eclipse and Microsoft VisualStudio .NET, developers can be granted permission to validate their changes using approved CM processes in advance of a nightly run. This capability enables developers to execute partial or complete builds directly from their IDE in a clean and complete environment (which is often not feasible on their local machine). They receive immediate notification of errors and pointers to the log file data so they can resolve problems quickly. Build Forge also makes it possible to test code before it has been committed to the source control system by effectively redirecting part of a project's execution to their local code base. This allows developers to check in more frequently with greater confidence that their code will work correctly in the integrated environment.

Build Forge's self-service capabilities remove communication bottlenecks and enable developers to maximize their productivity since they are no longer dependent on one or more persons to be able to validate their code. This results in an automated, continuous flow of information between development, configuration management, and QA team members throughout each phase of the process to keep Agile projects on track.

Secure Access to Processes

One of the principal differences between small companies and large enterprises is the increased requirement for security. As companies move from internally built systems to commercial or open source alternatives, security implications must be considered. Build Forge has addressed this need through a built-in group definition and permission system that allows companies to control the access to information at a very detailed level. This role-based system enables administrators to limit the projects that can be accessed by different team members, and provides restrictions on what individuals can view, execute and modify – even down to subtasks within a project. The product also provides customizable SSL encryption capabilities so companies can ensure proper authentication for servers outside the firewall to their management console.

Some Industry Results

Results speak for themselves. As I work with companies who are considering moving to an Agile approach, I always recommend that a company focus more on achieving tangible results rather than trying to sell the organization on the merits of the theory. But let's face it – any departure from the norm involves a certain degree of faith. The trick is to make educated and calculated leaps based on the proven successes of companies that have gone before you.

The good news is that Agile approaches work and have been validated. Build Forge recently conducted customer interviews of companies who have adopted a more Agile and iterative approach to their product development process.

Consider the following results that were reported by companies within the first six months of implementing their automated build and release system in terms of the length and volume of their build activity:

Company	Before	After	% Improvement
Large telecommunications company	10 hours/wk	10 minutes/wk	60x
$1 billion dollar software provider	9 builds/wk	360 builds/wk	40x
Large electronic gaming company	60 hour builds	3 hour builds	20x
Large hardware developer	2.5 hour builds	12 minute builds	12x
Second large telecommunications company	8 builds/month	50 builds/month	6x
Mid-sized software company	6 hour builds	1.5 hour builds	4x

While a 4x improvement is a sharp contrast from the 60x improvement in the first example, consider the implications of a 4x improvement. The company is able to develop and deliver their products 4 TIMES FASTER than before—that is, a 400% improvement! This makes the 60x improvement even more staggering. Clearly, these levels of increased flexibility have a tremendous impact on making development teams more productive and getting products to market more rapidly, ultimately impacting the company's cost and revenue bottom lines significantly. One company estimated that their Agile initiatives will save approximately $25 million dollars annually.

Conclusion

There is a growing wealth of data that demonstrates the profound benefits that an Agile process can bring to the large enterprise. However, few would argue that the large company must be more deliberate in its adoption of Agile techniques. Automated systems are essential for complex, high-volume, distributed development environments to be successful. When determining the systems that will serve as the foundation for an Agile initiative, companies must be careful that the overall architecture facilitates flexibility, scalability, security, and self-documentation to ensure the success of their Agile initiative over the long term.

About the Author

Peter Schuh is the author of *Integrating Agile Development in the Real World*, a field guide for anyone in software development whose prime interest is to see useful and usable software built and delivered in a timely manner. He has held virtually every position on a software development project team, including project manager, account manager, coach, programmer, DBA, business analyst, technical writer. He has manage and coached IT projects in the financial, leasing, healthcare and e-commerce industries. When he is not working with project teams in the digital world of software development, he is rehabbing and managing small apartment buildings in Chicago, where he often plays the role of the customer in the physical world of construction and building maintenance. Peter can be reached at agile@peterschuh.com.

About IBM Rational Build Forge

For more information about Build Forge and its products, go to www.buildforge.com or contact sales@buildforge.com.

Appendix of IBM Rational Build Forge Capabilities

Recommended Agile Practice	Build Forge Capability	Value in an Agile Environment
Automating build tasks	Build Forge provides an intuitive interface to create and execute recurring build tasks.	• Fewer build errors • Faster Agile turns
Push-button build execution	Projects can be executed by authorized team members from the Build Forge Management Console without detailed project knowledge.	• Reduced training requirements • Fewer build errors • Faster Agile turns
Continuous Integration / iterative builds	Trigger automated code-build-test cycles on demand or based on source code changes.	• Identify and resolve errors more quickly
Integration between key development systems (source control, test automation, defect tracking)	Build Forge offers out-of-the-box adaptors with leading source control, test automation, and defect tracking systems.	• Rapid implementation of Agile processes • Increases automation • Improves team productivity • Supports distributed team collaboration
Flexible integration options	Build Forge provides a command line interface and an API toolkit to create custom integrations.	• Increases automation • Flexibility for individual groups to customize their Agile approach
Real-time access to information	Build Forge's Management Console provides a real-time view of release activity.	• Improves team communication • Supports distributed team coordination

Recommended Agile Practice	Build Forge Capability	Value in an Agile Environment
Rapid error detection/resolution	Build Forge allows teams to create customized error filters and provides automatic notification of build errors directly to team members.	• Faster problem resolution • Improves team productivity
Incremental builds	Build Forge allows developers to conduct "pre-flight" builds to test their code changes in an integrated environment. CM managers can also pause and resume a build in progress to resolve errors.	• Faster problem solution • Faster Agile cycles
Developer access to established build processes	Developers can be given access to run established build processes on clean production machines.	• Fewer errors in the integration builds • Improves team productivity • Removes dependence on the "lone buildmaster"
Retention of the build process	Build Forge stores and retains build process knowledge in a central location.	• Reduces errors through consistent, repeatable processes • Reduces training and dependence on the "lone buildmaster" to run processes manually.
Multi-project support	Build Forge supports processes across multiple projects from a single location	• Supports Agile processes in distributed or multi-team environments • Enables the creation of Agile best practices
Multi-platform support	Build Forge supports projects across multiple operating platforms from a single location, and allows multi-platform builds to occur simultaneously.	• Supports fast Agile turns in multi-platform environments
Secure access for distributed teams	Build Forge provides customizable security settings for different development roles to control who can view, run, and modify individual projects.	• Improves team communication and collaboration • Supports distributed teams

Recommended Agile Practice	Build Forge Capability	Value in an Agile Environment
Shared use of hardware resources	Build Forge creates hardware pools that allow teams to share hardware resources.	• Supports multi-project environments • Supports distributed teams
Reuse of build processes across multiple projects	Build Forge creates libraries of standard build/release tasks that can be shared across multiple teams and projects.	• Establishes Agile best practices • Improves team efficiency
Shared workload between teams	Build Forge provides centralized access to build and release processes so teams can more effectively share their workload.	• Improves team efficiency
Allows developers to customize processes to meet their project's needs	Build Forge provides flexible integration options that help teams determine the optimal build/release process for their project's unique requirements.	• Provides flexibility to improve Agile implementation success
Automated release/deployment capabilities	Build Forge provides smooth migration of build projects into deployment environments	• Improves team handoffs • Reduces production errors
Enables the creation of build and release best practices	Build Forge enables build and release processes to be standardized across multiple teams, products, and geographies.	• Improves Agile implementation success • Reduces build errors • Reduces build/release burden of specific teams

