



Implementing build and release best practices with Rational Build Forge software.

*Rob Cuddy, worldwide enablement engineer, Rational software,
IBM Software Group*

*Hadley W. Paul, business analyst, Rational software,
IBM Software Group*

Contents

2	<i>Introduction</i>
3	<i>Practice one: develop a build management strategy</i>
5	<i>Practice two: establish complete reproducibility</i>
7	<i>Practice three: automate processes and integrate essential systems</i>
8	<i>Practice four: institute centralized access and collaboration for all stakeholders</i>
9	<i>Practice five: link build processes to deployment environments</i>
12	<i>Practice six: use metrics to evaluate progress and performance</i>
14	<i>Practice seven: apply build acceleration techniques</i>
16	<i>Practice eight: abstract processes and metadata from physical resources</i>
17	<i>Practice nine: optimize processes and supporting architecture design</i>
18	<i>Practice 10: build early and often</i>
20	<i>Why Rational Build Forge and IBM?</i>

Introduction

What if your software development organization could vastly accelerate the build and release process? Enable you to manage your development system from virtually anywhere? Free your developers from the most repetitive, mundane tasks? What if you could do all this while leveraging your existing investments? That's what a good process automation framework, such as IBM Rational® Build Forge® software, can help your business do for your development organization. With process automation, you can centralize, automate and accelerate software development using the tools you already have. You can connect multiple applications and hardware platforms to enable global reporting, centralized tracking and distributed access to hardware resources.

Process automation is most effective when a series of best practices is applied to it. Why invest time in best practices? Because they enable you to efficiently develop and deliver higher-quality software. Best practices also help your organization improve your development team productivity, product quality and regulatory compliance. The 10 best practices included in this paper are the result of years of working with software development organizations in a variety of build processes, environments and applications.

Some of these concepts are simple, and some are more complex. Some involve a one-time change in the project setup, architecture or design, while others need to be implemented in iterations, over time. You will need to implement some practices before others can be addressed. But following these practices can help you create a streamlined, centralized system for consistently developing quality software.

Rational Build Forge is one such automation framework that can help streamline and organize the software delivery process. The best practices discussed in this paper have been developed in partnership with Rational Build Forge customers over the course of many years, and have proved effective in a wide variety of projects, environments and applications. Useful whether your development team has 10 people or 10,000 people, these practices can improve efficiency, build reliability and enhance product quality—even if your group is in the same office or located in multiple offices around the world. Keep in mind that these practices do not have to be adopted in the order presented. In fact, you should consider first implementing those practices that will make the biggest impact on your unique environment.

Practice one: develop a build management strategy

Before you build, you must plan. Build management is more than just code compile. True build management also includes establishing effective and consistent methods for builds and their execution. It includes not only the execution of complex build and release tasks but also the centralized control and management of the multiplatform configurations on which the builds are performed. Build management also ensures that, once established, builds and system configurations themselves can be reproduced. An effective build management strategy is critical to developing a solid build and release process.

A well-defined build process helps businesses meet today's common software development challenges, including:

- *Globalization and distributed development.*
- *Quality and time to market.*
- *Regulatory compliance management and governance.*

Whether you are implementing a new build process or refining an existing one, it's important to keep a few things in mind.

Consider your whole build and release process

A common mistake that teams make when defining and documenting a build management strategy is to focus only on the technical aspects of the build process. There are also nontechnical factors that impact your level of build and release effectiveness. For example, how the team is structured and how it performs different development tasks are crucial considerations, as they may show where process gaps or inefficiencies exist, and these vulnerabilities can affect the build process. You may also want to understand how process metrics and data are gathered so the team can consistently and accurately measure its progress over time. What's important is to design your build and release process with the big-picture goal in mind. This goal could be maximizing team productivity, accelerating product delivery cycles or reducing downtime for business-critical applications. This mindset will help you attain the greatest return on investment for your team.

By keeping both the technical and the business goals in mind, you will be more successful in creating a living build and release management strategy that helps you meet the business needs of the company.

Continually assess and refine your process

Developing a build management strategy isn't a finite task. It is an organic process. As you would with any other software development artifact, you need to continually assess your process and make changes as necessary. Start with the build process as it exists today, and then expand the strategy to include every facet of the software delivery lifecycle. Include input from other team members, such as deployment and testing personnel. Your business changes over time, and so will your build management strategy.

Include three key elements

While a build management strategy can be implemented in a variety of ways, there are three essential elements needed for a successful build process.

- **Repeatability and reliability.** *Do the same thing over and over again, with the same accuracy and results each time. Because a developer may need to rebuild later, a build process should snapshot everything at the moment it is created, including source file versions, compiler settings and the operating system environment itself. This information is critical if a developer needs to reproduce an environment to fix defects after a product has been released.*
- **Agility and speed.** *Be able to integrate changes quickly, when needed. You can meet the needs of users more effectively with a build process that is easy to set up and execute. The process should also be capable of being executed continually—maybe many times a day. This makes it possible to deliver hot fixes quickly, and it also facilitates projects practicing continuous integration, where developers are working on small incremental changes and committing them frequently.*
- **Traceability and completeness.** *Know why you're doing what you're doing throughout the complete development lifecycle. A build process should automatically capture and report on new features, defects and other changes that have gone into a build. This information is critical to quality assurance (QA) teams because they need to know which of the defects they raised have been included in the build. Implementing a solid strategy also helps provide accountability that's necessary in order to address regulatory compliance initiatives such as Sarbanes-Oxley.*

Practice two: establish complete reproducibility

After establishing a build management strategy, you need to establish that your whole build process can be reproduced. This is an important step because several situations in the development process require that you be able to go back in time to accurately generate an artifact in the same manner in which it was originally created. When you have complete reproducibility, you can use the exact same steps, exact same source code, tools, environments and servers that you used to produce a deliverable – whether you produced it an hour ago or a year ago. By reproducing an artifact, you can help:

- *Identify defects or successful processes for future builds.*
- *Satisfy audit requests.*
- *Provide a baseline for reuse and other development work.*

How do you help ensure that you can reproduce an artifact exactly? First, ensure that project and application builds are performed in a standard structure. This structure should include such aspects of the build process as the locations of source components, test components and generated build outputs, as well as the naming conventions for build scripts. Whenever possible, builds should be performed in a clean and controlled environment. And if you're releasing a build, such an environment is mandatory.

While reproducibility is affected by several entities outside of Rational Build Forge (such as a repository from which to access old source code), once you have a consistent structure and process in place, two actions are essential for reproducibility: establish and maintain a high level of security, and identify and account for all dependencies.

Establish and maintain a high level of security

It is impossible to reproduce exactly something that continually changes or something that is not tightly controlled. Limit access and permission to implement build, process and artifact changes. To enable high security, one recommended option is to implement a role-based security model. For example Rational Build Forge implements this model using access groups to control user permission. You gain more control over security when you can develop access groups within Rational Build Forge that map to roles within the organization.

You can then assign appropriate permissions for each group, and you can add users to the appropriate group or to multiple groups. In Rational Build Forge, security access is determined by group membership so that information can be displayed in an appropriate manner. If you use Lightweight Directory Access Protocol (LDAP), you can map existing groups in the domain to access groups in Rational Build Forge.

In Rational Build Forge, objects also have permission settings with attributes that can be set. For example, maybe you need to hide sensitive information, such as a password setting. You may only want users who oversee password maintenance to have access to this information. Figure 1 shows how you can set an environment group in Rational Build Forge that includes a variable that will not be visible to general users. Attributes of the restricted variable are set to *Assign/Hidden* and *Suppress Display*. With *Assign/Hidden*, the value of the variable is displayed as a series of asterisks. With *Suppress Display*, the variable does not appear in the list of project environment variables when the project is run manually.

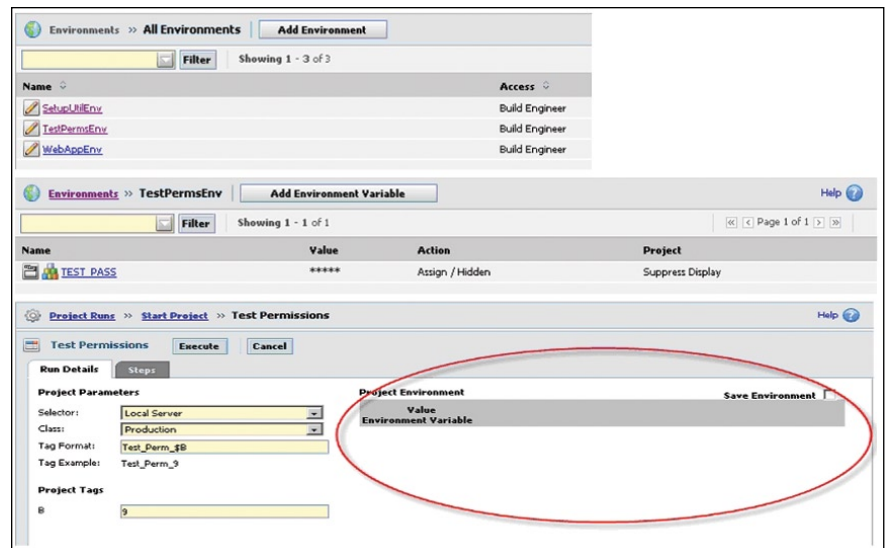


Figure 1: Sample screenshot showing environment variable behavior

Identify and account for all dependencies

A second key action in establishing complete reproducibility is to identify and account for all dependencies used to build a release. To account for dependencies, you need to capture all the information that is pertinent to the build process, including environment variables, system settings and build machine configuration. Identifying and storing this information enables you to successfully rebuild an artifact in a repeatable fashion. Rational Build Forge captures this information during a build process run, and with environment groups, you can define variables to capture their values. In addition, you can write custom information to a bill of materials to obtain a complete representation of the run-time environment.

Practice three: automate processes and integrate essential systems

Once a stable and secure environment and a basic build management strategy are in place, the next objective is to automate individual processes and integrate essential systems. The goal should be to remove as much manual intervention as possible and, ideally, automate each process entirely. Fully automated processes depend on integration with key systems as a prerequisite. There are a few important elements in this system integration.

Map out process workflows

Because Rational Build Forge allows for a great deal of flexibility in managing process workflows, it's possible, and recommended, to map out different process workflows before connecting systems. Process workflow mapping can identify any dependencies in steps. Rational Build Forge can accommodate different workflow paths by using exit codes, filters and pass/fail chains for process decisions. Don't simply focus on the build portion of the workflow. Consider items that influence the build as well as items influenced by the build, including:

- *Source code repositories.*
- *Deployment locations.*
- *Unit and/or smoke tests.*

Identify relationships between processes

Another important element in connecting essential systems is to understand the relationship between the different steps in the process workflow. Rational Build Forge allows you to make process decisions at the end of each step. The process workflows should be constructed so that these decisions can happen in an automated fashion. With Rational Build Forge, you can consider what you would want to happen for each possible outcome of the steps in your process to keep work moving.

For example, consider what you would want to happen should the deployment step of your process fail. You could configure Rational Build Forge to notify appropriate access groups concerning the failure. But you also might want to perform cleanup actions or launch other processes in addition to just providing notification. Rational Build Forge can help you set decisions for multiple actions and multiple scenarios.

Once you have mapped out your processes and understand the relationships, you can use Rational Build Forge to:

- *Automate the setting of a specific environment using environment groups.*
- *Build large systems through builds of the subsystems.*
 - *Create a “super” or “master” project where steps are really calls to smaller build processes. These steps are connected to smaller processes using inline and chain configurations in Rational Build Forge.*
- *Implement different workflows with different projects.*
- *Include the complete set of processes required—not just the master build process:*
 - *Developer incremental builds*
 - *QA integration builds*
 - *Master builds*
 - *Patch/service pack releases*

Practice four: institute centralized access and collaboration for all stakeholders

Centralized collaboration allows team members to access and utilize all build process resources. It is essential for team-oriented projects, especially ones with remote teams. Centralized collaboration facilitates clearer understanding of the process, consistent information concerning project builds, greater communication

between team members and increased visibility of anything and everything connected to the process. Rational Build Forge enables you to define users and user groups within the centralized framework, which can help enforce policies and ensure that users see information that is pertinent to their role.

As a result, you get communication without chaos. Reporting, resources, process steps and configurations are stored and centralized. All users see the same information for the same builds, which eliminates the confusion that can occur when there are multiple builds and artifacts. There are several features of Rational Build Forge that help facilitate centralized information and communication.

The bill of materials

The bill of materials (BOM) is a central aspect of Rational Build Forge collaboration. Produced each time a process is run, it contains all of the results and configuration information concerning the run. The BOM can be customized to include source code changes, defect IDs, test results and more specialized information. By consolidating this information into a single location, users can more quickly and easily address and resolve issues.

Targeting notifications

To improve collaboration, you can also use targeting notifications within Rational Build Forge. Target notifications help ensure that the right people receive the right information at the right time. You can set notifications for a variety of parameters. For example, instead of having to e-mail the entire organization when a build fails, you can simply send a notification to the appropriate development team. Notifications can also be customized with specific information, such as a link to the build results.

Practice five: link build processes to deployment environments

Too often, software organizations fail to tie together the transition between generating deliverables and then packaging and deploying them. Instead, they handle internal deployment in an “over the wall” fashion. To realize a successful build and release framework, you must automate and streamline these transitions as much as possible. Automating the movement of files from one group to the next in a development process saves significant time. Target deployment environments

can be final destinations or intermediate drop zones for another application to pick up and do final packaging and deployment. Users and groups have files already waiting for them, instead of requiring those users and groups to migrate files manually after receiving an e-mail.

Typically the deliverable deployment process consists of the same basic steps across many projects. Rational Build Forge allows users to modularize these reusable steps into libraries and connect them to process runs. As a result, deployments can happen in a controlled and consistent manner.

You can also configure process workflows where some steps can only be run by certain users. For example, maybe only a team lead may deploy to a QA environment.

With Rational Build Forge, you can configure workflow so that the deployment step runs only when a team lead performs the process run. Figure 2 below shows how roles can be specialized in a process run.

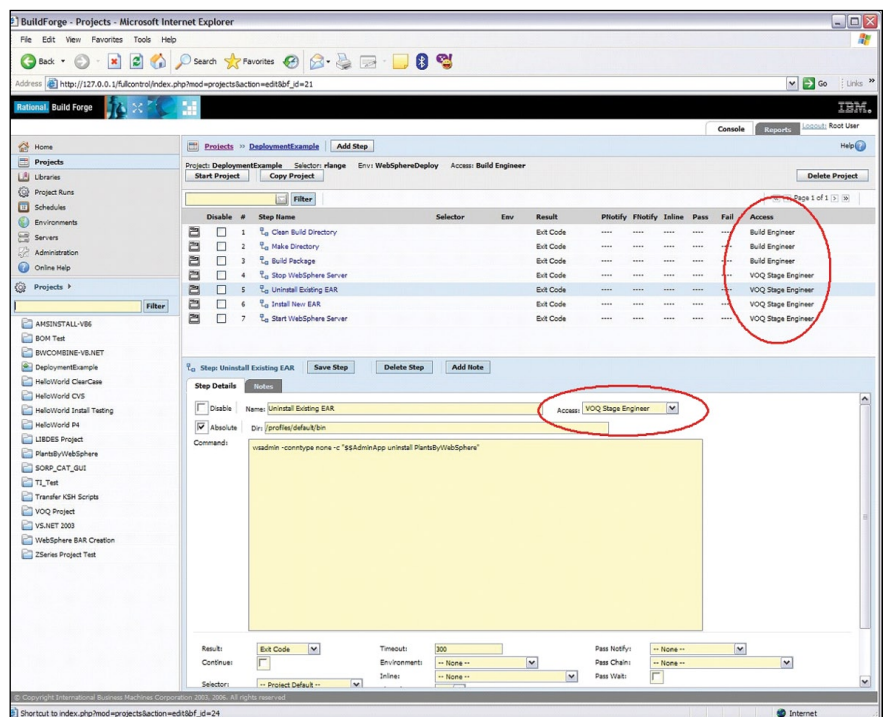


Figure 2: Sample screenshot showing steps configured for different groups

You can also perform operations on process runs that have already occurred. For example, say you are performing a build and sending it to a QA team. After some time, the build is approved and moved to the next level of testing. In Rational Build Forge, a process run is associated with an entity called a *class*. Approved users can change the class of a given process run and configure Rational Build Forge to perform appropriate steps when a class changes. This is accomplished by connecting the class to libraries containing the steps to perform. Steps can be run when changing to a class, from a class or both. Figure 3 highlights where an existing run class can be changed. Figure 4 illustrates where the class attributes are set. The **Start on Entry** attribute defines the steps to run when a process run is changed to this class. **Start on Exit** defines the steps to run when a process run is changed from a class.



Figure 3: Sample screen shot showing where the class property can be changed for a run

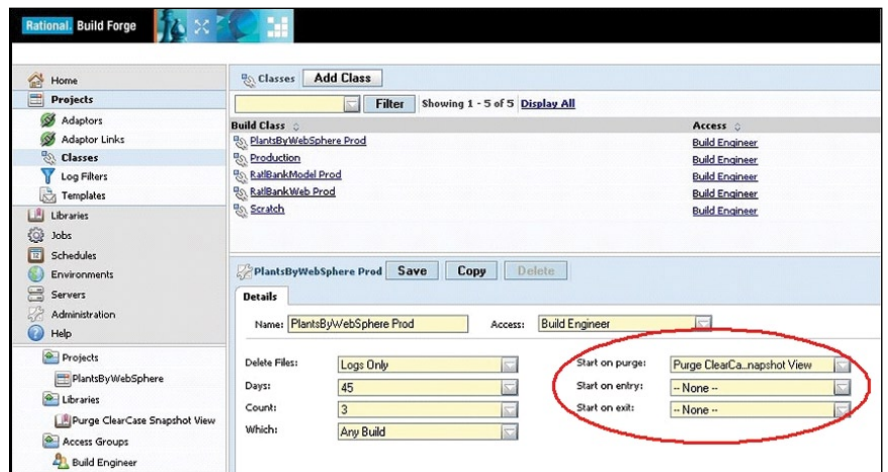


Figure 4: Sample screenshot showing class attribute locations

Practice six: use metrics to evaluate progress and performance

Your business needs accurate information in order to make informed business decisions. In terms of software development, it's impossible to improve what cannot be measured. Reporting improves visibility and provides reliable data that can help you understand and manage various project metrics—including the build process itself. Rational Build Forge includes reporting features that can provide this valuable information.

It's important to analyze these metrics in the context of your business. For example, a lengthy build process may not be just an inconvenience, it may be a roadblock that delays product delivery—a potentially costly situation. A lengthy build process may also be an indication that hardware resources aren't being used efficiently, which can add needless costs to a project. It's possible that a build error isn't just affecting the development team—it may also be affecting key stakeholders, such as internal employees, partners or end users. By linking process metrics to the business impact, you will be better able to quantify the value of making build and release process improvements for the organization.

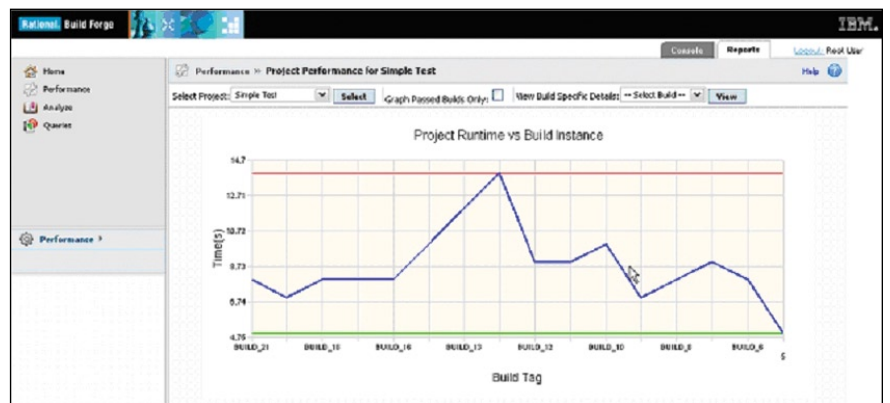


Figure 5: Sample report showing build duration time on a per-build basis

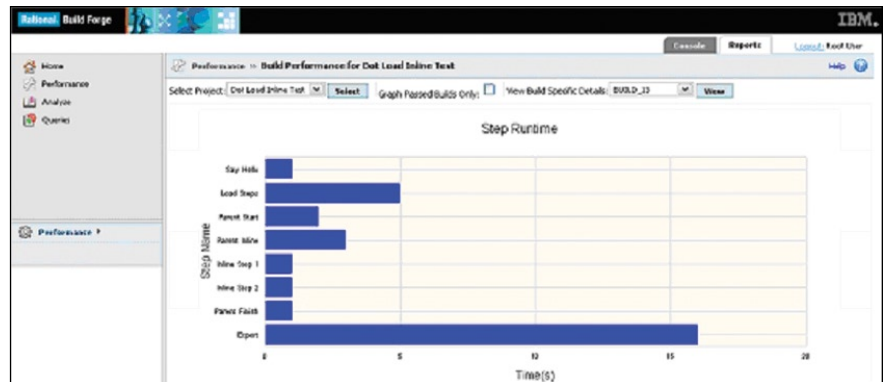
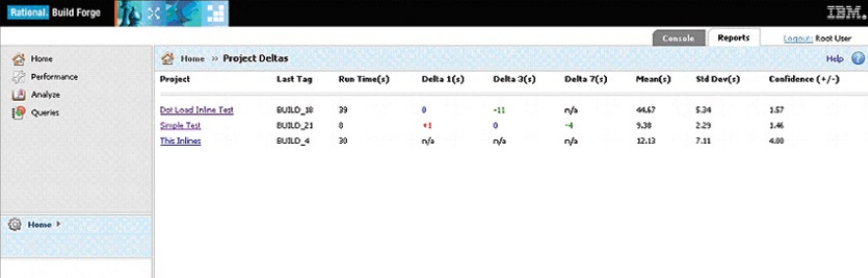


Figure 6: Sample report showing step duration time on a per-step basis

Many development teams use the build process as the primary mechanism for capturing project metrics, but often developers don't capture metrics about the process itself. For example, when executing a build, metrics might be gathered concerning unit tests passed, code coverage or source code changes. However, information pertaining to that build will be missed. Process metrics, such as how many builds have passed or failed or the average total time required for a build, yield invaluable information that can be utilized to increase efficiency. Project metrics that can be helpful include:

- *Number of passed or failed builds per project.*
- *Build confidence (the ratio of successful to failed builds).*
- *Average total build time per project.*
- *Components reused in a build.*
- *Change tasks and/or defects implemented in each build.*

Figure 7 illustrates some of the metrics captured in the reports. This enables the user to evaluate the changes in project runs over time, also providing the mean, standard deviation, and confidence values for this distribution. These metrics include the last tag, the run time and the deltas of each.



The screenshot shows the Rational Build Forge interface. The top navigation bar includes 'Rational Build Forge', 'Console', 'Reports', and 'Logout: Root User'. The left sidebar contains 'Home', 'Performance', 'Analyze', and 'Queries'. The main content area is titled 'Home >> Project Deltas' and contains a table with the following data:

Project	Last Tag	Run Time(s)	Delta 1(s)	Delta 3(s)	Delta 7(s)	Mean(s)	Std Dev(s)	Confidence (+/-)
DotLoad Inline Test	BUILD_38	39	0	-11	n/a	46.67	5.34	1.57
Simple Test	BUILD_21	0	+1	0	-4	5.38	2.29	1.46
The Inline	BUILD_4	30	n/a	n/a	n/a	12.13	7.11	4.88

Figure 7: This graphic is a screenshot of a reporting data from the Reports tab. It shows data of the Project Deltas for projects in various builds.

Other useful reporting metrics include the following:

- *What is the duration of the project over time?*
- *Are the total number of defects found/fixed increasing or decreasing?*
- *What step in my project typically runs the longest?*
- *What step in my project typically breaks most often?*
- *Who is responsible for the code that breaks the build?*
- *How many successful vs. unsuccessful builds have occurred for project X?*
- *What is the number of passed/failed builds per project?*
- *What is our build confidence?*
- *What components can be reused in a build?*
- *Which change tasks and/or defects are implemented in each build?*

Practice seven: apply build acceleration techniques

Build acceleration is the concept of reducing the total time of the build process from end to end. This benefit applies to both iterative and agile methodologies. There are two ways that you can help accelerate your overall build process using Rational Build Forge.

Execute project steps in parallel

Rational Build Forge allows you to execute project steps in parallel rather than in a series, helping to minimize run time. Running project steps in parallel is called *threading*, and it is an effective option for processes that do not depend on an immediate previous step.

Figure 8 illustrates how, during a sample process, you can compile the same source code on several different platform types. During the compilation stage, you would traditionally compile in a series – on one platform, then the next and so on – resulting in a long overall build process. But since the compilation steps do not rely on one another (for example, the Linux® compile does not depend on the completion of the Microsoft® Windows® compile), they can be threaded to run in parallel, reducing the total build time.

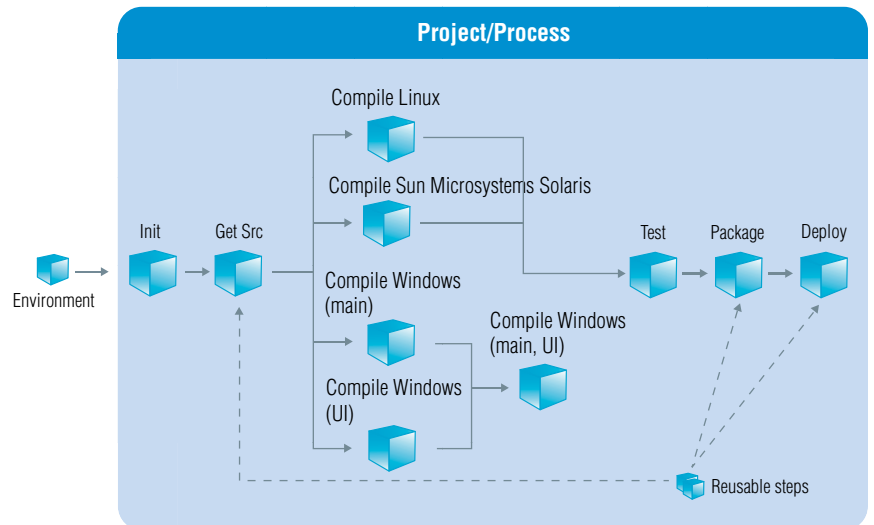


Figure 8. Sample process workflow broken into steps

You can apply threading to any manual or inefficient steps within the overall process, not just the compile step. For example, Rational Build Forge can execute simultaneous source extractions, running test scripts, packaging or multiple deployments.

Efficiently use existing hardware

You can also use Rational Build Forge to help you determine the most efficient and effective way to use your existing hardware. Rational Build Forge uses objects called selectors to locate the best machine at a given moment for a given task. You can also distribute parts of your overall process to different machines. For example, it may be faster to build different components of a product on several different machines and then have a step in the process to pull the built objects together from the various machines.

Practice eight: abstract processes and metadata from physical resources

A best practice when creating the steps to a project is to abstract the configuration information from physical resources. Over time, build scripts can quickly become large and unwieldy as specific configuration information gets associated with them. Often development teams attempt to manage these large scripts by extracting the configuration data. With a scripted build process, this typically means creating a separate data file alongside the main build scripts. However, as the process becomes more complex, it is common to construct these data files to rely heavily upon hardware resources. This method firmly ties the process to the hardware resource. But in general, hardware resources for building should be thought of as transferable. Hardware failure should not jeopardize software processes that are critical to a project.

Rational Build Forge enables you to store configuration information for different processes in environments in its database. For example, if a step needs to use a compiler, you define the location of the compiler in an environment variable and then use the variable in the step, rather than hard coding the location of the compiler into the step command. Similarly, you can abstract metadata about the hardware resources, such as operating system level or amount of available memory, using collectors and selectors to store that information in manifests for the different resources. Figure 9 shows how you can obtain and store machine-specific information into a manifest.

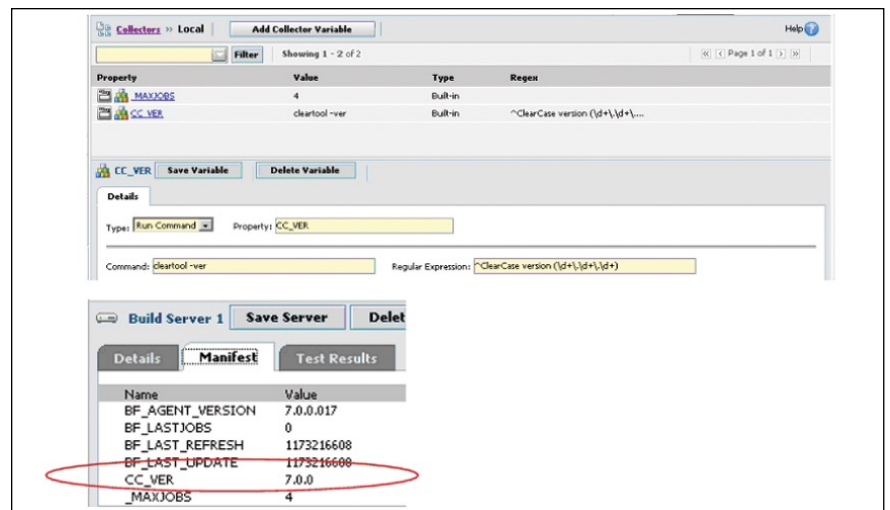


Figure 9. Sample illustration of capturing and storing resource information

When you abstract steps from physical resources, it's also a good idea to examine each process step and to identify places where steps are repeated. Just as developers implement code routines to save time and promote sharing and reuse, build processes can also benefit from reuse. Steps that are repeated similarly across multiple processes, just with different parameters, can be grouped into a reusable library. This modularization helps simplify steps when you start new projects.

Practice nine: optimize processes and supporting architecture design

Once the complete processes have been placed into Rational Build Forge, they're ready for optimization. Optimization includes both the Rational Build Forge architecture and the processes implemented in it.

Optimizing the architecture

To optimize the Rational Build Forge architecture, you must fine-tune the application's management console. You need to examine several parameters to ensure that the console runs efficiently. Two important ones are the run queue size and the maximum console processes. The run queue size limits the number of project runs that can occur at one time. The maximum console processes setting determines the maximum number of processes that the management console can run at one time. These two parameters work together. The maximum console processes setting must be at least five greater (+5) than the run queue size so that the system can support jobs that are queued.

You can control the maximum number of jobs for a given server using the `_MAXJOBS` parameter in a collector for the server. You can also configure this parameter in the management console for all servers at once. In addition to these two parameters, the auto-logoff time and the password format settings should be implemented to help enhance security.

Optimizing the process

Process optimization involves assessing the different building blocks in the process, identifying reusable parts and parts that use parameters, and implementing changes. Process steps can be difficult to break down and difficult to refactor in a single increment. Instead, start simple with Rational Build Forge projects in an as-is fashion, using existing scripts and commands. Once you get the process in, you can further optimize it over time. As with optimizing the

architecture, examine the different processes and look for potential reuse and parameters, and then place these tasks into libraries. Use environment groups to influence different parameters. These modular pieces are easier to maintain and reuse, and they scale better with growth.

Rational Build Forge is very helpful in optimization because it allows flexibility of environmental variable usage. You can define environment variables for servers, projects and steps. Figure 10 illustrates different behaviors exhibited by environment variables defined in groups.

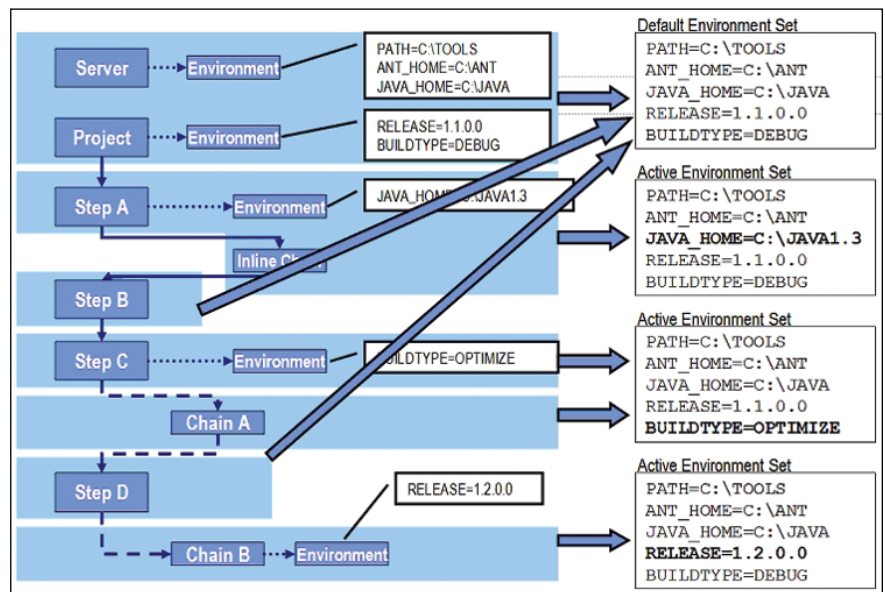


Figure 10: Environment variable behavior

Practice 10: build early and often

After optimization, you can increase the frequency of product builds. More frequent builds increase feedback and help identify problems quickly. They also help ensure that your project is on track. Feedback comes from builds, and the more feedback you have, the easier it is to gauge the health of your software application. This makes it easier to fix problems earlier in the release process, rather than waiting until the end of the project to address an issue.

Rational Build Forge provides several options for achieving this best practice. First, in the management console, you can give permission to developers and other team members to run the process on an as-needed basis. Once again, *Assign/Hidden* and *Suppress Display* allow you to set role-based access to projects and steps. Rational Build Forge also enables you to schedule automated project runs at predefined intervals, such as a nightly build. Figure 11 shows a sample schedule. Finally, Rational Build Forge allows developers to validate their changes as part of an integration build using an integrated development environment (IDE) plug-in, so they can run a preflight check before integrating changes into a real build.

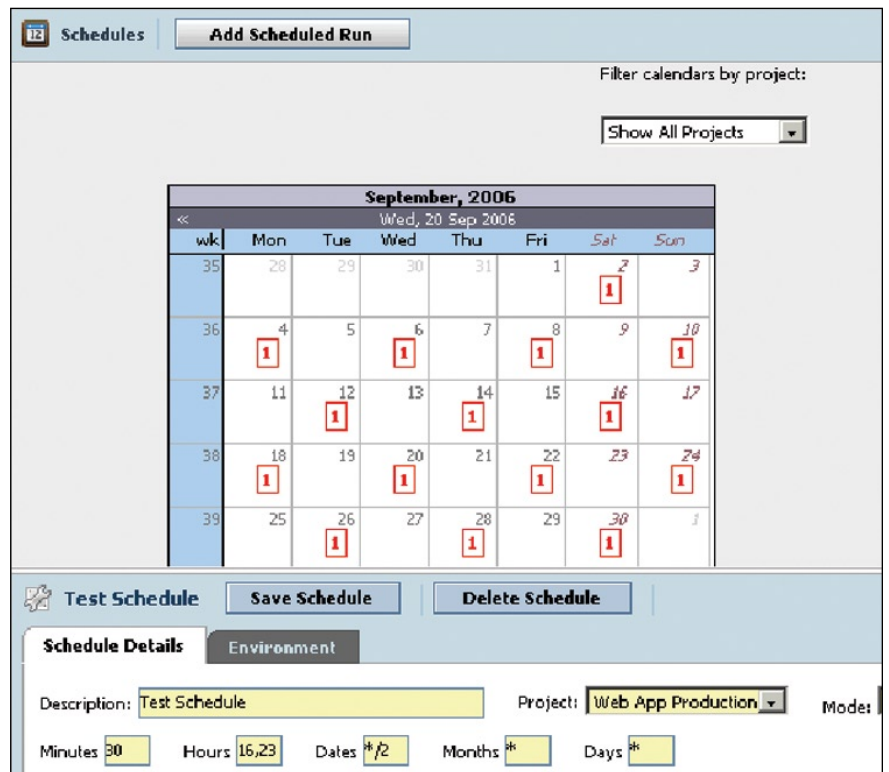


Figure 11: Sample schedule screenshot



Why Rational Build Forge and IBM?

IBM Rational Build Forge Enterprise Edition software offers comprehensive build and release management for medium to large development teams. With Rational Build Forge software, teams can easily automate and reuse repetitive build and release tasks for greater efficiency and reliability. Compatibility with existing build scripts, batch files and development tools speeds implementation, and error log filtering and automated notifications allow developers to rapidly detect and resolve errors. Rational Build Forge software provides Web-based access, so build activities can be viewed and managed from anywhere, at any time. IBM Rational Build Forge software also helps you make use of your IT investments by integrating with existing development technologies and by offering broad operating system support, including the IBM AIX®, UNIX®, IBM i5/OS®, Linux, Apple Macintosh, Sun Solaris, Microsoft Windows and IBM z/OS® platforms.

For more information

To learn more about how IBM Rational Build Forge software can help you implement build and release best practices, visit:

ibm.com/software/awdtools/buildforge/

or consult your IBM Rational Build Forge account representative.

© Copyright IBM Corporation 2008

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
02-08
All Rights Reserved.

AIX, Build Forge, i5, IBM, the IBM logo, Rational and system z are trademarks of International Business Machines Corporation in the United States, other countries or both.

Windows is a registered trademark of Microsoft Corporation in the United States, other countries or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries or both.

Other company, product and service names may be trademarks or registered trademarks or service marks of others.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.