Test Automation with Open Source Tools in an Agile SDLC Process (A Case Study)

Authors: Dan Lavender, Senior Architect, Comcast Media Center
Pete Dignan, President, ProtoTest

Abstract: Test automation, open source tools and agile methods are three important trends in software development. By integrating these three, a project team at Comcast was able to quickly build and deliver a critical application to its customers. The use of test automation enabled an agile software development lifecycle (SDLC) based largely on XP. The use of open sources tools for automation reduced the project cost while providing additional flexibility. The end result was on-time delivery with solid return on investment (ROI) for the business, and satisfied customers.

1 Introduction to the Case Study: Project Goals and Constraints

1.1 Introduction

Comcast is the country's largest provider of cable services, and is expanding its cable operations to deliver digital services, provide faster Internet service, and develop and deliver innovative programming. The *Channel Change Tool* (CCT) is an application that enables Comcast cable affiliates to manage aspects of their business that were previously managed at the Comcast Media Center.

The project to develop the CCT began late 2002. The charter was to build a new tool to allow cable systems the ability to change their channel lineups, as the old tool was being decommissioned. The application is far more complex than it may sound, involving as is does the association of specific satellites, transponders and frequencies. There are numerous business rules and dependencies involved. By the end of the project, CCT would consist of about 3000 classes, more than 200 of which were developed specifically for this system. The project team consisted of one technical lead (co-author Dan Lavender), two J2EE developers, one database administrator and one test engineer.

1.2 Project Goals

The CCT was to be a Web-based system to replace a client-server system that was being retired as a result of the merger between AT&T Broadband and Comcast. The Phase 1 deliverable, due to the field in just about 30 days, would use existing Microsoft Excel spreadsheets to capture data and Outlook to send spreadsheets in for processing via a Java application. This was really a placeholder until Phase 2, which would replace the use of spreadsheets and Outlook with a full Web-based J2EE system providing the same capabilities by March of 2003. The Phase 3 deliverable would enhance Phase 2 features and would provide automation and real-time data access from data sources, by May 2003. Finally, the Phase 4 deliverable was to enhance Phase 3 features to provide automation for additional cable system devices.

The business users of the system did not establish specific goals regarding software quality. Expectations may have been lowered by past experience with deliverables characterized by poor quality.

1.3 Constraints

The customers of CCT needed quick delivery of initial capabilities, but they weren't sure exactly what they wanted. The project methodology would have to be flexible and adaptable, since requirements were likely to evolve throughout the three phases.

As was the case with many IT projects during the down economy of late 2002-early 2003, the earmarked budget for CCT did not include any purchases of software or hardware (except for an Oracle database license). While it was possible to request software and equipment, the approval process for purchases of this magnitude exceeded the duration of Phase 1 and Phase 2. So the team found it necessary to work with available tools.

2 Agile Methods and Comcast's Modified eXtreme Programming Model

Based on the aggressive schedule and uncertain requirements, the CCT technical lead felt that Comcast's emerging agile development process held the best chance for success. He already had some positive experience with this process at AT&T Broadband in the pre-Comcast days.

The Comcast Media Center division had only used traditional waterfall SDLC processes in prior projects, and until this point management believed that agile methods were for "hobby shops or prototyping" only. So the technical lead had some work to do to convince management to take a risk and permit the use of an agile SDLC for the CCT project.

2.1 What are Agile Methods?

Over the course of the last 5 years, agile methods have emerged as a viable alternative to traditional software development lifecycle (SDLC) models. Agile (sometimes called "lightweight") methods can be seen as a reaction to the fact that so many software projects fail to meet expectations. The people who initiated the agile movement had the insight to observe that given the large number of failures, maybe it's not the people involved but the traditional methodologies themselves that are at fault. They questioned the underlying assumptions and beliefs of traditional approaches.

The agile methods movement, through a Web-based manifesto, claims to favor: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan.

According to Martin Fowler, "Agile methods are adaptive rather than predictive." Fowler's point is that while software development has long been compared with older engineering disciplines (like Civil Engineering), it is a poor analogy. Unlike building a bridge, where design costs are small compared with construction, in software, design costs often significantly overshadow construction costs (coding). So traditional methods resist changes to requirements. Fowler observes, "Everything else in software development depends on the requirements. If you cannot get stable requirements you cannot get a predictable plan." Agile methods recognize that predictability, in most environments, is not realistic. So the methodology must be structured to adapt quickly and easily to change.

The authors were unable to find current, hard data on the adoption rate of agile methods. Anecdotal evidence suggests that agile usage has made inroads into many large companies, including Comcast, Qwest and Cendant. Projects tend to be small (fewer than 20 team members), short in duration (mostly under a year) and are often viewed as experimental.

Agile methods include Alistair Cockburn's Crystal family of methodologies, Jim Highsmith's Adaptive Software Development, and Scrum. The most prominent example of agile methods is eXtreme Programming (XP).

2.2 What is XP?

XP concepts originated in the Smalltalk community in the late 1980s. Kent Beck and Ward Cunningham are generally credited with creating the methodology. In 1996, Beck was asked to review the progress of a software project at Chrysler. Beck recommended throwing out the work that had been done; the project then became the first prominent example of XP.

Some salient characteristics of XP are:

- User stories
- The customer is always available
- Write unit tests before coding
- Pairs programming
- Frequent small releases
- No functionality added before needed
- Refactoring
- Acceptance tests are run early & often

Much more detail on XP is available in books and online (see bibliography).

2.3 Comcast's Modified XP

Comcast has modified Kent Beck's eXtreme Programming model and named their version Collaborative Product Development (CPD). Under CPD, the CCT team would:

- Create a core operations concept document with a framework of user wishes
- Develop in iterations of one to three weeks
- Reset priorities before each iteration
- Conduct daily 15 - 30 minute "standup" meeting
- Develop tests in parallel with writing code
- Create only enough code to satisfy the tests
- Automate unit, regression and system testing
- Capture requirements and bugs in an online tracking system
- Provide a deliverable to the customer after each iteration
- Perform a post-mortem after each iteration

3 The Decision to use Test Automation

3.1 CCT Decides to Automate

Once committed to the use of the CPD process, the CCT team faced a new problem. In a January 2003 article for Datamation called "The 'Extreme Programming' Testing Trap," Linda G. Hayes described it perfectly:

> "Let's say that you are delivering an application in 10 iterations of two weeks each. In the first iteration, you have the same amount of code to write as you have to test, so all is well. The problem is that for the second iteration, and every iteration thereafter, testing has to cover all of the cumulative functionality as well

> as what has been added, so that by the tenth iteration there is at least 10X the amount of testing to be done. In reality there is probably 100X because of the interactions among functions, but no matter how you slice it you can't turn around the testing as fast as you can turn around the code."

The CCT team suspected that unless they were to automate nearly all testing, they couldn't keep up with the pace of development. In making the decision to automate, the team considered the following questions:

- Is the CPD process better supported by automated or manual testing? Should the CPD process be modified?
- Will our core components and subsystems be better on initial release and in subsequent releases using manual testing or automated testing?
- Can the project more *quickly* deliver quality releases with manual or with automated testing?
- Are the costs associated with automating testing justified by improved quality and greater speed?

The Comcast CPD recommends using fully automated build and test tools scheduled to run every 2 hours. However, most of the Comcast IT organization believed that this was unattainable. The CCT team committed to give it a try.

By using automation to set and restore the test environment to a known and repeatable state, and to execute a known and growing set of tests (comparing results to a known set of expected results) the project stood a fighting chance to stay on schedule. This applied equally to development activity (unit tests) and to the testing activity (system and regression tests).

If performed properly, test automation would significantly increase the breadth and depth of testing on CCT and would result in much higher quality software delivered to the customer. It would also significantly reduce the reintroduction of software bugs fixed in prior releases.

3.2 Automation – a Little Background

The authors' extremely informal and anecdotal observation, based on interaction with about 100 companies over the past five years, is that only about 10-20% of software tests in corporate America are automated as of mid-2003. There are many reasons for this, such as the historical under-appreciation of the economic value of effective testing, the complexity of the implementation effort and the relative immaturity of the available tools.

IDC sizes the market for distributed automated software quality (ASQ) tools at $610M in 2001, growing to about $900-950M this year. They project compound annual growth of 26% through 2006.

Of course there should be a theoretical limit to the percentage of tests that will *ever* be automated. Some tests don't yield to automation easily, and some code is too temporary or transient to justify the automation effort and cost. It is our view that this limit has not yet been approached, and significant additional growth in test automation is quite likely.

While Windows is the dominant target platform for automation tools, it is worth noting that IDC expects demand for Linux tools to pass the demand for *Unix* tools by 2006. This reflects the dramatic growth in the use of open source software. While it is clear that Linux has economic advantages over Unix when it comes to server platforms, it remains to be seen whether open source automation tools will make a similarly compelling case when compared with the leading commercial tools.

4 Decision to Use Open Source Tools

Since test automation appeared to be the only way for testing to keep up with CCT development, the next step was to choose which tools to use. But the CCT project didn't have budget or time to select commercial tools for automation, so the decision was made to use open source and other freely licensable technologies throughout the project (with the exception of the Oracle relational database management system).

According to the Open Source Initiative,

> "The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing. We in the open source community have learned that this rapid evolutionary process produces better software than the traditional closed model, in which only a very few programmers can see the source and everybody else must blindly use an opaque block of bits."

The most prominent example of open source software in widespread commercial use is the Linux operating system. IDC observed a 41% increase in worldwide Linux server revenue in 2002 (in an otherwise flat economy), and IBM says it did $1.5B in Linux revenue in 2002, with 4600+ Linux customers. Other examples of broadly accepted open source software include Apache Tomcat, MySQL, and JBoss.

Members of the CCT team had been actively involved in the open source movement. So the CCT team felt confident in making a decision to use open source tools wherever possible on the project. This Comcast division does not have an "approved technologies" list; therefore the team was free to choose.

5 Implementation

5.1 Getting Started

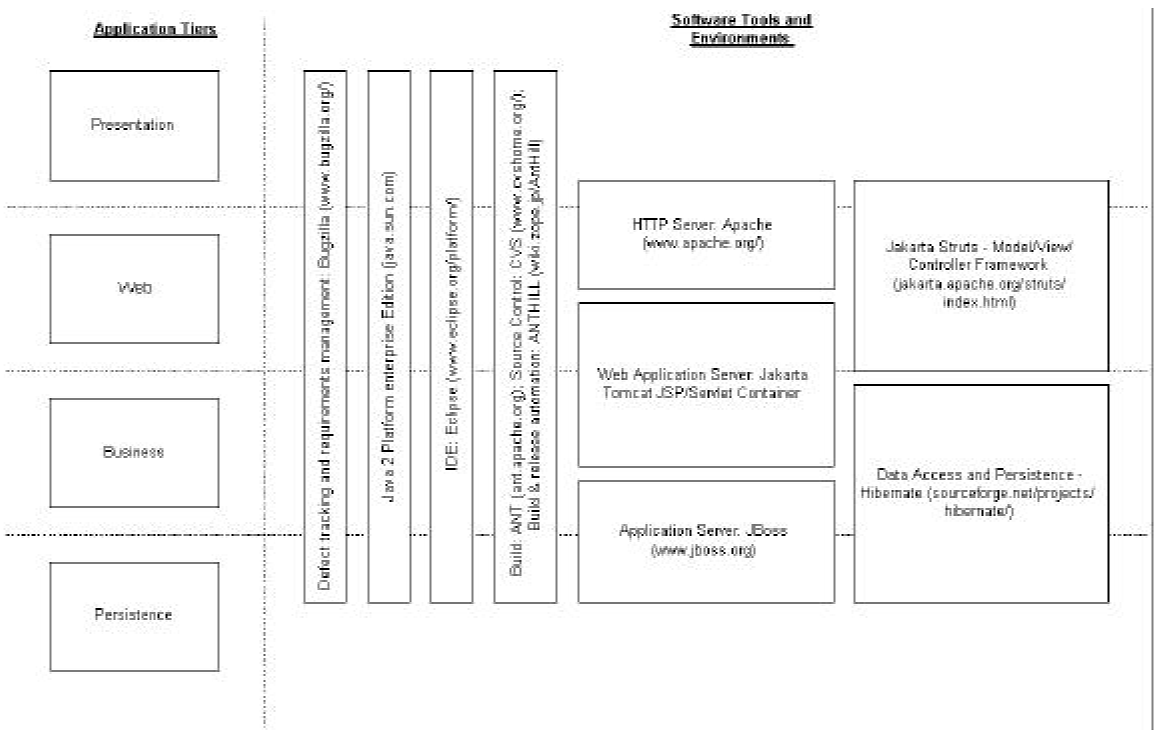
The CCT project began in November 2002 with about three weeks of exploration and planning. An initial set of high-level user stories were written to reflect system requirements, and the basic system architecture plus development and test strategies were outlined.

When work began in earnest on the first release, time was of the essence. Per the XP (and CPD) process, there was a 15-minute "standup" meeting each morning to establish goals for the day and to expose any obstacles. This phase was a proof-of-concept using Microsoft Outlook and Excel, with Java parsing. Given the short duration of the first phase, there was only one complete iteration of the code/test/fix process. The release was delivered as promised.

5.2 Architecture and Environment

The second release phase of the project was a critical step. It required creation of a Web-based J2EE replacement for the first release. This effort followed the same CPD process as the first phase, while adding all the tools required for the new implementation (not all the selected tools were needed for the first release).

The system architecture consists of four application tiers, known to the project as presentation, web, business and persistence (see diagram below). Users connect to the CCT application via their browser through an intranet connection. The application itself runs on an open source Apache web server using Tomcat (an open source container for Java Server Pages and Servlets) and JBoss (open source application server). The CCT database is held on a separate Sun server running Oracle 9i. Separate instances of Tomcat and Oracle were used for development, test, database administration (DBA) and production.


Application Tiers
Presentation
Web
Business
Persistence
Software Tools and Environments
Defect tracking and requirements management: Bugzilla (www.bugzilla.org/)
Java 2 Platform enterprise Edition (java.sun.com)
IDE: Eclipse (www.eclipse.org/platform/)
Build: ANT (ant.apache.org); Source Control: CVS (www.cvshome.org/); Build & release automation: ANTHILL (wiki.zope.jp/AntHill)
HTTP Server: Apache (www.apache.org/)
Web Application Server: Jakarta Tomcat JSP/Servlet Container
Application Server: JBoss (www.jboss.org)
Jakarta Struts - Model/View/ Controller Framework (jakarta.apache.org/struts/ index.html)
Data Access and Persistence - Hibernate (sourceforge.net/projects/ hibernate/)

The CCT development environment is based on Eclipse, an open source Integrated Development Environment (IDE), with Struts as the framework for implementing server-side Java technologies. Struts is an open source framework for building Web applications using J2EE. Struts uses a variation of the Model-View-Controller (MVC) architecture/design pattern.

CCT chose the open source tool CVS (Concurrent Versions System) for controlling code and documents associated with the project. A basic process for check-out, check-in and version management was established in the beginning. Code was managed with versions, branches and tags (more on that later) while documents were simply stored under CVS. Separate directories were created for development code and code under test. A related tool called TortoiseCVS enabled users to work with files under CVS version control directly from Windows Explorer.

Bugzilla gave the project a convenient way to track and manage all known defects, to share defect information with interested parties, and to track system requirements. (By August 2003, 150 defects had been logged.) Bugzilla is an open source defect tracking system originally written by Terry

Weissman in a programming language called TCL (and later ported to Perl) for use at Netscape Communications.

5.3 *How the Testing Worked*

Once the second phase of the project got underway, the team detailed the user stories into use cases, sequenced them, and created designs under the new architecture. Unit test stubs were created for each story (several hundred new unit tests would be created before the project's conclusion). Since unit testing is a key aspect of XP and Comcast's own CPD, and because the project was a Java development effort, JUnit was an obvious choice for this work. JUnit, as the name suggests, is an open source framework for creating unit tests for Java classes. Erich Gamma and Kent Beck were the original authors of JUnit. Since Kent Beck is known as the 'father of XP,' one can begin to see that the connection between agile methods and open source has deep roots. CCT developers used JUnit extensively.

The CCT team also had a need to create unit tests to verify that application data was being written and modified correctly by the code. DbUnit is a JUnit extension that has the ability, among other things, to put the database into a known state between test runs. DbUnit can export and import database content to and from XML datasets. DbUnit also provides the capability, through assertions, to ascertain that database contents match the expected values. CCT developers and testers used this tool to set the database objects to known states prior to running unit, system, and regression tests.

CCT developers used Cactus for unit-testing JSPs and Servlets at the Web tier. Cactus, which is compatible with the JUnit framework, is an Apache Jakarta open source project. It was used to validate server-side Java code allowing straightforward white-box testing.

The team made extensive use of Ant, an open source Java-based build tool. Ant's original author, James Duncan Davidson, has said the name is an acronym for "Another Neat Tool." Ant configuration files are XML-based and call out a target tree (Ant supports dependencies – key to its use in testing) where various tasks get executed. On the CCT project, ANT was used as a build tool. But because Ant can be extended using Java classes, it was also used by the CCT team to run unit, system, and regression tests, and to set up the application and database environments for running the tests. For a list of the tasks CCT implemented, see Appendix 1.

Anthill is an open source tool that acts like a test harness or test driver. On the CCT project, the developers used it to automate unit tests into a sort of integration test suite. Once every 1-2 hours Anthill was executed via a CRON job. It pulled all current code from CVS, compiled and built the code, deployed to the test environment, and then launched a suite of tests via Ant.

If new changes were checked in and the automated unit tests failed, the developer was notified immediately. One especially useful aspect of this was the tendency for these automated unit tests to uncover defects that were not apparent prior to integration. By catching these quickly, the bugs were not allowed to propagate and interfere with the work of other developers. Examples: The database schema changed, resulting in broken hibernate DAO objects; a controller object (a servlet) broke because a view object (a JSP) was removed from CVS. Catching these errors quickly helped limit the rework that occurs when the other developers access broken code. The team was are able to confidently change code, unit test, and regression test at the unit level knowing that even if errors arise, they will be quickly identified and fixed.

One other testing footnote - the DBA environment (in Tomcat and on the database server) was also used to specifically test the *process* for promotion from test to production. It served as a simulated production environment for this purpose.

5.4 *Open Source Doesn't Cover Every Base*

While the project was able to make extensive use of open source tools, not everything went exactly as planned. For instance: Canoo WebTest is an open source tool for automated testing of web applications. The CCT project's dedicated tester began using Canoo WebTest for system testing, to simulate keystrokes in the browser at the presentation tier. However, it quickly became apparent that Canoo had problems recognizing and handling JavaScript. Since many of the screen-based objects in the application were written using JavaScript, or kicked off processes that employed JavaScript, this was a serious obstacle. During the time of the CCT project, there was a plug-in available for Canoo that was supposed to solve this problem, but the team was not able to make it work. While this issues was expected to be resolved fairly soon, the project schedule demanded a workaround.

CCT turned to a tool called TestSmith, which is a commercial product in beta and as such is freely downloadable through December 31, 2003. TestSmith is a record and playback engine that recognizes applets and embedded objects, including JavaScript. TestSmith became the workaround for tests that Canoo could not handle.

The project team also found it necessary to use two other low-cost non-open-source tools for testing.

Scooter Software's Beyond Compare is a file comparison utility with features that made it easy to create many similar, but not identical, end-to-end scripts for execution in TestSmith. More than 100 system tests (beyond the unit tests automated in Ant) were created this way.

WAPT 2.0 is a load- and stress-testing tool from Novosoft. The CCT team felt that when compared with open source performance testing tools, WAPT supplied capabilities that justified its modest price tag.

5.5 *Some Bumps in the Road*

While using an agile process with extensive test automation seems to have distinct advantages in terms of defect prevention and detection, the CCT project wasn't without its challenges.

Just after release two hit production, the team began hearing from users that *intermittently* they couldn't access application data. The team worked together to investigate the problem. The application supports field-level validation of data, business rules checks prior to inserting data into the database, and further validation against complex rules once inside the database. None of these steps appeared to play a role in the problem. Eventually, in a discussion with network operations staff, the team learned that there was a firewall between the production web server and the database server, something not present in the test environment. The firewall was configured to terminate connections that remained idle for a certain period of time. The CCT application used static connections to the database, which if periodically active, presented no problem. But if the application went unused long enough to generate a timeout on the firewall, users could no longer get to the data. A one-line change in the firewall configuration fixed the problem. The team noted the importance of *fully* replicating the production environment in test whenever possible.

The development leading up to the third release focused on enhancements to functionality and real-time data connectivity. During these iterations, the team's tester concentrated on adding all new features to the system-level regression tests.

Yet it was not all smooth sailing in this phase of the project either. The third release broke several key capabilities of the system in production, including users' ability to sort on certain fields and something called the "self-registration" process. This happened because the project team was maintaining two versions some parts of the code in CVS, one in the "head" path and one in a branch, to support the differing needs of various customers of the system. Although the soon-to-be released version of code was thoroughly tested, the *wrong* (untested) version was released into production. As soon as users encountered the problems, a "Yikes!" message was generated by the application and logged to the server, and the CCT team was automatically paged. The team reviewed the CM process and decided to stop using the branching feature of CVS, and to use the tagging feature instead.

Of course these kinds of difficulties are common in all development projects and under all methodologies. But the CCT team felt its agile approach held several benefits in these cases. They were able to rapidly uncover the problems due to frequent delivery of functionality to the users. Taking advantage of the daily "stand-up" meetings, they made quick recoveries and speedy repairs to the *processes* that led to the defects.

Release four centered on the addition of encryption and decryption devices to the system. During these iterations, the team added significant new functionality to the application. This included adding a new set of features automating the management of the devices that receive and decrypt the cable signals from the satellites. The developers refactored a couple of the action servlets, improving processing and increasing the agility of the application. This final release was delivered in July 2003.

6 Outcomes, Lessons Learned and Conclusions

6.1 Outcomes

The CCT project is viewed as a success by Comcast management and customers alike. The team successfully delivered all the promised features, on schedule and within budget for all phases. The decisions to use the agile methodology, to automate tests, and to use open source tools were all validated by the results.

6.2 Lessons Learned

- When using an agile SDLC, certain "traditional" practices that support software quality, like thorough CM and careful replication of the production environment in test, are still important and useful.
- Test automation in a fast-moving project can help to uncover defects that almost certainly would have been missed with a purely manual testing approach.
- It is important to run with recent releases of open source tools, because feature sets can change quickly in the open source community and usually only the current release is fully supported. Staying current requires a pro-active approach by the project team.
- As with commercial tools, take time to evaluate open source tool features to determine if they fully support project needs. If time permits, you can take advantage of the open source capability to extend a tool to meet specific needs.

- The availability of bug fixes and support for the open source tools was often superior to our experience with commercial tools.
- The CCT team's greatest difficulty was with the funding and procurement processes. In large companies, these are geared toward waterfall models where timeframes are significantly longer. In the waterfall model delays are more easily hidden due to the length of the project. With agile projects, it is critical that computers, network connectivity, user accounts, etc. are procured and set up quickly or the project deliverables will slip.

6.3 Conclusions

Some people argue that agile methods use many of the concepts of the Waterfall SDLC model and so aren't truly "new." The experience of the CCT team suggests that in the case of the CPD (modified XP), there are substantive differences that have positive implications for software quality. User stories enforced requirements capture more effectively than the techniques we've seen on many larger projects. Pairs programming led to defect avoidance. Writing unit tests before coding meant unit testing actually was done, and the code was written to satisfy only the tests, reducing the amount of code that was written. Test automation supported frequent, comprehensive testing. Shorter delivery cycles for 'finished' code forced a high degree of accountability.

For the CCT project at Comcast, test automation was a key enabler for successful implementation of an agile process, because it enabled frequent, thorough testing even as the development proceeded quickly. The selected open source tools largely supported the team's test automation goals. Combinations of open source tools provided features, stability and scalability that rivaled what is available in commercial products. The integration of test automation, open source tools and an agile SDLC methodology resulted in an acceptable return on investment (ROI) for the business. Hard-dollar annual savings in the Comcast Media Center, plus annual savings at cable system "headends" plus, estimated goodwill benefits with customers were approximately three times the total cost of the project.

Appendix I – CCT Ant Tasks

Task Name	**Task Description**
Clean	Removes build artifacts
compile-common	Compiles common module
compile-ejb	Compiles ejb module
compile-web	Compiles web module
copy-resources	Copies properties and .xml files from source directory
copy-struts	Copies Struts library and TLD files
copy-test-jars	Copies test-related JAR files to WEB-INF/lib
copy-web-files	Copies static files
db-create	Creates database for ${database.type}
db-drop	Drops database tables
db-export	Exports database using DBUnit
db-export-prd	Exports production database using DBUnit
db-init	Creates database tables
db-load	Loads database from exported DBUnit file
db-load-prd	Loads database from exported DBUnit file
deleteUsers	Runs Canoo WebTests in Tomcat
deploy	Unwars into the servlet container's deployment directory
deploy-war	Deploys to local Tomcat
deploy-web	Deploys only web classes to servlet container's deploy directory
dist-doc	Creates zip and tar.gz of docs for distribution
doc	Runs javadoc, todo, checkstyle and pmd tasks
ejbdoclet	Generates Persistence and form classes
gen-test-reports	Generates test reports
gen-tests-common	Generates tests for common module
gen-tests-ejb	Generates tests for EJB module
gen-tests-web	Generates tests for web module
init	Inits target directory
java2html	Creates HTML pages of code for online viewing
javadoc	Generates JavaDoc API docs
new	Creates a new project with the specified name
package-ejb	Packages EJB JAR
package-web	Packages WAR
ping-tomcat	Pings Tomcat to make sure it's running
pmd	Locates unused imports, unused variables, etc.
setup-tomcat	Copies Oracle JDBC driver and context.xml to Tomcat
stage-web	Calls other targets to gather static resources
test-all	Runs all tests for common, EJB and web
test-all-running	Runs all tests for common, ejb and web
test-cactus	Runs Cactus tests in Tomcat, starts/stops server
test-canoo	Runs Canoo WebTests in Tomcat
test-common	Tests common module
test-dbexport	Exports database using DBUnit
test-dbload	Loads database from exported DBUnit file
test-ejb	Tests EJB module
test-jsp	Runs Canoo WebTests using Cactus
test-web	Tests web module
undeploy	Undeploys WAR file to servlet container's deployment dir
webdoclet	Generates web and Struts descriptors

Appendix II - Bibliography

Regarding Agile Methods:

1. http://www.martinfowler.com/articles/newMethodology.html
2. http://alistair.cockburn.us/
3. http://www.adaptivesd.com/
4. http://www.controlchaos.com/
5. http://agilemanifesto.org/
6. http://www.extremeprogramming.org/
7. The 'Extreme Programming' Testing Trap; January 29, 2003; Datamation; Linda G. Hayes

Regarding Test Automation:

1. IDC: Worldwide Distributed Automated Software Quality Tools Forecast and Analysis, 2002-2006

Regarding Open Source Tools:

1. http://www.apache.org
2. http://jakarta.apache.org/tomcat
3. http://www.jboss.org
4. http://www.eclipse.org
5. http://jakarta.apache.org/struts/
6. http://www.cvshome.org/
7. http://www.tortoisecvs.org/
8. http://www.junit.org/index.htm
9. http://dbunit.sourceforge.net/
10. http://jakarta.apache.org/cactus/index.html
11. http://ant.apache.org
12. http://www.urbancode.com/projects/anthill/default.jsp
13. http://webtest.canoo.com/webtest/
14. http://qualityforge.com/testsmith/
15. http://www.scootersoftware.com
16. http://www.loadtestingtool.com/

Additional examples of open source testing tools:

17. Test management tool Software Testing Automation Framework (STAF): http://staf.sourceforge.net/index.php
18. Functional testing tool Anteater: http://aft.sourceforge.net/
19. Performance tools such as OpenSTA (http://portal.opensta.org/index.php) and Push-to-Test TestMaker (http://www.pushtotest.com/ptt/)