Software Development Tools

COMP220/COMP285

Sebastian Coope

More on Automated Testing

and

Continuous Integration

These slides are mainly based on "*Java Tools for Extreme Programming*" – R.Hightower & N.Lesiecki. Wiley, 2002

Automated Testing

- ***Testing software continuously*** validates that
  - the software *works* and
  - meets the *customer's requirements*
- ***Automating the tests*** ensures that
  - testing will in fact be *continuous*
- ***Without testing***, a team is just ***guessing*** that its software meets those requirements.

Need for automation

- Humans make mistakes
- Humans sometimes don’t care!
- Manual testing is slow, hard to gather statistics
- Automated testing can be done at all hours
- Automated testing is fast, 10,000 tests/second
- Regression testing builds up with the project size

Tests and refactoring

- ***Refactoring*** is changing existing code for simplicity, clarity and/or feature addition.
  - *cannot be accomplished without tests*.
- Even the most stable or difficult-to-change projects *require occasional modification*.
- That is where ***automated testing*** comes in.

Tests and refactoring

- *Comprehensive tests* (running frequently)
  - verify how the system should work,
  - allow the underlying behaviour to change freely.
- *Any problems* introduced during a change are
  - *automatically caught* by the tests.
- *With testing,* programmers
  - *refactor with confidence*,
  - the code works, and
  - the tests prove it

Types of Automated Testing

1. Unit Testing

- *testing of a **unit of a code***
  - everything that could possibly break
- usually *exercises all the **methods in public interface*** of a class
- verifies that
  - the unit of code ***behaves as expected***
- with this verification
  - the *public interface **gains meaning***

Types of Automated Testing

Unit Testing

- part of the cycle of *everyday coding*
- *writing tests **before** coding*
- *test as a **guide*** to assist in implementation

Types of Automated Testing

Unit Testing

- tests grouped into *test suites* run *multiple times per day*
- all tests *should always pass*
- results in *high quality system*
- leads to *clean architecture*

Types of Automated Testing

Unit Tests:

JUnit tool is

- lightweight *unit testing framework*
- for testing **Java code**
- implemented itself also in **Java** by
  - *Erich Gamma* and *Kent Beck*

Types of Automated Testing

2. Integration Tests

- ***Unit tests*** are deliberately supposed to be
  - ***isolated*** and
  - as ***independent*** as possible,
- ***Integration testing*** ensures that
  - *all the code* ***cooperates,*** and
  - *differences* between expectation and reality are precisely ***localized***.

Types of Automated Testing

Integration Tests: Cactus tool

- Testing of ***Web applications*** *with multiple tiers* becomes significantly more difficult.
- More (and more complex) testing tools are needed.
- **Cactus** is such a tool ***extending*** **JUnit**

  - to support ***testing server-side code*** (specific classes and methods).

Types of Automated Testing

3. Acceptance/Functional Tests

Functional testing ensures that

- the ***whole system*** behaves as expected
- called also ***acceptance testing*** to verify for the customer that the system is complete
- **For example**, an e-commerce *web site* is not done until it
  - can log in users,
  - display products, and
  - allow online ordering

Types of Automated Testing

Acceptance/Functional Tests: HttpUnit

- There exists ***no*** *universal acceptance testing tool* to be used for arbitrary applications
- But we have **HttpUnit** as a *specialised* **Acceptance/Functional** *testing tool* for
  - programmatic ***calls to Web resources***
  - and ***inspection of the responses***

Types of Automated Testing

Acceptance/Functional Tests:

Cactus *vs.* HttpUnit

- Both these tools *test Web application* components
- **Cactus** is more ***unit-oriented*** – to exercise the *behaviour of* ***specific classes and methods***
- **HttpUnit** is designed to exercise ***requests to specific resources on a server***

Cactus

- Works with Java servlets
- Cactus will
  - Create JVM for client (prepares the request)
  - Create JVM for server (handles request)
  - Run test across both server and client
- Essentially tests across a Java servlet interface
- Limitation… Servlets only… very coupled with Java technology

HttpUnit

- Makes requests to external website
- Relies on Java to make requests
- External website can be written using anything you like
  - PHP, ASP.NET, Perl, Ruby on rails

HttpUnit example…

- WebConversation wc = new WebConversation();
- WebResponse resp = wc.getResponse("http://www.google.com/");
- WebLink link = resp.getLinkWith("About Google");
- link.click();
- WebResponse resp2 = wc.getCurrentPage();

Web testing frameworks

- HttpUnit
  - Low level simplistic web API
  - Poor Javascript support
- HtmlUnit
  - A lot better Javascript support
  - Better support at the document level
- JWebUnit
  - Essentially a wrapper for
    - HtmlUnit
    - Selenium (Browser based test frame-work)

Types of Automated Testing

4. Performance Tests:

JUnitPerf and **JMeter**

- the most functional system in the world won't be useful if end users give up on the software because of poor ***performance***

Types of Automated Testing

Performance Tests:

JUnitPerf and JMeter

- **JUnitPerf** does
  - ***unit*** *performance testing*
  - it *decorates* existing **JUnit** tests so that they fail if *running times* exceed expectations
  - Is for Java testing, not web testing
  - *supports refactoring* by verifying that performance-critical code remains within expected boundaries

Types of Automated Testing

Performance Tests:

JUnitPerf and JMeter

- **JMeter** provides
  - ***functional*** *performance* testing—*times to requests sent to a remote server* like:

    "the Web server will maintain a three-second response time to requests with a 150 users simultaneous load"

Jmeter is

Site agnostic … (code can be PHP etc.)

Does NOT run Javascript

Continuous Integration

Continuous Integration:

- *building* a complete copy of the system so far (and running its full test suite)
  - *several times per day*
- to be sure that the *current version* of the system *is ready* to walk out the door *at any moment*
- should be relatively *automatic*, or no one will ever do it

Continuous Integration

Continuous Integration:

- allows the customer and team to *see the progress*,
- *integration bugs are reduced*, and
- the *tests run frequently*
- reduces integration pain:

  makes sure the incompatible "dance partners" meet within hours or minutes

How to make it ***automatic*** ?

Continuous Integration & **Ant**

Ant ***will help!***

- Unlike many other practices of **XP**, ***continuous integration*** is mainly a ***technical problem***
- These lectures will cover **Ant**, the *emerging standard* for build automation in **Java**
- **Ant** allows to *invoke tests*
- **Ant** is *cross-platform* and easy to *extend* and *modify*

Continuous Integration & **Ant**

Ant performs all the basic tasks of a build tool:

- compilation,
- archiving,
- classpath management,
- supports testing
- FTP, etc.

All of this in an automatic way!

Continuous Integration & **Ant**

With a ***single*** **Ant** ***command***,
a **Java application** can be

- *built*,
- *customized* to its intended environment,
- *tested*, and
- *deployed* to a remote server

Testing and Continuous Integration: **Software Tools**

- Amongst all these testing and integration tools, we will devote the most part of our lectures to **Ant,** which can also invoke **JUnit**.

- We will also consider **Eclipse** – Integrated Development Environment (**IDE**) mainly for **Java** programs which can also invoke both **JUnit** and **Ant.**