COMP314: Virtuous Cycle

Cycles of activity and software projects

Automation and tools in a software company

Testing as part of feedback cycles
Programming Cycle

- Specification
- Program

Code Construction

Inspection
Programming Cycle - single bug

~8 hours

Selected Bug → Fix Bug → Shared Code

Continuous Integration
Programming Cycle - 10s lines of code (personal task)

- Selected Task
- Personal copy of code
- Compilation
  - Unit tests
- Type code

~10s minutes
Multiple levels of cycles

Selected Bug ➔ Fix Bug ➔ Shared Code ➔ Central Checking ➔ Quiet Point

Selected Task ➔ Type code ➔ Personal copy of code ➔ Compilation ➔ Unit tests
A Virtuous Cycle

- High bandwidth – get as much feedback as possible
- Low Latency – get feedback as soon as possible
Examples of interesting cycles

- Innovation cycle - development of new products and ideas
- Science - selection of ideas and deliberate testing of them
A Very Short History of Software Engineering

- **October 1968**
  NATO Workshop term “Software Engineering” coined

- **1960’s - now**
  Lots of fumbling around in the dark

- **1990s - now**
  “Extreme programming” or “Agile programming” – keep it light, simple, fast, only do what is needed
NetValue Development

- Tools that are used as part of the development cycle
  - cvscheck
  - compilation
  - style checking
  - testing
  - javadocs
  - documentation
  - jumble
  - quality of unit testing
NetValue Development

Continuous Integration

- cvscheck - Source Code Control and Build
  - Shared
  - Centralized
  - Automatic
The system currently checks the code out of version control, compiles the src and test hierarchies separately, runs the unit tests under various jvms, generates javadocs for the src and test hierarchies. It performs additional checking that all test classes are linked into the root tester, and that each class has at least rudimentary documentation.

Current Status: Problems on other platforms
Latest Check: Level 5 started at 2008-07-20 14:30

- No changes in CVS
- cartesian/src
  - Clean compile (all output)
  - Documentation OK (all output) -- View JavaDocs
- cartesian/internal
  - Clean compile (all output)
- cartesian/test
  - Clean compile (all output)
  - AllTests linked correctly
- Unit Test Results

<table>
<thead>
<tr>
<th>Platform</th>
<th>Results</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux x86 2.4.x</td>
<td>Tests run OK (1724 tests) (stderr)</td>
<td></td>
</tr>
<tr>
<td>MacOS X 10.4.7 (G4 1.4GHz 1GB)</td>
<td>No test summary in results file (stderr)</td>
<td>Last updated: Sep 17 2007</td>
</tr>
<tr>
<td>Windows2008 Server x64</td>
<td>Tests run: 1724, Failures: 1, Errors: 0 (stderr)</td>
<td>Last updated: Jul 19 23:02</td>
</tr>
</tbody>
</table>

- Graphical status history
- Unit test history
- Source code stats
- Package dependence summary
- Jumble unit test quality analysis (log, queue: classes, packages, prequeue)
- Emma coverage results
- log from this run (previous log)
## Historical Table

<table>
<thead>
<tr>
<th>Date</th>
<th>CVS Update</th>
<th>src compile</th>
<th>src docs</th>
<th>internal compile</th>
<th>test compile</th>
<th>test linkage</th>
<th>tests</th>
<th>ant build</th>
<th>Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-07-18 17:00</td>
<td>9 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 16:30</td>
<td>10 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 16:00</td>
<td>5 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 15:30</td>
<td>5 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 15:10</td>
<td>5 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 14:30</td>
<td>529 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td></td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 13:50</td>
<td>124 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td>No build status</td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 13:20</td>
<td>5 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td>No build status</td>
<td>ERROR</td>
</tr>
<tr>
<td>2008-07-18 13:10</td>
<td>5 files changed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OK (1724 tests)</td>
<td></td>
<td>No build status</td>
<td>ERROR</td>
</tr>
</tbody>
</table>
cvscheck - 15mins

- Compilation with and without test code
- Running unit tests
  - full set of 1,700 tests every 15 mins
  - In jar file and locally
- Comments to html documentation
Source code breakdown as at 2008-07-20

Lines
50,000

[Bar chart showing code and comments lines for three CVS Modules: cartesian/src, cartesian/internal, and cartesian/test.]
cvscheck cont.

- Source code checks
  spelling, style, variable name conventions
- Full release build
  - Every 6 hours
  - Tested on four platforms (Linux, MacOSX, Windows, HPC, Windows Server)
Testing – If you can’t measure it why should you believe it?

Test in multiple ways:

- Unit tests
- “assert” statements
- Pre/post conditions
- Others ...
Unit Tests

- Independent
  important when isolating problems
- Can be run automatically
  speeds process of integration and
  makes cycles faster
What are unit tests easiest for?

- “Small” self contained classes with clean interface and well defined state
- Facilities for constructing more complex environments
- Can easily set up many tests for different classes that implement the same interface
What are unit tests good for?

- Fast feedback on system wide errors in a shared environment
- Documenting usage
- New hires - what are the local coding standards?
What do unit tests find hard

- Non-deterministic code
- Timing dependent code
- Strongly system and environment dependent code
- GUI (human artistic judgment) although regression testing is possible
How good are our unit tests?

- Necessary to test coverage of tests.

- “Coverage” of lines not enough – just because a line was executed and didn’t throw an exception doesn’t tell us much.

- Hence use mutation testing where code “mutated” and unit tests run to check if the mutation found.
jumble

- Jumble - Computes score for how well testing covers code - runs incrementally

Now an open source project. Combination of work by Reel Two, student projects and department.
Module: cvscheck_cartesian
Package: com.reeltwo.cartesian.index (enqueue all classes in this package)
Average jumble score: 92%
Last updated: 2008-07-18 16:39:52

/com/reeltwo/cartesian/index/

- 93% collection
- 93% hash
- 96% params
- 60% similarity

100% [64] BitVector (2008-07-18)
100% [12] HashBitHandle (2008-07-18)
100% [17] HashBitVector (2008-07-18)
100% [0] IdPositionTranslation (2008-07-17)
100% [0] IdTranslation (2008-07-17)
88% [304] IndexImplementation (2008-07-18)
100% [27] IndexImplementationUtils (2008-07-18)
81% [93] QuickSort (2008-07-18)
85% [174] Utils (2008-07-18)
Jumble results for com.reeltwo.cartesian.index.IndexImplementation

Enqueue this class for a jumble run
View JavaDoc

Jumble run at 2008-07-18 14:42:21
Mutating com.reeltwo.cartesian.index.IndexImplementation
Tests: com.reeltwo.cartesian.index.IndexImplementationTest
Mutation points = 304, unit test time limit 5.83s

M FAIL: com.reeltwo.cartesian.index.IndexImplementation:171 CP[44] "Threshold must be positive:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:219 CP[80] "Index sort:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:223 CP[82] "Index overflow:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:226 CP[84] "Index position:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:317 CP[95] "count:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:317 CP[96] "is:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:329 CP[98] "is:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:351 CP[99] "low:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:352 CP[100] "high:" -> "_jumble"
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:366 CP[102] "x:" -> "_jumble"

-----------

-----------

-----------

-----------
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:335 CP[335] negated conditional
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:335 CP[335] 0L -> 1L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:335 CP[335] 1L -> 0L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:335 CP[335] negated conditional

-----------
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:316 CP[316] 1L -> 0L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:316 CP[316] 0L -> 1L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:316 CP[316] 1L -> 0L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:316 CP[316] 0L -> 1L

-----------
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:320 CP[320] 0L -> 1L
M FAIL: com.reeltwo.cartesian.index.IndexImplementation:320 CP[320] 1L -> 0L

-----------
Score: 88%
Other testing

- “assert” statements at loop/statement level
- Integrity constraints (pre/post-conditions) at instance/class level
- Good techniques for non-deterministic code (early detection of failure)
- Interacts well with unit tests
Documentation of code

- Use the standard “javadoc” system
- checkstyle will insist on all public methods and classes being documented.
- I will insist that the documentation be good quality
Good documentation

- Documenting data structures is more important than operations on them
- Too much documentation is bad
- Document *why* a method exists rather than what it does
<table>
<thead>
<tr>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.reeltwo.annotate</td>
</tr>
<tr>
<td>com.reeltwo.annotate.chemsmart</td>
</tr>
<tr>
<td>com.reeltwo.annotate.entity</td>
</tr>
<tr>
<td>com.reeltwo.bean.bunker.uspto</td>
</tr>
<tr>
<td>com.reeltwo.bean.bunker.wopct</td>
</tr>
<tr>
<td>com.reeltwo.bean.cs</td>
</tr>
<tr>
<td>com.reeltwo.bean.entity</td>
</tr>
<tr>
<td>com.reeltwo.bean.misc</td>
</tr>
<tr>
<td>com.reeltwo.bean.network</td>
</tr>
<tr>
<td>com.reeltwo.bean.nlp</td>
</tr>
<tr>
<td>com.reeltwo.bean.search</td>
</tr>
<tr>
<td>com.reeltwo.bean.search.db</td>
</tr>
<tr>
<td>com.reeltwo.bean.util</td>
</tr>
<tr>
<td>com.reeltwo.bunker.alchemy</td>
</tr>
</tbody>
</table>

This package contains Reel Two manually curated chemical information.
Class GrantRedBook25Handler

java.lang.Object

com.reeltwo.bunker.patent.AbstractPatentHandler
com.reeltwo.bunker.uspto.USPTOPatentHandler
com.reeltwo.bunker.uspto.USPTOXMLPatentHandler
com.reeltwo.bunker.uspto.GrantRedBook25Handler

All Implemented Interfaces:

public class GrantRedBook25Handler
extends USPTOXMLPatentHandler

An XML Handler for the Grant Red Book 25 DTD.

Version:
$Revision$

Author:
Richard Allen

Field Summary

Fields inherited from class com.reeltwo.bunker.uspto.USPTOPatentHandler
RELATIONSHIP_ASSIGNEE, RELATIONSHIP_ASSISTANT_EXAMINER, RELATIONSHIP_ATTORNEY_OR_AGENT,
RELATIONSHIP_INVENTOR, RELATIONSHIP_PRIMARY_EXAMINER

Constructor Summary

GrantRedBook25Handler()  

Method Summary

void characters(char[] ch, int start, int length)  

void endElement(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName)  

/**
   * Ensure that the working directory for the environment exists and all
   * the remote contents have been copied over.
   * @param remote directory where the remote environment file is to be found.
   * @param local the local working directory.
   * @param name name of the environment.
   * @throws IOException
   */

private void getEnvironment(
    final RemoteFile remote, final File local, final String name)
throws IOException {

cvscheck cont.
Cycles and Times

- Code construction unit and style checking - interactive to minutes
- cvscheck - 15mins to 6 hours
- Performance and extreme testing - days
- Release - weeks and months
- Product sales - months to years
Programming Cycle - 10s lines of code (personal task)

Selected Task

Type code

Compilation
Unit tests

~10s minutes

Speed up using IDE

Quiet point when local tests pass

Personal copy of code
Programming Cycle - single bug

~8 hours
Speed up with svn and cvscheck
Quiet point when all central tests pass

Selected Bug → Fix Bug → Shared Code
Continuous Integration
Release Cycle

~weeks to months

Quiet point when user accepts

System Construction

Feature Selection

Code Documentation hardware

User inspection
Product Cycle

~months to years

Quiet point when getting Revenue from product

Product specification

System Construction

Code Documentation hardware

Market feedback

$$
How to improve a cycle

- make it fast
- remove unnecessary work
- automate where possible
- plan the quiet points

- get as much feedback as possible
- look for “risks”
Programming Iterations

Following our principles above
  make each iteration as short as possible
  less than one week (8 hours programming)
  in some cases hours or minutes

Think about the iteration beforehand
  you may be surprised how small and short they can be
Terms introduced

**Cycle** - process of performing action and receiving **feedback** on result

**Quiet Point** - when a cycle has achieved its goal

**Continuous integration** - cycle where programmers contribute code to a shared repository and global testing is done

**Unit tests** - tests that are independent and automated
Summary

Can view software projects as cycles of activities

Modern “agile” programming emphasizes the value of short productive cycles

The goal is to deliver value - not to spend time doing useless make work

Automated tools are an important way of speeding and improving such cycles