Global Headquarters: 5 Speen Street Framingham, MA 01701 USA P.508.872.8200 F.508.935.4015 www.idc.com

WHITE PAPER

Establishing Build Management for IT Efficiency and Business Adaptability

Sponsored by: IBM Rational Build Forge (formerly BuildForge, Inc.)*

Melinda-Carol Ballou

January 2006

EXECUTIVE SUMMARY: UNDERSTANDING BUILD MANAGEMENT

Software build management impacts successful software deployments, as well as business and IT productivity, and is becoming an increasing focus for IT organizations. The need for more consistent, reliable, and high-performance build management processes; organizational strategies; and automation has always been important, but a number of factors are driving companies to target the build management area of the application life cycle now. The pressures on Global 2000 corporations include:

- Distributed development using offshore, outsourced, and internal resources that demand more effective build management because of the need for better communication, collaboration, and coordination
- Regulatory compliance initiatives resulting from legislation such as Sarbanes-Oxley that mandate auditability, consistency, and reproducibility from development to production, necessitating effective build management
- The complexity of emerging new development paradigms, such as service oriented architecture (SOA), that require close coordination across business and IT groups and a higher level of quality, change, and build management to deliver services

However, build management has typically been viewed as a minor subcategory within software change and configuration management (SCM), and both users and vendors tend to have inadequate approaches. Build and release management should be considered a distinct category and process within application life-cycle management (ALM), with a similar level of focus to other ALM phases. Not doing so jeopardizes successful software implementations.

The purpose of this white paper is to lay out the role that build management plays — its impact on the overall software development life cycle and the business adaptability enabled by more consistent approaches to build management.

In that context, we consider today's market drivers, the current practices of typical Global 2000 companies, the challenges they face, and the benefits of moving to consistent build management approaches and practices. In addition, we present user case studies that exemplify strategies that incorporate effective build management by leveraging automated technology combined with best practices and organizational strategies.

**Note: This White Paper was originally sponsored by BuildForge, Inc. in January, 2006. BuildForge was acquired by IBM in May, 2006.*

MARKET TRENDS AND EVOLUTION: SETTING THE CONTEXT

Build management should be viewed as a distinct and key element of ALM within the overall application and IT life-cycle management (ITLM) framework. Therefore, it is important to consider the ITLM landscape and context for build management before moving into this area specifically.

ITLM encompasses a range of phases that are increasingly integrated with development environments. These phases include requirements, modeling, testing, software change and configuration management, version control, and increasingly, IT project portfolio management. These integrated suites seek to provide IT and corporate executives with access to quantitative data that has typically been locked up within disparate ALM systems. Access to this data can enable qualitative assessments based on quantitative metrics, such as change management progress and testing success (or lack thereof). The addition of portfolio management capabilities to ITLM makes it possible to prioritize resources, establish effective evaluations of internal and external sourcing, and make more adaptive business choices with regard to software development projects and programs with dashboards. This drive for effective and coordinated application life-cycle management in the context of the business is a direct result of the critical nature of software for corporate success. Without the ability to respond quickly to business change and competitive pressures with appropriate decision making, applications become brittle and unresponsive.

Challenges to the successful adoption of integrated ALM suites include usage of products from multiple vendors (including proprietary tools) that are working effectively and provide similar functionality, cultural and political barriers involving poor process coordination across ALM phases, and the lack of effective ALM tools integration from a single vendor/provider. In areas where functionality is missing, we see users augmenting existing ALM tools with internal capabilities or third-party solutions. A key functionality gap we have identified within ALM is build management, including process management, communication, automation, control, and traceability of builds and releases throughout the development life cycle.

DEFINING COMPREHENSIVE BUILD MANAGEMENT

Build management brings together versions of software continually during development and also for production builds — the end process prior to preproduction testing — to then deploy the software to the enterprise. Given the corporate value of software and the core impact to the business of software failure, build management inefficiencies and failures are visceral and extremely costly because they, in turn, can lead to software delay and failure. Because this process delivers the executable — the artifact that is delivered to the customer — it contains all of the essential data regarding what was in the release, what defects were resolved, how the release can be recreated, and what systems and processes were used. All of this information is helpful for compliance or to address issues that occur following the release. If this process is unmanaged, untracked, and uncontrolled, it can be the Achilles' heel of a company's development strategy.

Full build management is more than a mere "code compile." Other elements of a more comprehensive approach to build management include build process management (establishing effective and consistent methods for build management), compliance/audit (to ensure management and traceability for regulatory compliance initiatives such as Sarbanes-Oxley), execution of complex build and release tasks as well as centralized control and management of the multiplatform configurations on which the builds are run, and reproducibility of builds and system configurations. Such coordination becomes business critical for distributed development across groups that are typically run as separate fiefdoms (e.g., developers, testers, and change and configuration management and release teams).

It is important to note that builds don't occur just close to release/production time — they happen on a continual basis during the development life cycle. Integration builds (where work from each of the developers is combined, packaged, and tested as a cohesive product) typically occur on a nightly basis, if not more frequently. Processes that are manual and error prone can slow down the entire development cycle. They also can have a significant impact on product quality, which is why eXtreme Programming and Agile process management proponents suggest building as often as possible so that errors can be detected and resolved quickly. If the build process is inefficient and lengthy, quick detection and resolution simply aren't possible, and ineffective build management can be a huge productivity drain on the development team. If developers are waiting for an error to be detected or resolved, they can't move ahead with their work. If quality assurance workers are waiting for developers to fix a build error, they may not have a version to test. This bottleneck can affect the entire team and is particularly impactful from timing and business perspectives for production builds.

Although there is increasing IT visibility into other life-cycle phases, build management has escaped scrutiny and is typically addressed by homegrown systems. Automated software change and configuration management tools and vendors have focused on other areas of functionality but have not typically incorporated either effective build management functionality or processes to support better approaches to build management. Those vendors have looked to third-party integrations to provide such capabilities. Companies have only recently begun to realize the impact that build mismanagement can have on their release time frames and team effectiveness.

CURRENT GLOBAL 2000 BUILD APPROACHES AND CHALLENGES

For the majority of Global 2000 companies, approaches to build management usually consist of cobbled-together scripts that are understood by merely one or two individuals or "build managers" within an organization. Despite lack of automation, some companies have better processes, but more often the processes are also ad hoc and rarely documented. However, because the knowledge of these specialized processes is dependent on a few individuals for integrated build processes, time frames for testing and deployment are often gated by the availability of an individual or team, which leaves the development and quality assurance teams guessing about the status of the release. This situation leads to tremendous

inefficiencies and lost team productivity, particularly in larger organizations. Basic automated build management tools (such as the open source Ant and Make alternatives) facilitate rudimentary build automation and are not comprehensive. Few processes this important to software deployment and production success have received so little focused industry attention.

In smaller, less sophisticated and less complex environments, companies could almost get away with inefficient approaches to build management. However, in the world of distributed development, where software development projects encompass local resources in conjunction with outsourcers or offshore providers, lack of management and visibility into build management is unsustainable because these teams can't interoperate if they have no standardized way to share or hand off work at the end of their workday. Similarly, the audit requirements demanded by regulatory compliance such as Sarbanes-Oxley necessitate build management visibility because end-to-end traceability is required and no single system (e.g., source control, test, defect tracking) can provide a completely reliable record. More dynamic, iterative approaches to software development such as agile and extreme programming that could enable faster deployments are blocked by lengthy, manual build processes that limit the number of code-build-test cycles that can be performed. With current ad hoc approaches, IT, development, and project management teams have little or no visible access to know how far along the projects are on the road toward deployment. This situation results in organizational confusion and divisive finger-pointing when a release is late.

The complexities of multiplatform deployments for pervasive computing and multilingual deployments for global software implementations further compound the demand for more effective and more comprehensive build management approaches because the same processes must be repeated over and over again serially rather than run concurrently. This complexity demands more sophistication than existing homegrown systems provide. Additionally, the processing requirements of complex or graphically intensive applications particularly require cogent management of systems resources because if they are not optimized, build times can become extremely lengthy, sometimes spanning multiple days. However, all organizations benefit from efficient pooling and management of systems when creating production builds. In fact, huge resource and time savings can result from effective systems utilization in conjunction with the build process. Because teams have typically allocated one server per project, homegrown build management systems have no means of distributing work across multiple servers. Thus, teams are forced to overpurchase hardware to meet their peak workloads, which can encompass 20–30% of the time, but those systems then sit idly the other 70–80% of the time. Therefore, companies are hitting technical and process barriers that prevent them from scaling development operations on several fronts, including inconsistent approaches to build management that cannot support distributed teams, the inability of configuration management teams and hardware resources to support large numbers of projects and configurations, and a lack of build management best practices that enable process reuse to enable economies of scale across multiple projects.

What are the combined results of these issues for companies? Poor quality, low staff productivity, long release cycles, noncompliance — ultimately, less profitability for the business is the result of ineffective build management.

ESTABLISHING EFFECTIVE BUILD MANAGEMENT PROCESSES, ORGANIZATIONAL STRATEGIES, AND CAPABILITIES

Success with build management — as with other life-cycle phases — requires a focus on consistent processes and organizational strategies as well as appropriate functional capabilities to automate core build management functions. These functions include rigorously managing build processes across multiple projects, auditing release contents, coordinating parallel tasks and systems, applying consistent configurations, reusing and replicating build processes, and integrating with other ALM systems such as test, defect tracking, and SCM systems.

The most successful automation tools will allow companies to incorporate existing processes into their systems rather than require a full rewrite of their build capabilities. In general, human beings are configured more for consistency than they are for change, and radical change is typically met with great resistance. Therefore, it is quite helpful if the tools enable teams to leverage and incrementally improve upon current practices when introducing practice change. The flexibility to incorporate existing build management processes and evolve them gradually to enable fuller adoption appears to be an important success criterion for a successful shift to more consistent, effective build management practices. Process automation and repeatability is key so that builds can be repeated or reproduced from scratch with a high degree of accuracy for any build or customer release. When build management users are able to add role-based security, they can take these repeatable processes and more easily delegate them to others such as developers and quality assurance teams to enable better scalability and team efficiency.

Also, appropriate organizational support and an effective corporate framework are key to making the transition. When build-related delays and failures occur close to production time, executive visibility into build challenges is heightened and even galvanized. From a business perspective, the hobbling of business flexibility and postponement of time to market caused by build breakage and inefficiencies spur change. Management and executive buy-in is vital to establish consistent build management practices domestically and globally and to provide the necessary resources and corporate commitment for change.

Some resistance may be expected from lower-level staff members because they have often written the homegrown system that would be replaced. However, if management focuses on the overall goals of product quality, better team communication, better project visibility, and getting the build team out of "firefighting" mode, this resistance can be mitigated.

Understanding the role of build management within the overall application life cycle is important as well. Effective build management can be an important linchpin to automate the handoff between software change and configuration management systems and testing processes, and it can also provide documentation across these systems that can be used for audit and compliance purposes.

Assessing core capabilities for build management and evaluating current automated options for build management make up the next phase for success in approaching this shift.

MAKING THE SHIFT

As mentioned above, the move to effective build management is much easier if teams can incorporate their existing people, tools, and processes and improve upon them over time. Thus, better build management should be an evolution rather than a revolution. To effectively make this transition, teams must prioritize the creation of better build management practices and provide organizational support to these teams. Automated tools can also assist with implementing consistent processes, and companies can choose from a number of commercially available products.

The following section offers case studies of a large telco and a leading software gaming company that implemented Rational Build Forge to address their build management challenges. Global and cross-platform development and deployment were common problems for these two companies prior to implementing Rational Build Forge's build and release management solutions.

CASE STUDIES: EVOLVING TO CONSISTENT BUILD MANAGEMENT

Large Telecommunications Organization Implements Rational Build Forge for Flexibility and Consistent Control

For a major telco organization focusing on automating end-to-end processes and dealing with multiplatform development and deployment, build management processes were inconsistent and varied widely with little commonality across teams. The confusion around builds and redundant work led to inefficient use of the staff. Because the telco had so many divergent processes, it knew that a big bang approach would not work — it needed to migrate gradually. Rational Build Forge provides a high-level process "wrapper" that could automate the telco's existing processes, so it enabled each team to implement at its own pace, which contributed to the successful adoption. Because several software products needed to be deployed across two or more platforms, the telco needed a system that would enable it to conduct concurrent builds across multiple platforms.

The company did not want to change its existing software change, configuration management, and test automation products, including IBM Rational's ClearCase for source control, IBM Rational's ClearQuest for defect tracking, Telelogic's DOORS for requirement management, and Mercury Interactive's Test Director for test management. The company had a number of capabilities for life-cycle management, but build management remained the missing link. Without a consistent, repeatable build process, releases were still being delayed.

The teams used a range of automated build tools — such as ClearMake, imake, and Nmake — but they were not implemented in a consistent fashion. Therefore, build managers used batch files and Perl scripts to manage the high-level build process, and even though they had scripts in place, the ability to view and control build changes was difficult and ad hoc. Only a few "build gurus" had enough domain knowledge to make changes using command-line operations with no intuitive graphical user interface (GUI).

The build process was something of a "black box" — unless an experienced staff member could hack through the scripts, there was little visibility into the overall build process flow, the build status, or the impact of build changes.

Because build management occurs close to the final deadline for software deployments — e.g., release management — build failures became obvious to management as a source of project slips. The time it took to detect, troubleshoot, resolve, and report on these failures as critical production deadlines approached was unacceptable for the business. Build management began to be seen as one of the teams' biggest development problems, which united commitment to organizational change and adoption of automated build technology. Although the telco considered other build management tools, one of the key differentiators of Rational Build Forge was the flexibility to utilize the telco's own paradigm and leverage the positive aspects of its existing build approaches. Rational Build Forge provided a framework to deal with notifications and parallel builds, but staff could still incorporate existing scripts into Rational Build Forge and adopt best practices gradually. The rich feature set available with Rational Build Forge for additional areas (including server pooling, process reuse, detailed reporting, process audit trails, and tool integration) was also a differentiator. In addition, Rational Build Forge was flexible in its business and sales practices — enabling the telco to prototype Rational Build Forge prior to purchase to validate the product's value in its environment. Given the level of frustration and challenge the telco had experienced with its existing build management system, it knew it would not be able to transition quickly and needed proof that the solution would meet all its needs. Development teams had long struggled with making source changes for fear of breaking the build, and users had felt burned by promises that the world "will be better." For that reason, the telco went through a fairly elaborate evaluation process, including 40 criteria and a full qualitative and quantitative analysis, prior to making the purchase decision. By implementing Rational Build Forge prior to purchase, the telco was also able to determine the expected ROI after project rollout. Once the directors saw the initial results of the pilot, adoption occurred quickly and Rational Build Forge became the standard for build management across the organization.

After adoption, the results in cost savings and increased release stability were extremely positive. A product with completely manual build tasks that used to take one person as many as two days to execute can now be completed within an hour. With Rational Build Forge, team members have better visibility into the release builds and can get a real-time view of build progress. Rather than require a single "build guru," they can have project managers and/or team leaders start a standardized build process, get immediate status information, and request builds for future times and turnarounds. Rational Build Forge's single point of entry, with its centralized management console, enables a common interface for all team members to gather and request build information, regardless of the platform on which their project runs. It also provides the ability to coordinate a complex build and release workflow in a visible fashion so users can understand the overall process, modify specific tasks (if they have the proper approval), and then execute the build in an optimized manner.

The impact of the Rational Build Forge implementation was most apparent when delivering the business-critical, highly complex customer relationship management (CRM) product. The offering allows call centers to integrate global fax, text chat, and

email communications. Because the CRM product is one of the company's flagship products, new release dates receive significant corporate attention. The application, which is more than eight years old, had become extremely unruly and difficult to manage, encompassing hundreds of thousands of files and millions of lines of code, as well as requiring the support of multiple versions across diverse platforms. Rational Build Forge is still in the process of being fully deployed and is about 40% installed to date. However, to the extent that the product is in use, staff members report that they are able to manage this business-critical application more effectively and are using server pools efficiently to conduct parallel builds to speed up the release cycle. The company is beginning to coordinate builds with existing IBM Rational source control and defect tracking and to include testing for the Java work. Its longer-term goal is to enable complete integration between unit test, defect tracking, test management, source control, and builds to automate and track the release throughout the entire application life cycle.

For this telco organization, Rational Build Forge benefits include predictability and reproducibility that are critical for operations, better visibility into status, and more efficient use of available personnel and hardware resources. Rational Build Forge also enabled the company to turn out test builds faster and to better communicate with developers bidirectionally with meaningful, timely information. Rather than have developers throw code over the wall and wait for results based on the build team's availability, developers are now notified directly about problems and are immediately directed to a status page to find out the specifics of build problems and assign specific resources to make the appropriate fixes. These capabilities have resulted in productivity improvements across the development team — both for developers and for configuration management professionals.

Leading Interactive Game Developer Adopts Rational Build Forge for Scalability and Standardized Process

An interactive gaming software company has specific build management challenges that typify the rigorous demands of the gaming industry, including high iteration speed for engineering and the need for sufficient processing power to support builds for graphic-intensive software. Game development epitomizes the "extreme development environment" with highly complex applications, large distributed development teams, multiple delivery platforms, and unforgiving release schedules that are timed precisely to coincide with events such as movie premieres and holiday shopping seasons. As such, these conditions provide a strong stress test for effective build management.

This gaming company has adopted many Agile Development practices that require rapid build iterations. For example, its configuration management team has committed to delivering a complete software package to each product team in under an hour, an event that can occur multiple times each day. Using tools that modify animation, developers need to make changes to the program, make corresponding changes to the animation, and deliver a final image artifact (measured in gigabytes) to the pipeline within 30 minutes. Both the sheer number of data records and the speed required for entertainment engineering to meet deadlines are daunting with regard to build management.

Because gaming developers have traditionally been immature in their software configuration management discipline, developers had kludged together batch files for each project that didn't scale. Initially, the company's build management staff created a master build utility that was threaded to allow compiles, rendering, and packaging to occur relatively unattended. The company used these homegrown tools initially for build management and distribution, but it was hard coded to use a specific machine that would eventually die and cause operations to stall. The company determined that its internal system would require significant rework to handle distributed machine management, which would create an additional development and support burden for the team. It also wanted tighter communication between its build system and its Perforce source repository.

Rather than rework the homegrown applications, the company decided to evaluate commercial build management solutions and adopted Rational Build Forge for its scalability, server management, and process control capabilities. The company has automated the deployment of pipelines and empowered its product teams to execute approved production processes on demand. Before bringing in Rational Build Forge, the company could concurrently manage only two or three projects. By automating build processes, the team can support 10–20 projects simultaneously with the existing staff. Now the company allocates two build engineers per project where it used to require four or five people. Rational Build Forge provided a nonthreatening context whereby engineers could adopt industry-standard configuration practices without requiring substantial change to the way in which they worked.

Prior to implementing Rational Build Forge, the company ran out of processing power with regard to build distribution and machine management. It reached capacity quickly, but with Rational Build Forge, the company can leverage pools of more than 40 machines. Machine pools are available through a central portal and enable developers to execute a distributed build project with the click of a button — even on servers that may be located in another country. After successful compilation, Rational Build Forge invokes automated testing protocols so that the team can know within 10 minutes if the build is functional.

Rational Build Forge not only is a process execution engine, but it also has agent technology that runs on the production machines to thread and parallelize the build workload. Therefore, if there are four systems and one goes down, the work will be automatically reallocated to the remaining three. The system can deploy a file to one of the machines scheduled for testing and have the agent run the executable to determine if the application crashed on start-up.

The company also uses Rational Build Forge to manage build configurations to avoid inconsistencies and errors. Environment variables automate the use of the appropriate compiler (Microsoft C++ or C# compilers), version strings, and other parameters, depending on the project. Previously, this information resided on each build server and was rarely documented. Moreover, incorrect configurations were often a source of build errors. With Rational Build Forge, this information is retained in a central knowledgebase and used consistently for each build iteration.

Some game companies debate whether nightly builds are necessary — not so at this company. Because this gaming company now employs 100–150 developers per project, where each game can contain a million records, even single nightly builds are insufficient. The company can't have all team members checking in changes to the source tree without some barrier to entry. Rational Build Forge, in combination with Perforce for configuration management, has facilitated improved management and builds and minimized the need for "heroics" to repair broken builds.

Adoption of Rational Build Forge's automated build management capabilities enabled this gaming and entertainment company to significantly cut build management time, increase efficiency (by pooling multiple resources to do build management and run projects concurrently), and provide metrics and much faster release times.

SUMMARY: BUILD MANAGEMENT FOR EFFICIENCY, COST SAVINGS, AND BUSINESS ADAPTABILITY

Establishing effective build management enables economies of scale for resource management and the ability to structure successful software deployments. Global 2000 organizations should evaluate appropriate process, organizational, and automated tools for build management to enable adaptive, stable software implementations to drive business success. Build and release management consistency is no longer optional for businesses and for effective development in the complex, distributed world of contemporary software creation. In that context, build repeatability, traceability, team efficiency, and life-cycle automation capabilities are important for ISVs and IT organizations to support business adaptability.