

---

# Build Management Tools Research

---

51 pages

Date: 03.04.2007

Version: 1.0

# Table of Contents

1. History .....	3
1.1 Revision History.....	3
1.2 Review History.....	3
1.3 Approval History .....	3
2. Introduction.....	4
2.1 Purpose.....	4
2.2 Summary .....	4
2.3 Scope.....	6
2.4 Definitions, Acronyms and Abbreviations .....	6
3. Introduction to Build Management Software.....	7
4. Customer Needs .....	8
5. Estimation Criteria.....	9
6. Build Scripting Engines Comparison.....	12
6.1 Apache Ant 1.7.0.....	12
6.2 SCons 0.96.1.....	17
6.3 NAnt 0.85.....	19
6.4 MSBuild.....	22
6.5 Comparison Table.....	24
7. Build Management Tools Comparison.....	25
7.1 Maven 2.0.5.....	25
7.2 QuickBuild 1.2.3 .....	28
7.3 TeamCity 2.0 .....	31
7.4 CruiseControl 2.6.1 .....	34
7.5 AntHillPro 3.2 .....	36
7.6 IBM® Rational® Build Forge™ Enterprise Edition .....	39
7.7 Comparison Table.....	43



Appendix A. BSE Comparison Table..... 44

Appendix B. BMT Comparison Table ..... 47

# 1. History

## 1.1 Revision History

Version	Date	Description of Changes	Reason	Made by
0.1		Created	Customer request for research	D. Boriskin
1.0	03.04.2007	Proposed		

## 1.2 Review History

Version	Date	Reviewer	Reference
0.2	30.04.2007	A. Ignatov	
0.3	02.04.2007	E. Gomonova	

## 1.3 Approval History

Version	Date	Approved by	Signature or reference
04	02.04.2007	E. Povalyaev	

---

## 2. Introduction

### 2.1 Purpose

The main purpose of this document is to present the results of the research established by Luxoft to compare several Build Management Tools and find out the most appropriate solution, which is eminently suitable to business and technical requirements of Customer.

---

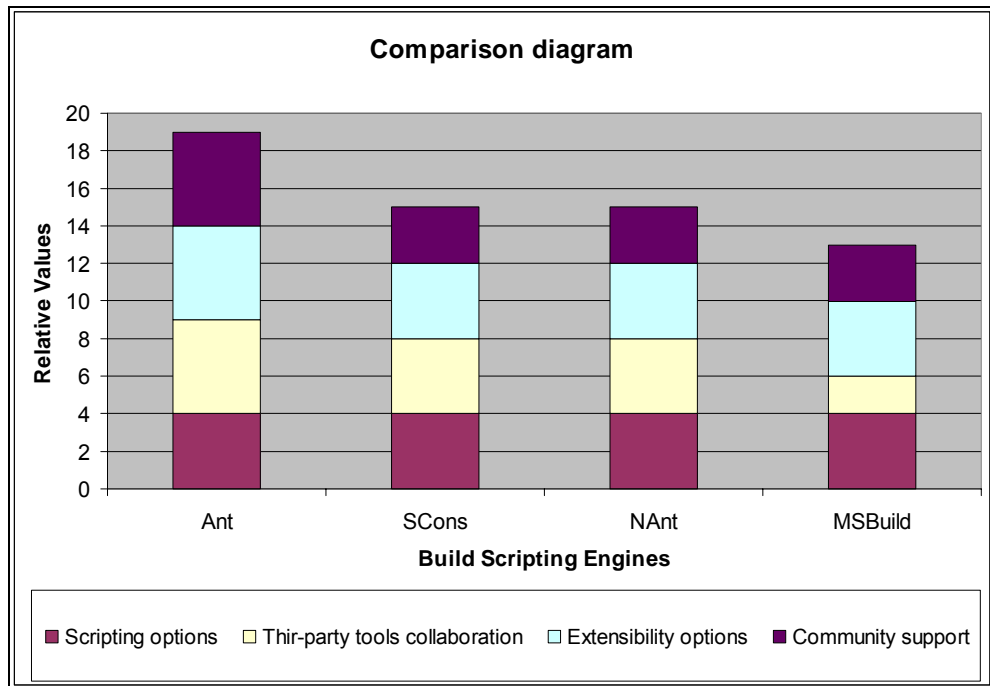
### 2.2 Summary

During evaluation process, we considered two classes of build management software: build scripting engines and build management tools.

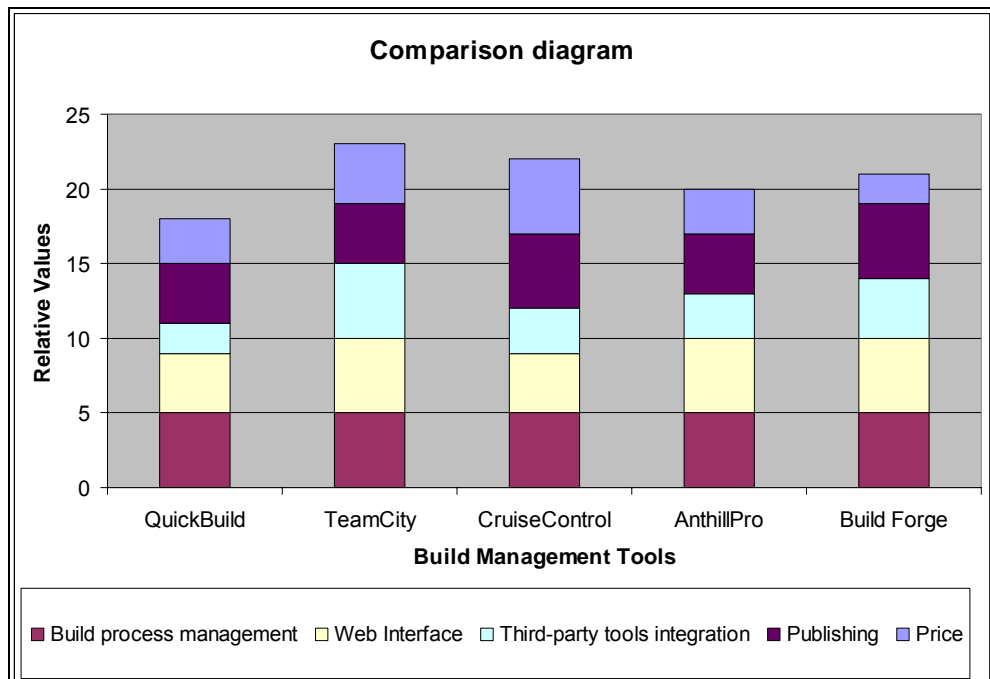
Build scripting engines (BSE) can be used stand alone for organizing build process and managing the complexity of building software programs from many source files and library components. BSE products read script files, which are description files that specify the parts and processes that should be used to build a software product.

Build management tools (BMT) improve productivity for software builds by providing more functionality than BSE products provide. Build manager tools usually include a desktop or web interface, a more convenient build process scripting language, and means for executing all or part of scripted build processes, and for logging and reporting build results. Some of the build manager tools main characteristics are GUI interface, increased interaction with builds, build scripting language or interface, and built-in capabilities for build execution, logging, and reporting.

Among BSE tools, **Ant is a leader** (see the diagram below for graphical representation of Ant leadership). Ant is an industry standard BSE tool for building Java programs. It supports a wide variety of custom QA tools, deployment options and provides a flexible framework for extensibility. Released from a year of 2000, the current version of Ant (1.7) automates the most of the routing build process tasks, provides integration almost with all build management tools and has plug-ins to many of Java IDEs. There are a lot of custom plug-ins and add-on utilities that help to generate Ant XML-based scripts, provide support for dependencies management, and automate some other routine tasks.



Among BMT, **JetBrains TeamCity is a leader** (see the diagram below for graphical representation of JetBrains TeamCity leadership). TeamCity is an innovative, IDE independent, integrated team environment targeted at .NET and Java software developers and their managers. It automates and coordinates key collaborative processes to eliminate manual systems and delays, providing tight integration with multiple build and test tools, real-time unit test frequency, server-side inspections and code coverage analysis. TeamCity has commercial support for a reasonable price.





### Interesting Finds

Build Forge deserves a separate mention among build management tools. A year ago, Build Forge was one of the build tools market start-ups that was acquired by IBM and included in its Rational® product line. In the light of the rapidly evolved Jazz project – IBM's team collaboration portal – Build Forge may become a valuable add-on. Build Forge feature set include concurrent execution for independent build steps, progress tracking, email notifications, logging, and a rich web interface with advanced reporting facilities. This is a single solution on the market that support so build clusters with dynamic allocation of build resources and build server inventory. Only high price for the product stops it from being a leader.

---

## 2.3 Scope

The scope of this analysis is limited by the explicit lists of Build Scripting Engines and Build Automation Systems for evaluation, including the following products:

Build Scripting Engines:

- Ant
- SCons
- NAnt
- MSBuild.

Build Automation Systems:

- CruiseControl
- TeamCity
- Build Forge
- Anthill
- LuntBuild

Solutions are evaluated basing on the list of requirement and criteria specified by the customer and listed in sections "[Customer Needs](#)" and "[Estimation Criteria](#)" correspondingly.

---

## 2.4 Definitions, Acronyms and Abbreviations

SCM	Source Control Management
BSE	Build Scripting Engine
BMT	Build Management Tool

---

## 3. Introduction to Build Management Software

The set of build-helper products can be divided into two classes – build scripting engines and build management tools.

The purpose of build scripting engines is to provide a language or a meta-language that compactly defines basic build steps. Typically, a build script describes a sequence of tasks that should be taken to get the project files compiled, validated against a set of automated tests and other quality tools (such as code checkers and code coverage), and deployed to some location. The language may also provide basic control flow primitives supported by mainstream programming languages such as loops, conditional jumps, etc. One of the key features in build scripting languages is extensibility, which is usually about creating new tasks. Build scripting engines interpret scripts written in build scripting languages, resolve cross-task dependencies and run script tasks in a predefined order. Often build scripting engines can be supplied with extension tools that may extend its core list of tasks or provide an upper-level more convenient, language that is directly translated into the underlying script language, or use underlying engine libraries to run the script.

A superset for build scripting engine toolset is build management tools that have their own meta-language, provide some environment for organizing builds and a scheme for build artifacts storage. Build management tools usually utilize build scripting engines and provide useful add-ons to their basic task list. In fact, build automation tools can be considered as build management best practices consolidators that are very helpful at solving common troubles when delivering builds and releases. To capture it shortly – build management tools are tools that provide a comprehensive build solution over native or custom build scripting engine sacrificing BSE flexibility to a process orientation and time save. BMT may also provide rich reporting facilities; build resources management, wide spectrum of notification options, security and role-management for access to build artifacts, different build scheduling strategies, support for consistent builds and releases automation and execution in globally distributed software environments, and some other features.

---

## 4. Customer Needs

During review of Customer's project architecture, several problems concerned with building process were discovered:

- Customer uses Maven product and build process depends on Maven repository. It means that there is no opportunity to get sources directly from Perforce and build - users need repository, which is not in Production state and cannot provide 24X7-work guarantee.
- There is no any strategy for storing old builds in safe way, so there is risk to lose releases due to disk issues.
- Database size is too large and hard to refresh and there is no process of database preparing for automated testing. This leads to situation when the functionality of the tested system is not fully covered by the automated tests.
- SQL scripts have to be tested several times per day and there is not automated tool that simplifies this task.
- Customer uses FTP is used in case of Master Install (for Database issues) and Maven in case of Java. There is no common approach to prepare both parts of the system.

To solve the above mentioned problems we need to make research of Build creating tools that should have the following abilities and features:

- Ability to be easily integrated with Version Control systems
- Ability to be High available (24X7) in distributed environment, without any single point of failure.
- Clear strategy for supporting old builds (archives, possibility to build any old build)
- Ability to provide convenient way for components to be depended from different versions of 3rd Party libraries. For example, it should be allowed for component A to depend on Hibernate version 1 and for Component B to depend on Hibernate version 2
- Ability to be easily integrated with Automation test tools, and coverage systems
- Easy way to assemble and deploy components developed via different languages (PL/SQL, Java, C++)
- Support of different protocols used for deploying: FTP, HTTP etc.

---

## 5. Estimation Criteria

For evaluation, we need to divide the build management tools into two classes: build scripting engines and build management tools. According to requirement specified in the [“Customer Needs”](#) section, the following options were defined for each class of tools:

For Build Scripting Engines (BSE) the following criteria are used:

- **Scripting options**
  - Loops, Conditions
  - Tasks ordering
  - Parallel task execution
  - Script extensions
- **Third-party applications collaboration**
  - SCM systems
  - QA tools
  - Compilers
  - Database connectivity
- **Extensibility options**
  - Custom scripts inlining
  - Custom scripts call
  - Extension API
- **Community support**
  - Documentation/Articles
  - Books
  - IDE integrations
  - Script generators
  - GUI management consoles
  - High-level script languages
  - Commercial support.

For Build Management Tools (BMT) the following criteria are used:

- **Build process management**
  - Dependency management
    - Dependency ranges
    - Version comparison
    - Conflict resolution
    - Snapshots
    - Profiles
    - Scope
    - Multiproject
  - Distributed builds
  - Parallel builds
  - Firing builds
    - Manually forced
    - SCM triggered
    - SCM poll based
    - Custom scheduling
  - Remote builds management
  - Dynamic allocation of build resources
  - Build server inventory
  - Permissions and security model
- **Web interface**
  - View changesets
  - Add new projects
  - Clone projects
  - Delete projects
  - Modify projects
  - Kill builds
  - Pause builds

- Access to build artifacts
- Search in builds
- Historic graphs
- Self-updating web pages
- Multiproject view
- Add /remove agent machines
- **Third-party tools integration**
  - Build Scripting Engines
  - QA Tools
- **Publishing**
  - Email
  - FTP
  - Instant Messengers
  - Custom client utility
- **Price.**

## 6. Build Scripting Engines Comparison

This section contains description of build scripting engines selected for current research.

### 6.1 Apache Ant 1.7.0

<http://ant.apache.org/index.html>

Apache Ant is a Java-based build tool that can be extended using Java classes. All build procedures are configured in the XML-based configuration files. The build process fires by calling out a target tree compiled from the configuration settings in the configuration file where various tasks get executed. Each task is run by an object that implements a particular Task interface.

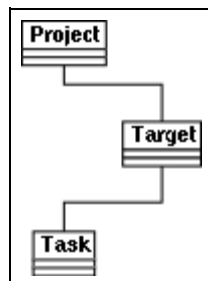
<b>Scripting options</b>	Good
<b>Third-party tools collaboration</b>	Excellent
<b>Extensibility options</b>	Excellent
<b>Community Support</b>	Excellent
<b>Total Evaluation</b>	★★★★★

Ant was conceived by James Duncan Davidson while turning a product from Sun into open source. That product, Sun's reference JSP/Servlet engine, later became Apache Tomcat. The closed source 'make' tool was used to build it on the Solaris Operating Environment, but in the open source world there was no way of controlling which platform was used to build Tomcat. Ant was created as a simple, platform independent tool to build Tomcat from directives in an XML "build file". From this humble beginning, the tool has gone on to become more ubiquitous - and perhaps more successful - than the Tomcat product for which it was created. Ant (version 1.1) was officially released as a stand-alone product on July 19, 2000.

#### Architecture

The system architecture of Ant as in other tools is based on the concept of task. Meanwhile there are some other architectural concepts that make Ant a flexible easily customized build tool that is widely used in Java world today.

All Ant tasks hosted into the 'target' element, which is a part of the 'project' entity (see the picture below).



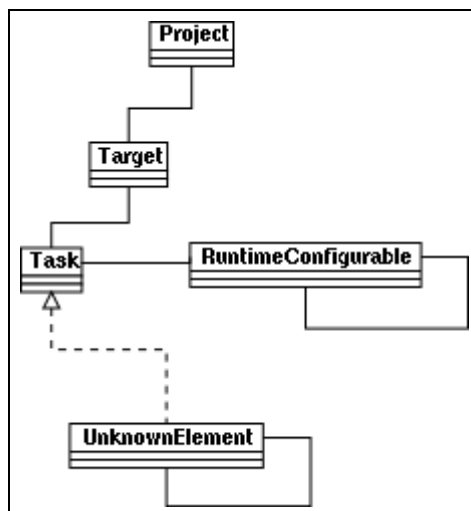
Ant engine supports nested elements and different data types that can be mapped to appropriate classes.

Late versions of Ant introduced the concept of dynamic tasks. The `RuntimeConfigurable` class was introduced to store the information about the build file structure. In effect, this class is the foundation of a lightweight Ant-specific DOM. Each `RuntimeConfigurable` represents a single non-`<target>` node in the build file, whether it is a task or a nested element. All aspects of the node are stored in the `RuntimeConfigurable` instance - the node's attributes, text content and child nodes (as other `RuntimeConfigurable`) instances. The name, `RuntimeConfigurable`, is probably a slight misnomer since it is not actually configurable in the sense of Ant configuration. It, in fact, maintains the configuration information from the build file and is eventually applied to tasks and nested elements, which are the real configurable components.

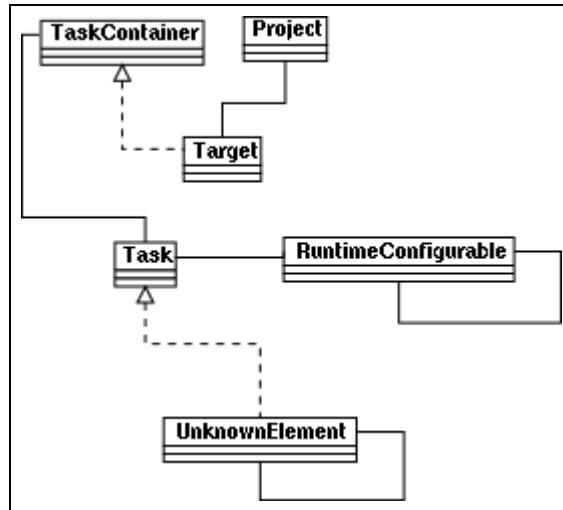
Each task in Ant has a reference to its `RuntimeConfigurable` instance. Prior to the task being executed, it would need to be configured from its `RuntimeConfigurable` instance. This was done by calling the somewhat oddly-named method `maybeConfigure()`. The standard task implementation of the `maybeConfigure()` method passed control to the `maybeConfigure()` method of its `RuntimeConfigurable` which would configure the task and its nested elements from its stored information and its child `RuntimeConfigurables`.

With the build file information stored in the `RuntimeConfigurables`, all that was left to do was to support tasks whose class was not known at parse-time. Since the basic Ant model of project-target-task could not be changed, a new class `UnknownElement` was introduced to allow the model to support storing this information.

`UnknownElement` extends `Task`, allowing it to be stored in the Ant object model. It is not, however, really a task - it is a placeholder for something which might become a task, a nested element within a task or something else entirely (Ant would later support dynamic definition of data types, which also are represented by `UnknownElements`). See the picture below for graphical representation of this architecture.



Task Containers were introduced to allow Tasks to be composed of other Tasks. The two examples of Task Containers that come with Ant, `<sequential>` and `<parallel>`, allow tasks to be grouped for parallel execution. Targets were also generalized as Task Containers (see the picture below).



Task Containers allow you to build logic tasks where the execution of a set of tasks is controlled by another task. Examples are the `<if>` and `<try-catch>` tasks of ant-contrib, which are actually built with instances of the Sequential task.

Task containers turn out to complicate the configuration operation of Ant, because the nested tasks of a Task Container should not resolved and configured when the TaskContainer task is itself configured. Property values may be set or tasks defined by some of the tasks in the TaskContainer.

### Main Functions

The core of Ant's functionality is in tasks provided with it out-of-the-box. Thus all functions available in Ant can be categorized by task categories. Those tasks include:

#### ■ Compile tasks.

`<javac>` - Compiles the specified source file(s) within the running (Ant) VM, or in another VM if the fork attribute is specified.

`<jspc>` - Runs the JSP compiler. The `<javac>` task can be used to compile the generated Java source.

`<rmic>` - Runs the rmic compiler.

#### ■ Archive Tasks

`<zip>`, `<unzip>` - Creates a zipfile.

`<jar>`, `<unjar>` - Jars a set of files.

`<war>`, `<unwar>` - An extension of the Jar task with special treatment web archive dirs

`<ear>` - An extension of the Jar task with special treatment enterprise archive dirs.

### ■ Testing Tasks

`<junit>` - Runs tests from the JUnit testing framework.

`<junitreport>` - Merges the individual XML files generated by the Junit task and applies a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

### ■ Property Tasks

`<dirname>` - Sets a property to the value excluding the last path element.

`<loadfile>` - Loads a file into a property.

`<propertyfile>` - Creates or modifies property files.

`<uptodate>` - Sets a property if a given target file is newer than a set of source files.

### ■ Miscellaneous Tasks

`<echo>` - Echoes text to System.out or to a file.

`<javadoc>` - Generates code documentation using the javadoc tool.

`<sql>` - Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the src attribute, or from between the enclosing SQL tags.

Other Ant features are as follows:

- **Builds History Support.** Different SCM tasks provide support different options for getting old builds. If builds received from source control are compressed by means of some compression utility it can be decompressed with a variety of archive tasks supported by Ant (CAB, ZIP, GZip, Tar and many others algorithms are supported).
- **Database Initialization.** Ant provides database connectivity support via “Sql” task. This task executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file or from between the enclosing SQL tags. Other pros for this task include support of multiple statements execution, transactions support, usage of JVM proxies, “classpath” to the JDBC classes, and the ability to stop or continue ant-script execution in case of database error.
- **Builds Deployment Options.** In the current version Ant builds can be basically deployed to WebLogic servers or JOnAS 2.4 Open Source EJB server.  
  
Builds can be deployed to local or network directory. There is also support for FTP and SCP (copying files to or from a remote server using SSH) channels.
- **Reporting Options.** Ant provides three options for reporting results about build lifecycle. These options are based on so called logging facility that receives a handle to the standard output and error print streams and therefore can log information to the console or the specified log file.

The first three logging (or reporting) options are `DefaultLogger`, `NoBannerLogger` and `TimestampedLogger`. The first one simply makes Ant run normally, the second one removes output of empty target output and the third – acts like the default logger, except that the final success/failure message also includes the time that the build completed.

Other logging options are more functional. They include:

- ***MailLogger*** – captures all output logged through `DefaultLogger` (standard Ant output) and will send success and failure messages to unique e-mail lists, with control for turning off success or failure messages individually.
- ***AnsiColorLogger*** – adds color to the standard Ant output by prefixing and suffixing ANSI color code escape sequences to it. It is just an extension of `DefaultLogger` and hence provides all features that `DefaultLogger` does. `AnsiColorLogger` differentiates the output by assigning different colors depending upon the type of the message. If used with the `-logfile` option, the output file will contain all the necessary escape codes to display the text in colorized mode when displayed in the console using applications like `cat`, `more`, etc.
- ***Log4j*** – passes build events to `Log4j`, using the full classname's of the generator of each build event as the category (build started / build finished, target started / target finished, task started / task finished, message logged, and so on). All start events are logged as `INFO`. Finish events are either logged as `INFO` or `ERROR` depending on whether the build failed during that stage. Message events are logged according to their Ant logging level, mapping directly to a corresponding `Log4j` level.
- ***XmlLogger*** – writes all build information out to an XML file named `log.xml`, or the value of the `XmlLogger.file` property if present, when used as a listener. When used as a logger, it writes all output to either the console or to the value of `-logfile`. Whether used as a listener or logger, the output is not generated until the build is complete, as it buffers the information in order to provide timing information for task, targets, and the project. By default the XML file creates a reference to an XSLT file "log.xsl" in the current directory.
- **Extensibility Options.** Ant provides several extensibility options. It allows to add new tasks by extending the API Java-class `Task`. Then, Ant has several classes that are designed to be extended for different purposes, such as handling JDBC configuration needed by SQL type tasks. Other options include extending `pack` and `unpack` tasks, `MatchingTask` – an abstract task that should be used by all those tasks that require to include or exclude files based on pattern matching, `DispatchTask` – for tasks that have multiple actions and `AbstractCvsTask`.

Also, Ant is capable of generating build events as it performs the tasks necessary to build a project. Listeners can be attached to Ant to receive these events. This capability could be used, for example, to connect Ant to a GUI or to integrate Ant with an IDE.

The other way to extend Ant through Java is to make changes to existing tasks, which is positively encouraged. Both changes to the existing source and new tasks can be incorporated back into the Ant codebase, which benefits all users and spreads the maintenance load around.

### Helper tools and extensions

- **Jelly** – XML-executable in Ant scripts
- **AntContrib** – Ant tasks description in Groovy
- **Gant** – Ant tasks description in Groovy
- **Gosling** – Ant tasks description in Java
- **Rant** – distributed build system that allows an Ant build file to launch builds on other systems and receive exceptions should they occur.
- **Invicta** – generates powerful build scripts (Apache ANT's), while hiding their complexity.
- **Savant** – Savant is an extension to the popular ant build system from the Apache group.
- **GenJar** – specialized Ant task that builds jar files based on class dependencies rather than simply the contents of a directory.
- **JAM** – collection of Ant scripts designed to perform common Java/J2EE build tasks such as compilation, packaging, testing, deployment and J2EE application server control. JAM combines MavenB's high-level project description and repository features (via a Maven-to-Ant bridge) with the low-level capabilities of Ant. By assembling JAM modules, one is able to quickly create sophisticated IDE-independent build scripts. JAM supports various J2EE application servers, XDoclet invocation, JUnit unit testing, Apache Cactus integration testing, UML-based code generation and other technologies.
- **Ivy** – dependency manager

## 6.2 SCons 0.96.1

<http://www.scons.org/>

SCons is an Open Source software construction tool. SCons is distributed under the MIT license, an approved Open Source license. SCons is an improved, cross-platform substitute for the classic `Make` utility with integrated functionality similar to `autoconf/automake` and compiler caches such as `ccache`.

<b>Scripting options</b>	Good
<b>Third-party tools collaboration</b>	Good
<b>Extensibility options</b>	Good
<b>Community Support</b>	Sufficient
<b>Total Evaluation</b>	★★★★

SCons is implemented as a Python script and set of modules, and SCons "configuration files" are actually executed as Python scripts. This gives SCons many powerful capabilities. For example using its scripting capabilities lets to do more sophisticated things in a build: construct lists of files, manipulate file names dynamically, handle flow control (loops and conditionals) in the build process, etc. SCons is written to work with Python version 1.5.2 or any later version. Extensive tests are used to ensure that SCons works on all supported versions. With SCons, you use Python functions to tell a central build engine about your input and output files.

Below there is the example of the output (on a POSIX system) from running the `scons` command for building simple java application:

```
% scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
javac -d classes -sourcepath src src/hello.java
scons: done building targets.
```

The main features of this product are:

- Configuration files are Python scripts – this allows to use the power of a real programming language to solve build problems
- Reliable, automatic dependency analysis built-in for C, C++ and Fortran – no more "make depend" or "make clean" to get all of the dependencies. Dependency analysis is easily extensible through user-defined dependency Scanners for other languages or file types
- Built-in support for C, C++, D, Java, Fortran, Yacc, Lex, Qt and SWIG, and building TeX and LaTeX documents. Easily extensible through user-defined Builders for other languages or file types
- Building from central repositories of source code and/or pre-built targets
- Built-in support for fetching source files from SCCS, RCS, CVS, BitKeeper and Perforce
- Built-in support for Microsoft Visual Studio .NET and past Visual Studio versions, including generation of `.dsp`, `.dsw`, `.sln` and `.vcproj` files
- Reliable detection of build changes using MD5 signatures; optional, configurable support for traditional timestamps
- Improved support for parallel builds – like `make -j` but keeps N jobs running simultaneously regardless of directory hierarchy
- Integrated `Autoconf`-like support for finding `#include` files, libraries, functions and typedefs
- Global view of all dependencies – no more multiple build passes or reordering targets to build everything.
- Ability to share built files in a cache to speed up multiple builds—like `ccache` but for any type of target file, not just C/C++ compilation.
- Designed from the ground up for cross-platform builds, and known to work on Linux, other POSIX systems (including AIX, \*BSD systems, HP/UX, IRIX and Solaris), Windows NT, Mac OS X, and OS/2.

## 6.3 NAnt 0.85

<http://nant.sourceforge.net/>

NAnt is a free .NET build tool. It can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

### Architecture

<b>Scripting options</b>	Good
<b>Third-party tools collaboration</b>	Good
<b>Extensibility options</b>	Good
<b>Community Support</b>	Sufficient
<b>Total Evaluation</b>	★★★★

NAnt is command-line tool, that can be launched manually or by any other tools. It is similar to Apache Ant, but targeted at the .NET environment rather than Java.

NAnt has been tested using the following runtime frameworks:

- Microsoft .NET Framework 1.0
- Microsoft .NET Framework 1.1
- Microsoft .NET Framework 2.0
- Mono (1.0 and 2.0 profile)

NAnt's build files are XML-based and each build file contains one project and one to many build targets. Within each of those build targets, you specify individual build tasks for your applications. NAnt is a great way to manage and automate your build process. This is obviously handy for developers and configuration managers. For Internet operations staff it provides useful way to package and deploy builds across servers and environments.

### Main Features

This product can be used for automating builds, automating unit testing runs, or for version control. NAnt has no built in GUI interface, and it does not write unit tests for you. Instead, it provides a powerful means of scripting these tasks so that they are performed automatically with a single command. It allows to easily write scripts that work unchanged on both Linux and Windows. NAnt provides an intuitive and powerful means of controlling the build process, and of automating common tasks encountered during the development process. It comes with a rich set of predefined tasks that cover most developer's needs.

The main features of this product are the following:

- **NAnt Properties.** A project can have a set of properties. They might be set in the buildfile by the `<property>` task, or might be set outside NAnt. A property has a name and a value. Properties may be used in the value of task attributes. NAnt has set of built-in properties.
- **Built-in Functions.** NAnt provides a rich set of built-in functions, that allow you to:
  - Manipulate strings
  - Manipulate date/time values
  - Manipulate path names

- Read the properties of files/directories
  - Access current build information and more.
- **Expressions Support.** Expressions are simple, yet powerful mechanism that allows to write advanced formulas to be used in task arguments and conditions that direct the build process. Expressions can access project properties and call built-in or user-defined functions.
- **Loggers and Listeners.** NAnt has two related features to allow to monitor the build process: listeners and loggers. A listener is alerted of the following events: build started, build finished, target started, target finished, task started, task finished, message logged. Loggers extend the capabilities of listeners and add the following features:
  - Receiving a handle to the standard output and error print streams and therefore can log information to the console or the -logfile specified file.
  - Logging level (-quiet, -verbose, -debug) aware.

See [“Reporting options”](#) below for more details.

- **Version Control Support.** NAnt uses CVS for source control and supports checkout, tagging, authentication, and updates functions by native commands. NAntContrib adds extra support for ClearCase, P4 Perforce, PVCS, Surround SCM, StarTeam, SVN, VSS. It allows to create tasks for signing delay-signed .NET assemblies or re-signing existing assemblies. There are also built-in tasks for zipping and unzipping file sets – a great feature for putting build into a distributable package for deployments. It also supports Internet resources and retrieving them via an HTTP proxy.
- **Logging.** NAnt allows to monitor the build process via NAnt’s listener and logger features. Listeners are alerted to different events throughout the build process, such as starting and stopping tasks and when a message has been logged. Loggers allow you to log information to the standard output, error print streams, or a specified log file. There are three levels of logging: quiet, verbose, and debug.

NAnt is definitely not limited to compiling .NET-based applications. Though the application requires .NET to function, you can compile unmanaged C/C++ programs, as well. It integrates with the NDoc code documentation generator, so you can specify as a build task an NDoc task that will generate class library documentation from .NET assemblies and the C# compiler’s XML documentation.

- **Tasks Support.** A task is a piece of code that can be executed. A task can have multiple attributes (or arguments, if you prefer). The value of an attribute may contain references to a property. These references will be resolved before the task is executed. NAnt provides support for the following tasks:
  - Compile and link tasks, like <al>, <csc>, <jsc> etc. allows to compile C#, J#, Visual Basic.NET, Visual C++, ASP.NET applications, run Assembly Linker, etc.
  - File system tasks like <attrib>, <copy>, <delete>, <touch>, <loadfile> etc, represents file processing functions.
  - .Net specific tasks <asminfo >, <delay-sign>, <license>, <regasm>, <regsvcs>, etc. allows using .Net Framework functions for assembly, application and servicing processing.

- Xml specific tasks `<xmlpeek>` and `<xmlpoke>` allows XML-file processing. Task `<style>` processes a document via XSLT.
- Archive processing tasks `<zip >`, `<unzip>` supports zip files creating and using.
- Tasks `<nunit>` and `<nunit2>` runs tests using the NUnit frameworks.
- Task `<ndoc>` runs NDoc V1.3.1 to create documentation.
- And lot of other tasks, like regular expression evaluation, Windows Registry reading, etc available.
- **Executing Task.** NAnt runs the specified program or command with the specified arguments. This task is useful when a specific NAnt task for the job that you want to perform is not available.
- **NAntContrib.** NAntContrib is the free open project for tasks and tools that haven't made it into the main NAnt distribution yet or for whatever reason don't belong there.
- Different tasks like sql processing, BizTalk interacting, generating statistics from source code, icrosoft HTML Help 2.0 Project compiling, XML validating and more and more.
- **Reporting Options.** NAnt provides Loggers that receive a handle to the standard output and error print streams and therefore can log information to the console or the `-logfile` specified file. There are three build-in loggers:
  - **NAnt.Core.DefaultLogger:** The default logger logs information to the console or the specified file.
  - **NAnt.Core.MailLogger** Extends DefaultLogger such that output is still generated the same, and when the build is finished an e-mail can be sent.
  - **NAnt.Core.XmlLogger** Generates output in XML format.

Additional reporting functionality can be added by using other managing systems, like Cruise Control.

- **Extensibility Options.** NAnt currently provides mechanisms to make third-party extensions available for use in build scripts. Extensions can be divided in the following categories:
  - Tasks
  - Filters
  - Functions
  - Global Types.

NAnt mechanisms use .NET reflection to scan one or more assemblies for the presence of extensions. When found, these extensions are registered in the NAnt type system, and become available for all build scripts (in the current NAnt instance).

## 6.4 MSBuild

<http://msdn2.microsoft.com/en-us/library/wea2sca5.aspx>

The Microsoft Build Engine (MSBuild) is the new build platform for Microsoft and Visual Studio.

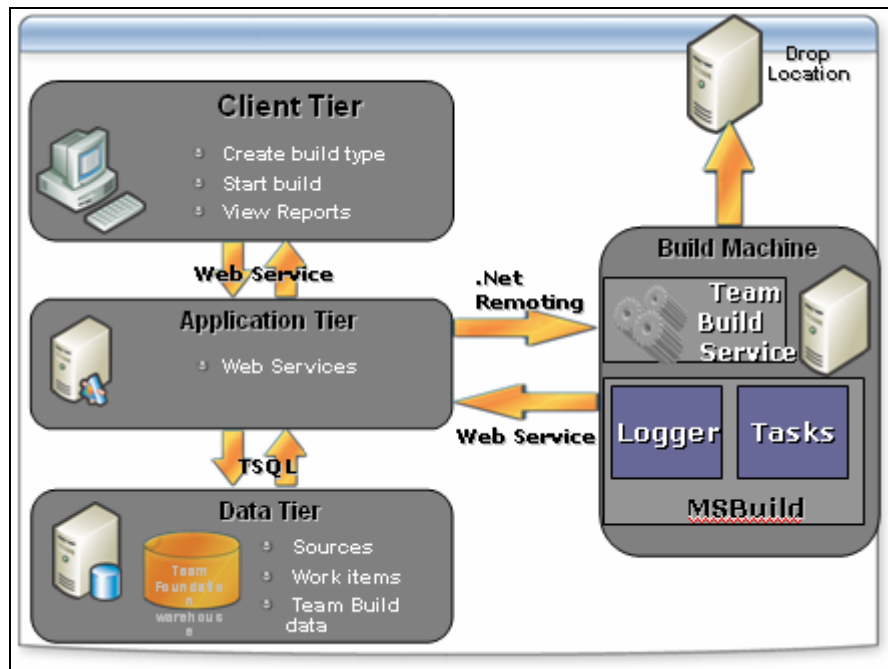
### Architecture

MSBuild is a core component of the .NET Framework redistributable. It is a command-line tool, that can be launched manually, from Visual Studio IDE of Team Foundation Build Server. The Visual Studio Team System also depends on MSBuild to perform the actual Team Builds via the Visual Studio Team Foundation Server.

<b>Scripting options</b>	Good
<b>Third-party tools collaboration</b>	Satisfactory
<b>Extensibility options</b>	Good
<b>Community Support</b>	Sufficient
<b>Total Evaluation</b>	★★★

Team Foundation Build provides the functionality of a public build lab and is part of Visual Studio Team Foundation Server. With Team Foundation Build, enterprise build managers can synchronize the sources, compile the application, run associated unit tests, perform code analysis, release builds on a file server, and publish build reports. Build result data is propagated to the warehouse for historical reporting. Team Foundation Build works with other Team System tools during the build process, including source control, work item tracking, and with test tools.

Architecture of Team Foundation Build is present on the picture below:



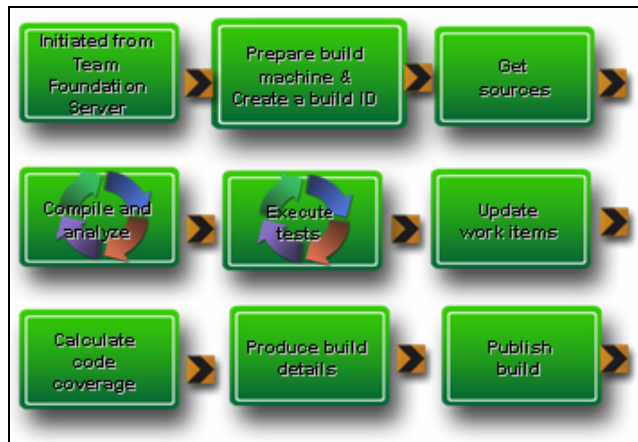
MSBuild is completely transparent with regards to how it processes and builds software, enabling developers to orchestrate and build products in build lab environments where Visual Studio is not installed.

## Main Features

The main features of this product are the following:

- **MSBuild Properties.** Properties are key/value pairs that can be used to configure builds. Properties are useful for passing values to tasks, evaluating in conditions, and storing values that will be referenced throughout the project file. MSBuild provides a set of reserved properties that store information about the project file and the MSBuild binaries
- **MSBuild Items.** Items represent inputs into the build system and are grouped into item collections based on their user-defined collection names. These item collections can be used as parameters for tasks, which use the individual items contained in the collection to perform the steps of the build process.
- **MSBuild Conditional Constructs.** MSBuild provides a mechanism for conditional processing with the *Choose*, *When*, and *Otherwise* elements.
- **MSBuild Batching.** MSBuild has the ability to divide item collections into different categories, or batches, based on item metadata, and run a target or task one time with each batch.
- **MSBuild Transforms.** A transform is a one-to-one conversion of one item collection into another. In addition to allowing a project to convert item collections, transforms allow a target to identify a direct mapping between its inputs and outputs. This topic explains transforms and how MSBuild uses them to build projects more efficiently.
- **MSBuild Logging.** MSBuild loggers provide a way to customize the reporting of build events, messages, warnings, and errors. Loggers can display information in the console window, write to XML or a text file, or enter build data into a database. In addition, specific tasks, like *Warning* and *Error* are available.
- **MSBuild Tasks.** Tasks provide the code that runs during the build process. The library of common tasks provided with MSBuild includes the following:
  - *Compile and link tasks* allows to compile *C#, J#, Visual Basic.NET, Visual C++, ASP.NET* applications, run *Assembly Linker*, etc.
  - *File system tasks* like *Copy, Delete, MakeDir, FindUnderPath, ReadLinesFromFile, WriteLinesToFile* etc, represents file processing functions.
  - *.Net specific tasks* *AssignCulture, GenerateApplicationManifest, GenerateDeployManifest, GenerateResource, GetAssemblyIdentity, GetFrameworkPath, RegisterAssembly, ResolveAssemblyReference, ResolveKeySource, SignFile* etc. allows to use .Net Framework functions for assembly and application processing.
- **Executing Task.** MSBuild runs the specified program or command with the specified arguments. This task is useful when a specific MSBuild task for the job that you want to perform is not available.

- **Team Build Functions.** MSBuild based Team Build Process provides significantly more complicated functions. Team Build Process is illustrated on the following picture:



- **Reporting Options.** MSBuild logs build process to plain text file. Team Foundation Server has own reporting features, based on collected in data warehouse statistics and MS SQL Server Reporting Services. Notifications can be sent by e-mails.
- **Extensibility Options.** MSBuild can be easily extended by custom tasks. There are two approaches available when implementing a task:
  - Implement the ITask interface directly.
  - Derive your class from the helper class, Task, which is defined in the Microsoft.Build.Utilities.dll assembly. Task implements ITask and provides default implementations of some ITask members. Additionally, logging is easier.

Microsoft.Build.Framework, Microsoft.Build.Tasks and Microsoft.Build.Utilities namespaces available to extend and interact with MSBuild engine.

---

## 6.5 Comparison Table

We compared the build scripting engines according to estimation criteria (described in section "[Estimation Criteria](#)") and created the comparison table for each group of comparison parameters (see [Appendix A](#)) to facilitate the analysis.

---

## 7. Build Management Tools Comparison

This section provides description build management tools selected for current research.

---

### 7.1 Maven 2.0.5

<http://maven.apache.org/>

Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Maven helps managing:

- Builds
- Documentation
- Reporting
- Dependencies
- Software Configuration Management
- Releases
- Distribution
- Project structure.

Maven advocates using a standard directory layout for source files, resources, and intermediate artifacts. Though every setting may be overridden, Maven assumes the files are organized according to the recommend directory layout:

Files	Explanation
<b>SRC/MAIN/JAVA</b>	Application/Library sources
<b>SRC/MAIN/RESOURCES</b>	Application/Library resources
<b>SRC/MAIN/FILTERS</b>	Resource filter files
<b>SRC/MAIN/ASSEMBLY</b>	Assembly descriptors
<b>SRC/MAIN/CONFIG</b>	Configuration files
<b>SRC/TEST/JAVA</b>	Test sources
<b>SRC/TEST/RESOURCES</b>	Test resources
<b>SRC/TEST/FILTERS</b>	Test resource filter files
<b>SRC/SITE</b>	Site

Files	Explanation
LICENSE.TXT	Project's license
README.TXT	Project's readme
TARGET	

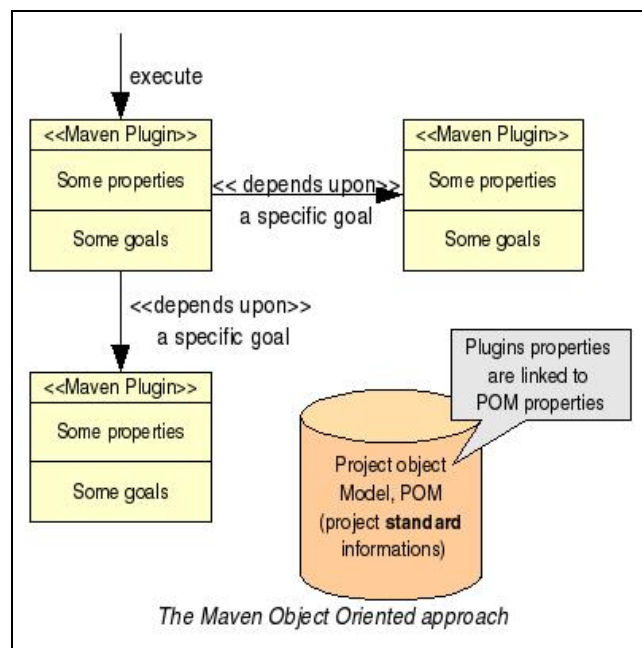
Maven promotes a standard build lifecycle where various plug-ins may hook into in order to customize or enhance it. Standard lifecycle phases:

- **Validate** the project is correct and all necessary information is available, generate any source code for inclusion in compilation.
- **Process the source code.** For example, to filter any values, generate resources for inclusion in the package.
- **Process resources.** Copy and process the resources into the destination directory, ready for packaging compile the source code of the project.
- **Process classes.** Post-process the generated files from compilation, for example to do byte code enhancement on Java classes.
- **Generate test source code** for inclusion in compilation.
- **Process-test-sources** process the test source code, for example to filter any values.
- **Generate test resources.** Create resources for testing.
- **Process test resources.** Copy and process the resources into the test destination directory.
- **Compile the test source** code into the test destination directory.
- **Run tests** using a suitable unit testing framework. Does not require the code be packaged or deployed.
- **Package.** Take the compiled code and package it in its distributable format, such as a JAR.
- **Integration test.** Process and deploy the package if necessary into an environment where integration tests can be run.
- **Verify.** Run any checks to verify the package is valid and meets quality criteria.
- **Install** the package into the local repository, for use as a dependency in other projects locally. Deploy in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

## The Project Object Model (POM)

The POM is defined in a pom.xml file that contains all the information necessary to build the project, generate reports and configure plug-ins. Maven supports the notion of a single artifact per project. This artifact may be a JAR, WAR, EAR, POM (just a pom.xml file) or some other special artifact. Maven stores project artifacts and third-party dependencies in repositories. Artifacts are identified by group id, artifact id, and version. Artifacts are typically jar files, but can also be war and ear files as well as pom.xml files. There are three types of repositories: The local repository on the developer machine, shared internal repositories (corporate repository), and public repositories. All repositories are basically directories with a certain structure.

See the picture below for POM architecture:



## Dependency management

Maven comes with a mechanism that project's clients can use to download any JARs required for building the project from a central JAR repository. Maven 2 resolves dependencies transitively. If A depends on B and B depends on C then you need both B and C in order to use A. However, A only needs to specify its direct dependencies (B in this case). Maven determines that C is also necessary and makes it available. Maven executes multi-module builds in the right order building dependencies and installing them into the local repository before building the modules that depend on them. Dependencies in Maven 2 have scope. The scope concept recognizes various dependencies are needed only in certain times during the development/deployment cycle.

The best example is junit, which is necessary during test time only. It is not required to build your project and it is not required at deployment time. There are five scopes:

- **Compile scope** means the dependency is necessary to build your project.
- **Test scope** means the dependency is necessary to test your project.
- **Provided** means the dependency is necessary at runtime but will be provided by the container (Servlet and JSP APIs are provided by Tomcat, for instance).

## Plug-ins

Maven does its work by running plug-ins in the core engine. Plug-ins is the mechanism to extend and customize Maven. A plug-in is basically a set of goals, a.k.a. MOJOs (Maven Old Java Objects) that are bound to a specific lifecycle phase. Maven executes the appropriate mojo for each phase in the build lifecycle as it chugs along the build. The mojos get their input from the POM.

Main plug-in types include:

- **Core.** Plug-ins corresponding to default core phases (ie. clean, compile). They may have multiple goals as well.
- **Packaging types / tools.** These plug-ins relate to packaging respective artifact types.
- **Reporting.** Plug-ins which generate reports, are configured as reports in the POM and run under the site generation lifecycle.
- **Tools.** Miscellaneous tools available through Maven by default.
- **IDEs.** Plug-ins that simplify integration with integrated developer environments.

## 7.2 QuickBuild 1.2.3

<http://www.pmease.com/app.do>

QuickBuild is the professional version of the popular open source build server, Luntbuild. It is a cross-platform build automation and management server that unifies all levels of builds such as continuous integration, daily build, QA and release build. Build can be promoted from one level to another, triggering desired steps such as sending notification, source code re-labeling, etc. QuickBuild enables a build-centric process to drive the smooth delivery of builds between different groups of the team, for example, from developer to QA, from QA to release manager, etc.

<b>Build Process Management</b>	Excellent
<b>Web Interface</b>	Good
<b>Third-party tools integration</b>	Satisfactory
<b>Publishing</b>	Good
<b>Price</b>	Sufficient
<b>Total Evaluation</b>	★★★

QuickBuild provides the following feature list:

- **User interface**
  - Tree organized build configurations make navigation through large number of projects easy.
  - Utilizes concept of inheritance and overriding to reduce complexity of configuring and managing large number of projects.
  - Ability to set up the build configurations, to monitor the build status and the build logs, to download/upload build artifacts.
  - Ability to set up project dependencies.

- Searches builds by criteria such as version, status, date, configuration, etc.
- Ability to rebuild particular version.
- Ability to build against specified label, or branch.
- Access to history builds.
- Keeps most recent number of builds by days or count.
- Ability to define build queues to control concurrent number of builds.
- Ability to stop running builds.
- OGNL expression helpers.
- A dashboard page can be used to perform all routine jobs.
- Error/Warning info in the build log is highlighted.
- Context sensitive help, intuitive error reports.
- **Version Control System support:** CVS, Accurev, Visual SourceSafe, Subversion, ClearCase base mode, ClearCase UCM mode, StarTeam, Perforce, File system based repository.
- **Supported builders**
  - Apache Ant, Nant, Apache Maven
  - Feeds QuickBuild managed build version to Maven.
  - Artifacts published to Maven repository are accessible from QuickBuild's web interface.
  - Rake builder.
  - Any other build tools with command line interface.
- **Build scheduling**
  - Periodical scheduling
  - Cron based scheduling
  - Manually
- **Build promotion**
  - Be able to promote build from one configuration into another (for example, from QA to Release). The new build can take next build version of the destination configuration, or just use the original build version. Combined with the build permission management and build notification, builds can flow smoothly between different roles of your team.
  - Source code of original build can be re-labeled as part of the build promotion process.

- Promoted build can use artifacts of original build directly, or can be re-produced using exactly the same set of source code of the original build.
- Build can be promoted between different machines.
- With the build promotion feature, one can select a particular build and run additional steps to further process it (for example, deploy it into the integration test machine).
- **Build notification**
  - Sends build notification through Email, Jabber, MSN messenger, or Google Talk.
  - Ability to customize content of the build notification through Velocity.
  - Build notifications can be sent to particular users/groups, user who triggered the build, or developers who checked in since the last build or the last successful build.
  - Build log or revision log can be attached to email notification.
  - Error lines from the build log can be extracted and inserted into email notification directly.
  - Users can self-manage their subscriptions to certain build events.
  - RSS for build search results, so that user can be notified of new builds satisfying the search criterion.
- **Build procedure**
  - Remote and parallel build support:
    - Ability to set up build for projects which consist of components that should be built on different OS platforms.
    - Ability to build different parts of a project on different build machines simultaneously to speed up build process.
    - Ability to build projects at one machine, and publish generated build results on another machine for automation/smoke tests. Test report can be collected and published back to the main build machine for later review.
  - Fully customizable build process, you can define your own build steps.
  - Ability to control what steps should be executed serially, and what steps can be executed simultaneously (to speed up the build process).
  - Ability to checkout from multiple Version Control Systems, or to build with multiple builders.
  - Ability to create label in Version Control Systems based on user-defined conditions.

- **Build version management**
  - Ability to define versions and increase the version number over time.
  - Ability to use date and iteration as build version.
  - Ability to define almost any kind of version strategy by using OGNL.
  
- **Authentication and authorization**
  - User and group management.
  - Anonymous access to configurations can be enabled or disabled.
  - User self-registering can be enabled or disabled.
  - Be able to authenticate users through LDAP protocol. Various LDAP server are supported including Microsoft Active Directory. User group information in LDAP server can also be retrieved.
  
- **Data management**
  - Ability to backup QuickBuild data to XML file, or restore them from XML file.
  - Ability to migrate QuickBuild data between different databases (such as Microsoft SQL Server, MySQL and HSQLDB).
  - Ability to migrate QuickBuild data from Luntbuild.
  - Database auto-backup.
  
- **Programming interface:** Java and CSharp web service API implemented with Hessian protocol.



### 7.3 TeamCity 2.0

<http://www.jetbrains.com/teamcity/>

TeamCity is an innovative team environment aimed at boosting the productivity of software development teams by removing typical bottlenecks in the development process that often bog teams down. It identifies necessary but redundant or time-consuming things in the team development process and then provides intelligent, automated solutions to reduce a human error, miscommunication, and other issues.

<b>Build Process Management</b>	Excellent
<b>Web Interface</b>	Excellent
<b>Third-party tools integration</b>	Excellent
<b>Publishing</b>	Good
<b>Price</b>	Good
<b>Total Evaluation</b>	★★★★★

The key features provided by TeamCity include the following:

- **Continuous Integration & Testing.** TeamCity automates the process of running tests and integrating and building code changes made by team members. TeamCity takes care of complex test running, error handling and notifications, and integration and building chores. All a developer needs to do is submit changes to version control and continue working.

Tests and builds run *remotely*, as often as you want, on the innovative *Build Grid* which can utilize multiple computing resources anywhere on your network, so developers' computers are never bogged down. Failed tests and builds are immediately reported to the people who most need to know, and are visible online for the entire team.

- **Efficient, Effective Build Management.** Flexibility in building is the name of the game in TeamCity. Define and run different build types (nightly, periodic, etc.) with different *build configurations* for any project, including customizable build triggers that provide unparalleled convenience and flexibility in managing the build process.

You can set up builds to run on any number of computers anywhere on your network. When a build is triggered (either automatically or manually) TeamCity processes it on any available agent whose environment is compatible with the build configuration. If no resources are free, the build enters a queue and is processed automatically as soon as resources are available. All this can be easily set up and monitored using the convenient web interface.

- **Timely Automated Team Communication.** You know the scenario: the build broke, but it's a while before anyone realizes. Then nobody is quite sure whose code broke it, but everybody assumes someone else is on top of the situation. Valuable time is lost sorting it all out.

A customizable array of notifications ensures that anyone who submitted code that could have broken the build is notified the moment something fails, and the failure shows up immediately in the web interface. If nobody has taken on responsibility for a fix, this is evident for all to see at a glance. A quick click enables any developer to not only take responsibility for the fix, but to effortlessly inform the entire team that someone has (and also let them know when the fix is complete).

- **Server-side Code Analysis.** In addition to having TeamCity run unit tests when creating builds, builds configured as an IntelliJ IDEA project file can also have TeamCity remotely run a set of IntelliJ IDEA code inspections. Any problems found are displayed in both web interface and the IntelliJ IDEA editor just like local inspections.
- **Code Coverage Analysis & Reporting.** Stop guessing how much of your projects' code is covered by tests. TeamCity can optionally run a code coverage analysis for Ant or IntelliJ IDEA projects. Just tick an option in an appropriate build configuration, and the build results page in the web interface will include an overview of coverage statistics plus a link to a detailed report. Results are also accessible in IntelliJ IDEA when using the TeamCity plugin.

- **Web-based Interface & Project Dashboard.** A late-model web browser is all anyone needs to work with TeamCity - you are not tied to any IDE or programming language. The rich web interface handles all administration and everyday user tasks. The project dashboard helps all team members stay up-to-date on the status of current builds, shows newly failed tests, and enables users to trigger builds and rearrange the build queue.

The interface is specially tuned to provide effective and painless navigation through large amounts of test/build-related information.

- **Extensibility.** TeamCity is extensible via a Java™ API. Developers can extend TeamCity functionality such as notifications into different clients, integration with development environments, support other version control systems, and more. A robust extension plugin for IntelliJ IDEA 6.0 is already available for IntelliJ IDEA 6.0 (see below) and more integrations with popular IDEs are in the works.
- **Tight IDE Integration.** TeamCity integrates with IntelliJ IDEA 6.0 via a plugin that provides the TeamCity features most essential to developers right in the IDE. TeamCity's reports and notifications are immediately available, and these provide convenient one-click navigation to source code, or to the web interface. There's also some special extended functionality (see next sections).

For .NET developers, TeamCity offers a possibility to open the stacktrace of failed tests in Microsoft Visual Studio 2005.

- **Pre-tested (Delayed) Commit.** This revolutionary feature of the TeamCity plugin for IntelliJ IDEA eliminates the all-too-frequent "5 o'clock check-in" syndrome. Submitted code changes first go through testing. If all tests succeed, TeamCity automatically submits the changes to version control. From there, they will automatically be integrated into the next TeamCity build. If any test fails, the code is *not* committed, and the submitting developer gets a notification by his/her preferred means. The defective code never gets into a build, so the process of the entire team is not affected in any way.
- **Remote Run for Personal Builds.** Another feature of the TeamCity plugin, this one removes all fear from experimentation with the code base, and virtually eliminates the chances of an IntelliJ IDEA developer submitting build-breaking code. Developers can run personal builds remotely on TeamCity's build grid just the same as a "regular" build except that changed code is never committed to version control. If a personal build fails, no "real" build is affected, so there is never any adverse impact on the team. The developer receives immediate notification of problems and can then work on a fix.

When changed code passes muster in a personal build, the developer can commit it with confidence that it won't break a real build. For the same reason, there's never any reason not to experiment as much as you want because you never risk causing problems for the team or the process.



## 7.4 CruiseControl 2.6.1

<http://cruisecontrol.sourceforge.net/>

CruiseControl is a Java-based framework for a continuous build process. It includes, but is not limited to, plugins for email notification, Ant, and various source control tools. A web interface is provided to view the details of the current and previous builds. It allows one to perform a continuous integration of any software development process.

CruiseControl is free, open-source software, distributed under a BSD-style license. It was originally created by employees of ThoughtWorks for continuous integration on a project they were working on. It was later extracted into a stand-alone application.

<b>Build Process Management</b>	Excellent
<b>Web Interface</b>	Good
<b>Third-party tools integration</b>	Sufficient
<b>Publishing</b>	Excellent
<b>Price</b>	Excellent
<b>Total Evaluation</b>	★★★★

CruiseControl is composed of the two main modules: build loop and reporting.

### Build Loop.

CruiseControl runs a build loop, which is designed to run as a daemon process that periodically checks your source control tool for changes to your codebase, builds if necessary, and sends out a notification regarding the status of the build.

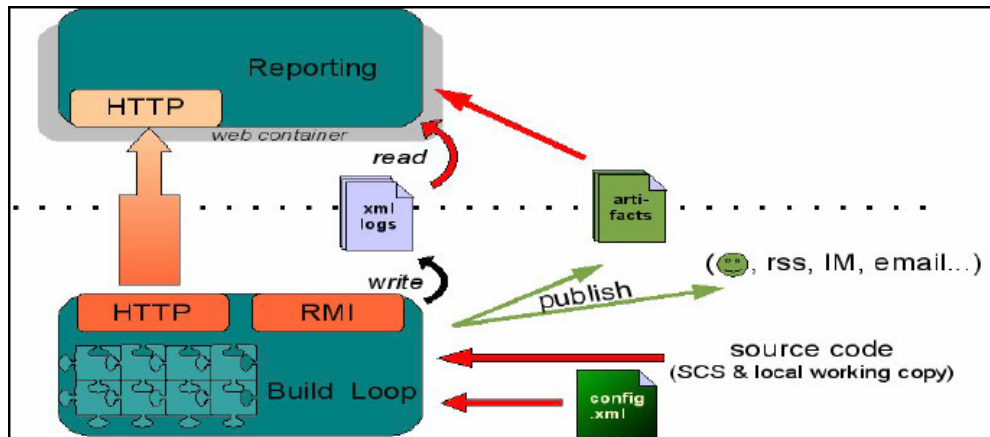
This is a loop that you define in the configuration XML file by defining project(s) containing information about timing, logging, and notification information. You can have interested parties notified by e-mail, or use the Reporting Application that comes with CruiseControl to make the build and testing results available, or both. There are also many other so-called Publishers available, and you can easily write your own.

You can update a project with new rules/information that are picked up by the loop every time it goes around, depending on your needs. You can define multiple projects in the same configuration file to run (and compile & test) in the same CruiseControl instance. The default is to use a build queue that builds one project at a time, but if you are using a CruiseControl with a version higher than 2.1.6 you can also have projects building in parallel, should you want that configuration. Just keep in mind that you always need to restart CruiseControl if you have added or removed projects, since it will only pick up changes to existing projects on every loop.

The build loop depends on Ant, jakarta's simple, extensible, XML-formatted version of make which is the de-facto standard for java build tools. Once you have your project building with Ant, setting up and using cruise control is twice as easy. You can also use Maven, which is becoming an increasingly popular alternative to Ant.

In addition, because Ant or Maven still allow you to execute legacy makefiles and scripts, CruiseControl only magnifies and enhances the longevity of any dependable, mature build processes you may already have.

**Reporting.** The reporting allows users to browse the results of the builds and access the artifacts (see the picture below for reporting architecture).



This modularity allows users to install CruiseControl where it will best fit their needs and environment.

Using remoting technologies (HTTP, RMI), it is possible to control and monitor the CruiseControl build loop. Those are turned off by default for obvious security reasons.

CruiseControl can be installed from source, or using the all in one binary installation.

The following table provides an overview of tools supported by CruiseControl:

Tools	Comments
<b>SCM Tools</b>	Accurev, Clearcase (dynamic views), Clearcase (snapshot views), Clearcase (UCM), CMSynergy, jCVS, PvcS, StarTeam, Subversion, View CVS, MKS
<b>Code Analysis, Testing, and Metrics Tools</b>	JCSC (Java Coding Standard Checker), PMD, FindBugs, MetricsTab, PHPUnit
<b>Operating Systems</b>	Windows, Unix, LinuxRedHatWay
<b>Reporting Web Application</b>	JSP, Trac plugin
<b>Tools for Monitoring CruiseControl</b>	Lotus Sametime, log4j, Chainsaw, Nagios, Firefox and Thunderbird plugin, kala, Yahoo/Konfabulator Widgets
<b>Development Tools, Environments, and Languages</b>	BEA WebLogic Workshop, C++

## 7.5 AnthillPro 3.2

<http://www.anthillpro.com/html/default.html>

AnthillPro3 is a third generation Build Management Server (BMS). It is built around an embedded workflow engine and a GRID computing engine, making it possible to define and automate processes such as distributed builds, automated tests, promotions, deployments, and more. These capabilities make AnthillPro3 the first Application Lifecycle Automation Server (ALAS).

AnthillPro provides the following out-of-the-box features:

<b>Build Process Management</b>	Excellent
<b>Web Interface</b>	Excellent
<b>Third-party tools integration</b>	Sufficient
<b>Publishing</b>	Good
<b>Price</b>	Sufficient
<b>Total Evaluation</b>	★★★★

- **Continuous Integration.** Support of commit triggers, quiet periods, integrations with testing and issue tracking, and flexible notification schemes makes AnthillPro3 a great continuous integration server. AnthillPro3 works with existing build scripts, whether they be Ant, NAnt, make, maven, or some other type of scripts. It will even launch your IDE in headless mode to perform the build if you don't have a build script. Integrations with testing and test coverage tools mean that you can create custom reports that can be included in your email, IM, or RSS notifications.
- **Distributed Build Farm.** The Central Server and distributed Agent architecture is ideal for distributed and multi-platform builds. Each agent can run multiple builds concurrently. A single build may be split among several tasks each of which can run on a separate Agent.
- **Dependency Management.** AnthillPro allows to configure dependencies between projects. A project may depend on the latest build of a dependency or be locked to a specific version or a status of a dependency. The Bill of Materials (BOM) for each build details the exact artifacts of each dependency that were used. The intelligent Anthill job scheduler creates an entire dependency graph for each build and guarantees that each project in that graph is built only once.
- **Configurable Workflows.** This is a huge feature that separates AnthillPro from every other tool on the market. AnthillPro3 contains an embedded workflow engine that allows you to configure a set of processes that can be applied to Living Builds.

AnthillPro automatically runs build verification tests across any number of machines concurrently. This allows the automation of sophisticated tests such as deploying the server piece of a three-tiered application to a cluster of servers while at the same time deploying a client piece to any number of client machines and then letting the clients run tests on the cluster of servers.

This product supports automated promotion of a build. What is involved in a promotion is completely customizable (in some cases it is only creating a new baseline in the SCM, in other cases it may include repackaging of the build artifacts, and it may include deployment to a particular environment).

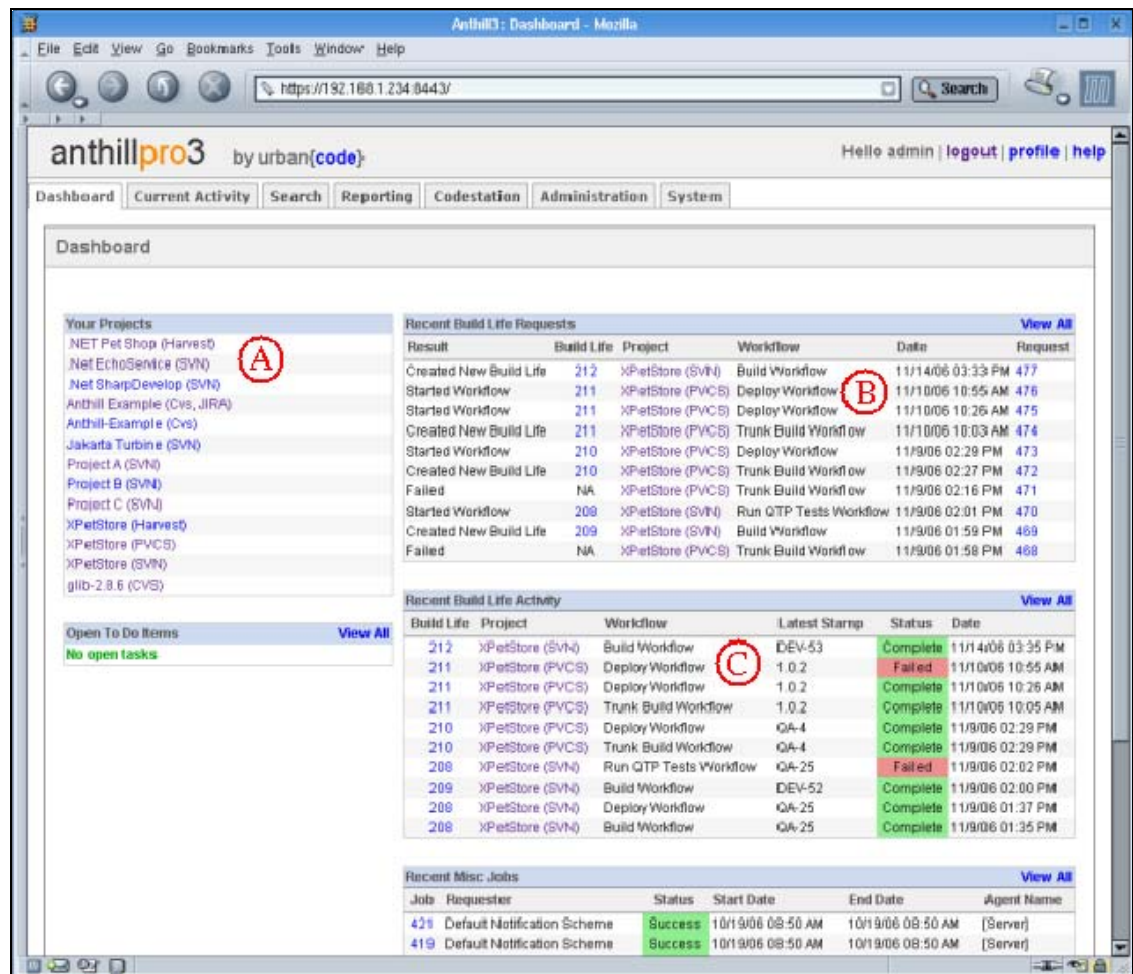
- **Living Builds.** No more having to configure multiple build "types" to support Continuous Integration builds, nightly builds, and release builds. The Living Build concept changes all of that. One Living Build can now be taken through various stages such as Continuous Integration, Nightly Build, Deployment, and Release.

AnthillPro introduces the BuildLife concept allowing multiple workflows to be applied to a single build.

- **Automated Deployments.** AnthillPro supports automated deployment of a build to a user-configured environment (for example: DEV, QA, STAGE, PROD). Gates can be set up between each environment requiring manual user intervention and providing an audit trail. Role-based security can be set up to control what roles (and thus what users) can deploy to which environment.
- **Support for Multiple Deployment Environments.** AnthillPro support user-specified environments such as DEV, QA, STAGE, PROD, etc. These environments may be used to partition work (for example BUILD\_FARM\_1 and BUILD\_FARM\_2) or for deployment, or for any other purpose. Each environment may be secured using role-based security.
- **Configurable Build Statuses.** AnthillPro supports user-specified list of statuses for BuildLives. The user-configured statuses comprise the "states" for a BuildLife (think: state diagram). Different statuses can be applied to a BuildLife as it is promoted or as it is deployed into an environment.
- **Configurable Build Jobs.** AnthillPro provides Customizable Build Steps. In addition to a Build Job Configuration, which makes assumptions about the steps that are going to be run and their order (check out code, run builders, run publishers, etc), the user can configure a Generic Job which allows any combination of steps in any order.
- **Role Based Security.** Role based security of this product ensures that users are able to access only the projects and environments to which they have approval. The authentication system can be configured to authenticate against LDAP or Active Directory. The authorization system roles can be obtained from LDAP or Active Directory as well.
- **Compliance Out-of-the-box.** Traceability, auditability, and role-based security. All deployed artifacts are guaranteed to be traceable to the exact source code used to produce them. Every build, promotion, deployment, and release is audited. Reports detailing the exact user that kicked off a build, or deployed a set of artifacts are included. Role based security means that some users can deploy a project to the DEV environment, another role of users may deploy the same project artifacts to the QA environment, and yet another role may deploy the artifacts to Production.
- **Scheduled Builds.** AnthillPro includes an embedded scheduler that may be configured with cron like expressions or to fire based on an interval. Single use schedule are also easily configurable so that you can set up a build to be kicked off at midnight or when you are long gone from the office.

On the picture below, you may see an example of the AnthillPro dashboard. It is divided into three parts marked with letters onto the picture. These parts represent the following elements:

- A. **List of projects the logged in user can access.** The security (authorization) system determines which projects show up in the list based on the roles associated with the currently logged in user.
- B. **Chronological list of requests received by AnthillPro3.** This list acts as a queue if the requests are not handled immediately.
- C. **Chronological list of workflows (builds, deployments, etc.)** executed by AnthillPro3. This list contains workflows executed across all projects, which the user can access. The user can drill-down on any workflow





## 7.6 IBM® Rational® Build Forge™ Enterprise Edition

<http://www-306.ibm.com/software/awdtools/buildforge/enterprise/>

This product provides reliable, high-performance builds throughout the software delivery lifecycle. The IBM Rational® Build Forge™ products help to automate complex processes and integrate diverse toolsets to compress development cycles, improve product quality and increase staff productivity. In addition, for decreasing time to delivery of the highest quality software, the Build Forge products provide visibility in key development trends to facilitate better informed management decisions.

<b>Build Process Management</b>	Excellent
<b>Web Interface</b>	Excellent
<b>Third-party tools integration</b>	Good
<b>Publishing</b>	Excellent
<b>Price</b>	Poor
<b>Total Evaluation</b>	★★★★

Build Forge solutions provide broad support for the major development languages, scripts and tools, regardless of the platform. If your business has made significant investments in an array of languages, platforms and other tools, you shouldn't have to change tools just to speed improvements. Build Forge products integrate into your current environment—your current scripts and tools can plug right in – so you get increased efficiency and automation without long implementation times.

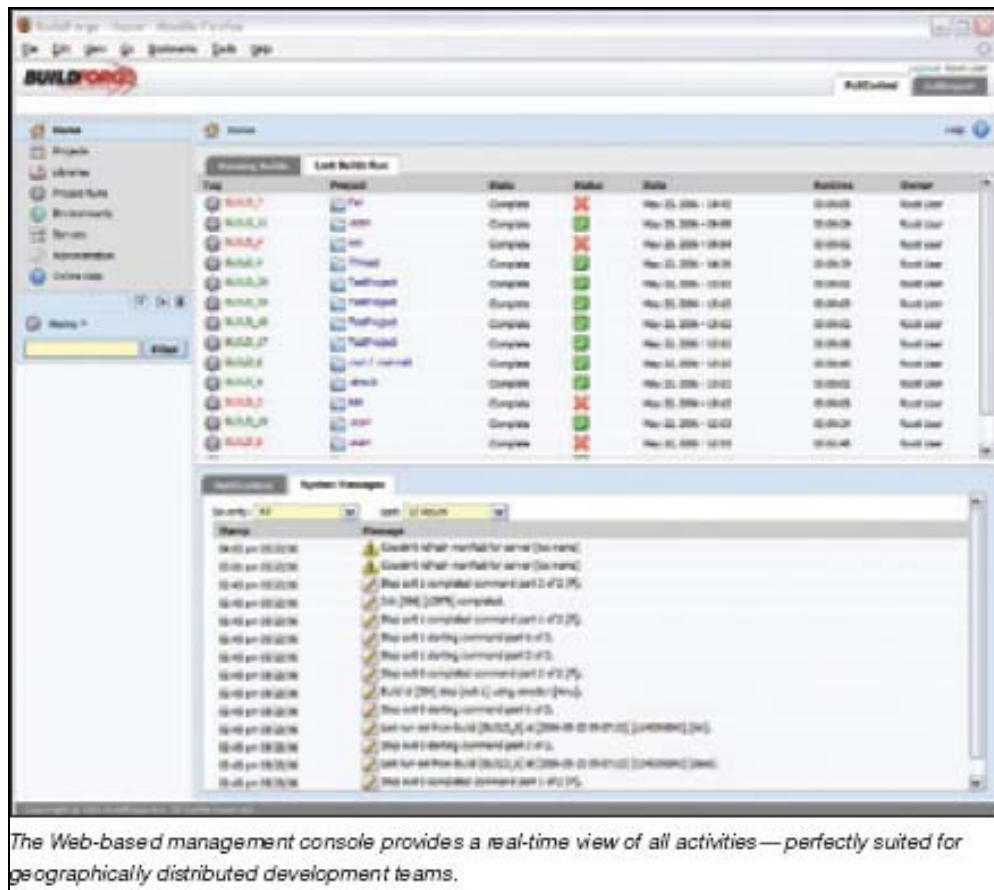
Build Forge allows to:

- **Eliminate the chaos of different build environments.** An open solution that is compatible with existing build scripts, batch files, development tools and other processes. There are no new proprietary scripting languages to learn and no throwaway work.

Out-of-the-box adaptors for creating automated handoffs between each phase of development, optimizing product delivery. Other software configuration management (SCM) and software development lifecycle tools are supported through the optional Build Forge Adaptor Toolkit offering.

Support for a variety of hardware platforms, including the Microsoft® Windows®, Apple Macintosh, Linux® and various UNIX® operating systems.

- **Keep distributed teams connected.** Build Forge products keep global teams in sync. The Webbased management console provides a consolidated, real-time view of each project so everyone stays on the same page. The Build Forge solutions can integrate with your existing tools so teams can share common processes, even if they are using different technologies. By linking the entire development process, handoffs are smooth and fast. Server pooling even allows distributed teams to use the same hardware resources. See the picture below for an example of the web-based management console.

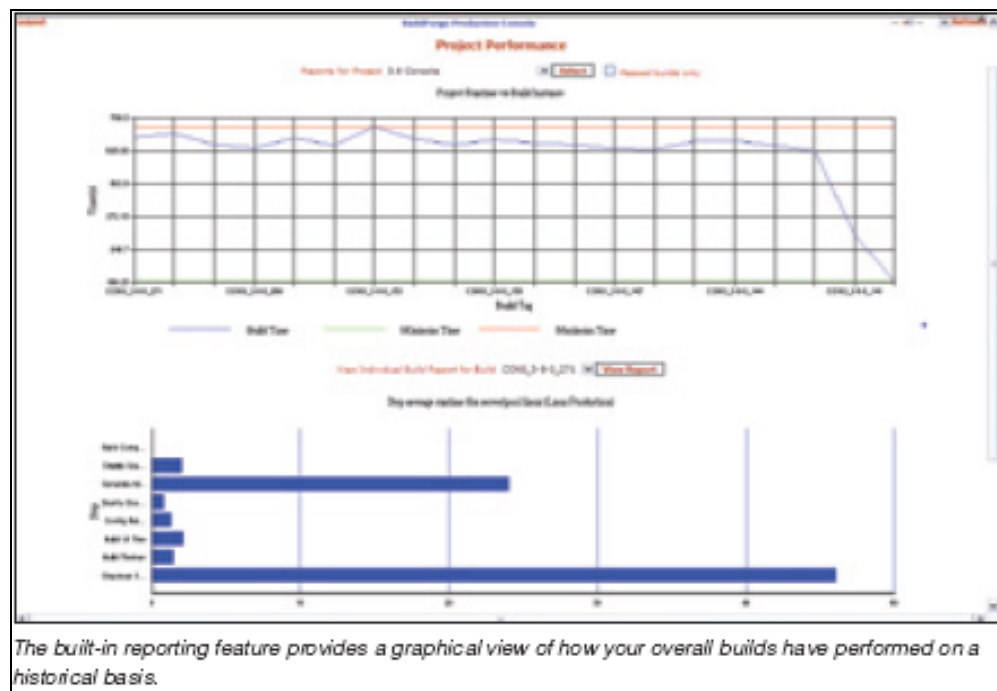


- **Promote consistency with automation and control from a single console.** Build Forge solutions provide a consistent management layer for your production build and release processes across all of your development platforms. Repetitive tasks are standardized and reused for multiple projects to help ensure process consistency and to reduce new-project setup time. Giving your team members a realtime view of all activities, the built-in, Web-based management console is perfectly suited for geographically distributed development teams that need access to the system virtually anytime, anywhere.
- **Simplify compliance management.** To simplify compliance of Build Forge solutions with existing environment:
  - Deliver a complete bill of materials that lists changes and the content of each build for accurate testing and problem resolution.
  - Correlate data from disparate source control, defect tracking and testing tools to give a one-stop view of development build artifacts.

- Enable role-based security to manage and enforce your compliance processes, with notifications and alerts to inform you if someone steps outside of your IT controls.
- Provide a documented audit trail of release contents and process changes for automated compliance management.
- **Speed problem resolution with rapid error detection and troubleshooting.** Build Forge products monitor and log build and release tasks step by step. When a problem occurs, an e-mail automatically notifies the responsible developer and links him or her directly to the location where the error occurred. Custom filters may be configured to detect language- or application-specific conditions. This allows defects and other build errors to be identified and resolved quickly.
- **Increase efficiency with flexible, automated scheduling.** Multiple scheduling options are offered that can align with your development approach. Builds and releases may be scheduled to execute at a specific date and time or when a source file changes. Or they can be scheduled to execute perpetually for continuous integration and agile development.
- **Shorten release cycles.** Because Build Forge products reduce build-related delays and help speed build cycles, development schedules can become achievable. Products are delivered to end users rapidly. Productivity of development engineers, build personnel and quality assurance (QA) staff increases because team members spend less time waiting and diagnosing, and more time doing their jobs—releasing quality products.
- **Protect mission-critical development knowledge.** The knowledge of how products and applications are built and delivered to your customers is a precious corporate asset. Typically, this knowledge resides in the minds of a few individuals who can leave your organization, complicate their replacements' backup and cross-training activities, and render your business vulnerable. Build Forge products capture, document and retain vital processes as they evolve, making it possible for teams to share their workload and execute builds and releases without requiring in-depth knowledge of the underlying processes.
- **Accelerate complex projects with concurrent processes.** The threading of Build Forge products enables discrete, independent processes that run concurrently for fast processing. Thread blocks allow you to increase performance for complex projects. Because they create simultaneously executing subprojects within the context of a single build, thread blocks are useful when you have two separate component builds that need to run on different platforms. Thread blocks also allow a build to be restarted at a failed step rather than entirely rerun when the build breaks partway through. Creating thread blocks also allows many tasks to be performed at the same time. Often, multiple development tasks can run independently up to a certain point, but they require a common library to be created before continuing. Build Forge solutions provide a global semaphore function, enabling separate thread blocks or even build projects to synchronize with one another.

- **Leverage team expertise with intuitive reporting and analysis.** These reports provide detailed:
  - Analysis of each project, including historical build performance and step run times.
  - Insight into the average run times for individual steps within your build so you can accurately measure how builds have been accelerated and identify areas for improvement.

See the picture below for an example of the Build Forge report:



- **Optimize your hardware environment with dynamic pooling and fault tolerance.** With Build Forge products, servers that were previously dedicated to a single project become highly productive shared server pools. The configurations and capabilities of each server in the pool are detected automatically so you can target all of your projects at a pool of servers to be distributed automatically across multiple machines. You can accelerate build times by dividing build tasks into smaller components. This same pooling capability allows fault tolerance. Build steps automatically redirect if a server or set of servers isn't available. By sharing and consolidating server resources, your operation can run at peak efficiency.
- **Transform idle servers into optimized build clusters.** Build Forge products offer a language- and script-independent build and release accelerator. The system accelerates builds by intelligently transforming idle servers into optimized build clusters. That can translate into a nearly immediate improvement in build-processing time.
- **Continually improve performance with detailed trend analysis.** Once you have optimized initial run times and the use of your servers, it's time to focus on day-to-day operation. Build Forge solutions keep a detailed historical record about each build—where steps were executed, how they performed and the servers that were used. A delta report shows details of previous builds and provides you with performance confidence indicators for how your build will run in the future.

- **Provide build capabilities to your extended development team.** Flexible user licenses allow developers to preview their local file changes in a staged build environment before committing the changes to source control. Developers can access project build results so they won't have to chase down build managers to determine what happened in the last build. Best of all, the preflight build feature means they can test the code they've been working on using a separate preconfigured build environment. It happens right from their desktop IDE, so they can check in with confidence. Capabilities include viewing build results, initiating new builds and viewing the status of currently running builds. Users cannot create new projects, delete projects or alter existing project step data. System-level permissions and access to Build Forge projects are honored, so users can only view and run the projects that they have been authorized to access. Scripts and processes stay intact. Additionally, build and configuration managers (CMs) grant the development team controlled access to build projects so they can reduce the amount of time spent manually responding to build requests and notifying developers of their status. Because the environment is controlled by role-based security, developers have access only to the projects that CMs authorize—keeping build scripts and projects intact.
- **Extend development capabilities with IDE plug-ins.** The Build Forge client leverages IDE plug-in technology. Because the client communicates directly with the Rational Build Forge console from within the IDE, network connectivity and user login authentication to the desired Build Forge console are required. IDE support is provided in the Build Forge Standard Edition and the Build Forge Enterprise Edition offerings. An Build Forge license is required for each user who accesses the Web console or the IDE plug-in for Eclipse, Application Developer or Microsoft Visual Studio .NET environments.
- **Create seamless links between third-party software environments.** Build Forge Standard Edition software includes out-of-the-box adaptors for ClearCase® and ClearQuest software. Adaptors allow users of source code managers, defect tracking managers and test managers to create seamless links between their SCM and build environments for increased efficiency and tracking of source code, defects and test changes. If you use third-party source code managers, defect tracking software or test managers, the Build Forge Adaptor Toolkit product allows you to create seamless links between these SCM and build environments for integrated tracking and control. Out-of-the-box adaptors are available for Rational ClearCase, Rational ClearQuest, CVS, Perforce SCM, Borland StarTeam, Microsoft Visual SourceSafe and Subversion software products. You can modify these adaptors or create your own for homegrown or other third-party software development tools.
- **Monitor source code changes.** Build Forge Adaptor applications provide continuous monitoring of the third-party source repository and execute builds automatically when a change occurs. The system directly monitors source changes and gathers metrics to provide detailed repository state tracking for each build. Using the Rational Build Forge management console, you can see where changes occurred—from submitted comments to actual file differences — on a per-build basis. Adaptors allow you to correlate source code changes, defects and tests with specific builds for a detailed understanding of the build components. Build Forge products capture key build statistics—including changes made, build time and build date—and store this information in a central location for quick access.

---

## 7.7 Comparison Table

We compared the build management products according to estimation criteria (described in section "[Estimation Criteria](#)") and created the comparison table for each group of comparison parameters (see [Appendix B](#)) to facilitate the analysis.

---

## Appendix A. BSE Comparison Table

Characteristics	Ant	SCons	NAnt	MSBuild
<b>Scripting options</b>				
Loops, Conditions	+	+	+	+
Tasks ordering	+	+	+	+
Parallel task execution	-	-	-	-
Script extensions	+	-	-	-
<b>Third-party tools collaboration</b>				
<b>SCM systems</b>				
CVS	+	+	+	-
Perforce	+	+	+	-
Subversion	+	+	+	-
Microsoft Visual SourceSafe	+	-	+	+
ClearCase	+	-	+	-
<b>QA tools</b>				
CheckStyle	+	-	-	-
PMD	+	-	-	-
JUnit (or NUnit in .NET)	+	-	+	-
Test NG	+	-	-	-

Characteristics	Ant	SCons	NAnt	MSBuild
Cover (or NCover in .NET)	+	-	+	-
JDoc (or NDoc in .NET)	+	-	+	-
<b>Compilers</b>				
Java	+	+	-	-
.NET	-	+	+	+
C/C++	-	+	-	-
<b>Database connectivity</b>	+	-	-	-
<b>Extensibility options</b>				
<b>Custom scripts inlining</b>	+	+	-	-
<b>Custom scripts call</b>	+	+	+	-
<b>Extension API</b>	+	+	+	+
<b>Community support</b>				
<b>Documentation, Articles</b>	+	+	+	+
<b>Books</b>	+	-	+	+
<b>IDE integrations</b>	+	-	-	+
<b>Script generators</b>	+	-	-	-

Characteristics	Ant	SCons	NAnt	MSBuild
GUI management applications	+	-	-	-
High-level script languages	+	+	-	-
Commercial support	-	-	-	-

## Appendix B. BMT Comparison Table

Characteristics	QuickBuild	TeamCity	CruiseControl	AnthillPro	Build Forge
<b>Build process management</b>					
<b>Dependency management</b>					
Dependency ranges	+	+	+	+	+
Version comparison	+	+	+	+	+
Conflict resolution	+	+	+	+	-
Snapshots	+	+	+	+	+
Profiles	+	+	+	+	+
Scope	+	+	+	+	+
Multiproject	+	+	+	+	+
<b>Distributed builds</b>	+	+	+	+	+
<b>Parallel builds</b>	+	+	+	+	+
<b>Builds Triggering</b>					
<b>Manually forced</b>	+	+	+	+	+
<b>SCM triggered</b>	-	+	+	+	+
<b>SCM poll based</b>	+	+	+	+	+
<b>Custom scheduling</b>	+	+	+	+	+

Characteristics	QuickBuild	TeamCity	CruiseControl	AnthillPro	Build Forge
Dynamic allocation of build resources	-	+	-	-	+
Build server inventory	-	-	-	-	+
Permissions and security model	+	+	+	+	+
<b>Web interface</b>					
View changesets	+	+	-	+	+
Add new projects	+	+	+	+	+
Clone projects	+	+	-	+	+
Delete projects	+	+	+	+	+
Modify projects	+	+	+	+	+
Kill builds	+	+	+	+	+
Pause builds	-	+	+	+	+
Access to build artifacts	+	+	+	+	+
Search in builds	+	-	-	+	+
Historic graphs	-	-	+	+	+
Self-updating web pages	+	+	+	+	+
Multiproject view	+	+	+	+	+
Add /remove agent machines	-	+	-	+	+

Characteristics	QuickBuild	TeamCity	CruiseControl	AnthillPro	Build Forge
<b>Third-party tools integration</b>					
<b>Build Scripting Engines</b>					
Ant	+	+	+	+	+
NAnt	-	+	+	+	+
MSBuild	-	+	-	-	+
Microsoft Visual Studio (2003, 2005)	-	+	-	+	+
<b>QA Tools</b>					
JUnit	+	+	+	+	+
NUnit	-	+	-	-	-
Clover	-	-	+	-	-
<b>IDE plug-ins</b>					
InelliJ IDEA	+	+	-	-	+
Eclipse	-	+	-	-	+
NetBeans	-	+	-	-	+
MS Visual Studio	-	+	-	-	+

Characteristics	QuickBuild	TeamCity	CruiseControl	AnthillPro	Build Forge
<b>Publishing</b>					
<b>Email</b>	+	+	+	+	+
<b>FTP</b>	-	-	+	-	+
<b>Jabber/XMPP</b>	+	+	+	+	+
<b>Custom client utility (Windows System Tray)</b>	-	+	+	-	+
<b>Price</b>					
<b>Price</b>	\$\$	\$	Free	\$\$	\$\$\$