Teaming Up for Software Development

Luís Soares

Departamento de Informática
Universidade do Minho

Engenharia de Aplicações

In the end of the session the attendee should be able to:

- Identify several super-sets of tools used in software development labs;
- Distinguish several stages in software development.

Outline

- Common software development pitfalls.
- Concepts and Tools
  - Dependency management and build frameworks.
  - Tracking changes.
  - Tracking issues.
  - Software quality control.
- A walk in the lab.
- Few final remarks.
- Additional links.

Common software development pitfalls

- Documentation almost nonexistent.
- No standards.
- No recorded history through project's time-line.
- Integration is not frequent.
- *Ad hoc* testing.
- Heterogeneous tool-set.

Dependency Management and Build Frameworks

Dependency Management Definition

Keeping track of third party libraries required to build/test/execute the project.

Build Tools Definition

Tool that relies on a set of rules to build the source code.

- Doing this manually is error prone and time consuming.
- Automatic dependency management removes the burden from developers of having to synchronize libraries every now and then among themselves.
- Versions are handled gracefully.
- Configure once, build anytime.

GNU Make

- Programming language agnostic;
- Dependencies statically listed and provided by the user;
- Wide-spread usage (build, install, document).
- *Makefile*: "Make uses the makefile to figure out which target files ought to be brought up to date, and then determines which of them actually need to be updated."

Apache Ant

- Requires Java Virtual Machine;
- Mostly used to build java source code;
- Dependencies statically listed and provided by the user;
- Wide-spread usage among java programmers (build, install, deploy, document);
- *build.xml*: "Ant's build files are written in XML. Each build file contains one project and at least one (default) target."

Apache Maven (v2)

- Requires Java Virtual Machine;
- Mostly used to build java source code;
- Dependencies are listed by the user and downloaded automatically;
- Wide-spread usage among java programmers;
- Much more: *"Maven is a software project management and comprehension tool."*
- *pom.xml*: "A Project Object Model or POM is the fundamental unit of work in Maven. It is an xml file that contains information about the project and configuration details used by Maven to build the project."

Tracking Changes (Version Control)

Version Control Software Definition

Software used to control multiple revisions of the same information unit.

Version Control Benefits

- Development history is preserved.
- Standard integration process for the development team.
- Easy to rollback in case a serious mistake is introduced.
- Automatic generation of changesets between versions/releases.
- Changes are always associated with a given developer.

Version Control Software Systems

- Centralized:
  - CVS - Concurrent Versions System
  - Subversion - CVS Done Right!
- Distributed:
  - Mercurial - Efficient (Mozilla, Xine, OpenSolaris)
  - Git - Tracks content (Linux Kernel)
  - Bazaar - Tracks Files (Launchpad, Ubuntu)

Tracking Issues

Issue Tracker Definition

Software used to report, maintain and describe issues and work done in time.

Issue Tracking Benefits

- Document changes and their motivation.
- Track issues and their impact in development life-cycle.
- Mean to get feedback from developers/users on the work done.
- Eases new developers/users integration by providing a documented record of the project's past.
- When tightly integrated with the *Revision Control System*, becomes a powerful bug tracking tool.

Bug/Issue Tracking Applications

- trac
- Jira
- roundup
- Bugzilla
- Launchpad/SourceForge/...

Software Quality Control

Software Quality Control

Controlling the quality of the software, resorting to tests (unit, integration and system tests), simulation and continuous integration.

- Test-driven Development: Test early and often. Start from an interface, a mock implementation and a unit test. End up with a full featured implementation passing the unit tests flawlessly.
- Revision Control commits should be small. Do not be afraid to commit often and small changes.
- Build early and often. If a commit breaks a build or fails a unit test, the problem gets detected early in the development cycle.
- Ideally, each commit should trigger new build.

Continuous Integration Definition

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.", by Martin Fowler.

Frameworks

- hudson
- cruise control
- continuum
- bamboo

A walk in the lab.
(Dry-run)

Setting up a development Environment using Maven2

1. mvn archetype:create -DgroupId=ea -DartifactId=calculator
2. mvn compile [test clean package]

Adding revision control

1. bzr init .
2. vim .bzrignore
3. bzr add .
4. bzr commit -m "Initial import."

Issue Tracking with trac

1. sudo apt-get install trac
2. mkdir $HOME/trac.env
3. trac-admin $HOME/trac.env/calculator initenv
4. trac-admin /home/ea/trac.env/calculator/ permission add anonymous TRAC_ADMIN
5. vi /var/www/trac.env/calculator/conf/trac.ini
6. tracd –port 8000 /home/ea/trac.env/calculator
7. create milestones (release 1.0) and tickets

Continuous Integration with Hudson

1. download it from https://hudson.dev.java.net/
2. java -jar hudson
3. create build jobs

Make a change, compile and commit

1. vim src/main/java/ea/SumService.java
2. vim src/main/java/ea/SumServiceImpl.java
3. vim src/test/java/ea/TestSumService.java
4. mvn clean compile test
5. bzr commit -m "Added sum service."

Implement SumService

1. vim src/main/java/ea/SumServiceImpl.java
2. mvn clean compile test
3. bzr commit -m "Implemented SumService"

Teaming Up

1. Create ticket for "John Doe": Implement MultiplyService
2. bzr branch /home/ea/calculator calculator.multiply
3. ... Implement tests and MultiplyService ...
4. bzr merge calculator.multiply
5. integrate/build
6. close ticket
7. milestone completed

Release

1. All milestones completed
2. Remove -SNAPSHOT tag
3. branch new trunk

Conclusions

Few final remarks.

- Teaming up requires third party tools that ease development and integration.
- Investing in some tools may pose a steep learning curve at first, but it pays off in the long run.
- You are not alone! Act as a team! Use standards and conventions that everybody acknowledges (common ground).
- Test, commit and build early and often.
- Document as much as possible. Documentation avoids issue rebounding.

Links

- Trac: http://trac.edgewall.org
- Hudson: https://hudson.dev.java.net/
- Maven: http://maven.apache.org/
- Bazaar: http://bazaar-vcs.org/
- Google: http://www.google.com ;)