Test Automation: An Empirical Perspective.

Part III – Testing Plug-ins

Long Tutorial at the GTTSE Summer School on Generative and Transformational Techniques in Software Engineering, Braga, Portugal, 2011

Arie van Deursen, Delft University of Technology
Joint work with Michaela Greiler (TU Delft)
Out of the comfort zone!
Plug-in Architectures

Create series of tailored products by combining, configuring, & extending plug-ins
Indigo

Indigo
62 Projects
46 Million Lines of Code

http://www.eclipse.org/blogs/author/irbull/
Eclipse Plug-in Architecture

The Maestro rover operations application displaying images received from the Spirit and Opportunity Mars Rovers. The Eclipse Rich Client Platform is the core foundation for Maestro and provides features such as a help system, automatic update, and organization of its many data views.

Eclipse-based software can be extended by adding different plug-ins, for example, Eclipse-based Flex applications for the Flash Platform; Adobe Flex Builder, Adobe IDE based on Eclipse for building Flex applications; Composed by, a Visual Platform for Desktop and Web Applications; Eclipse-based development environment for Java applications; and in general, Eclipse integration for multiple functions.

Eclipse is an integrated workbench framework to access the power of existing grid infrastructures. It is used for development, making and deploying Java-based applications, running them in simulation, and analyzing the results. Eclipse provides a flexible environment for developers to work on different types of projects, including web, Android, Java, and more. It includes a powerful set of tools for software development, such as code editors, debuggers, and profilers.

Eclipse provides a rich set of components for developing, deploying, and maintaining Java-based applications. The plug-in architecture allows for easy extension and customization of the platform to suit specific needs. Eclipse is open source, and its community is actively involved in improving and expanding its capabilities. It is widely used in the software development industry for its powerful features and extensive plugin ecosystem.
Underneath: OSGi

- Routers, Modems, Gateways, Control Panels, Phones, Cars, Trains, Trucks, Healthcare devices...
OSGi / Eclipse Plug-ins / Services

- OSGi: Dynamic modularization for Java
- Eclipse plug-ins are OSGi bundles
- Eclipse plug-ins can “extend” each other
- OSGi services
- e4 architecture

One Product = Many Plug-ins
Set of Plug-ins = Many Products
import junit.framework.Test;
import junit.framework.TestSuite;
import org.eclipse.mylyn.common.tests.AllCommonTests;
import org.eclipse.mylyn.content.tests.AllContextTests;
import org.eclipse.mylyn.discovery.tests.AllDiscoveryTests;
import org.eclipse.mylyn.ide.tests.AllIDETests;
import org.eclipse.mylyn.java.tests.AllJavaTests;
import org.eclipse.mylyn.monitor.tests.AllMonitorTests;
import org.eclipse.mylyn.resources.tests.AllResourceTests;
import org.eclipse.mylyn.tasks.tests.AllTaskTests;
import org.eclipse.mylyn.team.tests.AllTeamTests;
import org.eclipse.mylyn.tests.integration.AllIntegrationTests;
import org.eclipse.mylyn.tests.misc.AllMiscTests;

public class AllNonConnectorTests {

    public static Test suite() {
        // the order of these tests might still matter, but shouldn't
        TestSuite suite = new TestSuite("All Non-Connector Tests for org.eclipse.mylyn.tests");
        suite.addTest(AllCommonTests.suite());
        suite.addTest(AllContextTests.suite());
        suite.addTest(AllDiscoveryTests.suite());
        suite.addTest(AllIDETests.suite());
        suite.addTest(AllJavaTests.suite());
        suite.addTest(AllMonitorTests.suite());
        suite.addTest(AllResourceTests.suite());
        suite.addTest(AllTaskTests.suite());
        suite.addTest(AllTeamTests.suite());
        suite.addTest(AllIntegrationTests.suite());
        suite.addTest(AllMiscTests.suite());
        return suite;
    }
}
ETSE: Eclipse Test Suite Explorer
Test Implications?

**Plug-in Architectures**

- Do plug-ins simplify the life of the tester?
- What can go wrong?
- Is there a testing strategy to identify that?

**YOUR Technique**

- How does it make testing easier?
- What can go wrong?
- What testing strategy do you recommend?
Test Implications for Plug-in Architectures?

Fault model?
• Interacting plug-ins,
• Plug-in configurations
• Plug-in versions
• Plug-in extensions
• Resource usage
• ...

Test Approaches?
• Combinatorial
• Multi-version
• As part of product line engineering
• ...

12
Testing Component-Based Systems?

• N paths + M paths $\rightarrow$ N * M paths

• Too many “multiple-component defects”

• Integration patterns
Model-Driven Software Product Line Testing: An Integrated Approach

Andy Schürr (1), Sebastian Oster (1), Florian Markert (2)

Testing a Software Product Line
CMU/SEI-2001-TR-022
ESC-TR-2001-022
John D. McGregor

SOFTWARE PRODUCT LINE TESTING
A Systematic Review

Beatriz Pérez Lamancha
Software Testing Centre (CES), School of Engineering, University of the Republic, Montevideo, Uruguay
bperez@fing.edu.uy

Iacario Polo Usos, Mario Piattini Velthuis
Research Group, Information Systems and Technologies Department, University of Castilla-La Mancha, Ciudad Real, Spain
iacario.polo@uclm.es, mario.piattini@uclm.es

Product line, systematic review.

This is a set of software tools, method and managed set of core assets in a prescribed way (Kopetz, 2002). In Europe, the term SPL (Software product line) is used (van der Linden, 2002). In software engineering, the reuse is proactive and systematic as with classic development; the software is first designed and then reused (Kneuer, 2006). SPL includes the intensive use of models, all with the goal of reusing the investment (which will be high) to recompense the individual product. In SPL, the best practices of software engineering will be used to improve the research and state of the practice in integrated in this context.

Recently, Bertolino (Bertolino, 2007) presented a general analysis of the state of the art in testing research which serves as a roadmap for the most relevant challenges. This work begins with some important past achievements, while its destination consists of four identified goals which research tends, but which remain unreachable. She calls these dreams. The routes from achievements to dreams are paved by outstanding research challenges. The four dreams are: universal test theory, test-based modelling, 100 percent automatic testing and efficacy-maximized test engineering. She also distinguishes the transversal challenges that run through all four of the identified dreams. One of them is testing within the emerging development paradigm, in which the software product line can be categorised.

This work presents a systematic review of the literature (Kitchenham, 2004) which deals with testing in software product lines. Our objective is to analyse the existing approaches to testing in software product lines, discussing the significant issues related to this area of knowledge and providing an up-to-date state of the art that can serve as a basis for innovative research activities. As mentioned earlier, SPL articulates and applies the best practices and techniques.
What do Eclipsers Think about Testing?
Impact Panel

When: Friday, May 27 10:30AM
Location: South Pacific 1&2

Panel Abstract:
This Impact Panel Session is intended to be the focus of a discussion surrounding impact: What do we think research impact is? What do others think about impact? How should impact be determined and measured? How can we increase our community’s research? What new directions might the Software Engineering community take to stimulate thinking and action in examining these issues? How do the different communities define impact?

Panelists:
Carlo Ghezzi, Politecnico di Milano
Pete Rotella, Cisco Research
Richard N. Taylor, University of California, Irvine

Chair: Leon Osterweil
Organizer: Leon Osterweil

Dan Berry and many others: sometimes, researchers need to SHUT UP AND LISTEN to practitioners. #icse2011

27 May via web  Favorited  ReTweeted  Reply
# Example Studies

<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigby</td>
<td>Reviews in open source</td>
<td>ICSE 2011</td>
</tr>
<tr>
<td>Treude</td>
<td>Communication of software dev. Knowledge</td>
<td>ESEC FSE 2011</td>
</tr>
<tr>
<td>Dagenais</td>
<td>Developer documentation</td>
<td>FSE 2010</td>
</tr>
<tr>
<td>Hermans</td>
<td>Spreadsheet challenges</td>
<td>ICSE 2011</td>
</tr>
<tr>
<td>Hoda</td>
<td>Agile development</td>
<td>OOPSLA 2010</td>
</tr>
</tbody>
</table>
Grounded Theory

- Systematic procedure to discover theory from (qualitative) data
- Theoretical sensitivity
- Open coding
- Theoretical coding
- Theoretical sampling
- Constant comparative method
- Selective coding
- Memoing


What’s a Theory?

“A set of well-developed categories (e.g. themes, concepts) that are systematically inter-related through statements of relationships to form a framework that explains some relevant social, psychological, educational, or other phenomenon.”

Research Questions

• What testing practices are adopted in the Eclipse community?

• Why are certain practices adopted, and why are others not adopted?

• Does the plug-in nature of Eclipse have an impact on software testing?
<table>
<thead>
<tr>
<th>P</th>
<th>Role</th>
<th>CR</th>
<th>TS</th>
<th>Technology</th>
<th>KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>developer</td>
<td>C</td>
<td>4-7</td>
<td>Eclipse plug-in</td>
<td>closed</td>
</tr>
<tr>
<td>P2</td>
<td>project lead</td>
<td>O</td>
<td>6</td>
<td>Eclipse plug-in</td>
<td>90</td>
</tr>
<tr>
<td>P3</td>
<td>tester</td>
<td>C</td>
<td>7-8</td>
<td>Eclipse plug-in, RCP product</td>
<td>370</td>
</tr>
<tr>
<td>P4</td>
<td>developer</td>
<td>O</td>
<td>3-10</td>
<td>Eclipse plug-in</td>
<td>90</td>
</tr>
<tr>
<td>P5</td>
<td>developer</td>
<td>C</td>
<td>3-7</td>
<td>OSGi</td>
<td>280</td>
</tr>
<tr>
<td>P6</td>
<td>project lead</td>
<td>O</td>
<td>6-9</td>
<td>Eclipse plug-in</td>
<td>1700</td>
</tr>
<tr>
<td>P7</td>
<td>project lead</td>
<td>O</td>
<td>2-5</td>
<td>Eclipse plug-ins</td>
<td>50</td>
</tr>
<tr>
<td>P8</td>
<td>project lead</td>
<td>O</td>
<td>12</td>
<td>Eclipse plug-in</td>
<td>670</td>
</tr>
<tr>
<td>P9</td>
<td>project lead</td>
<td>O</td>
<td>3</td>
<td>Eclipse plug-in</td>
<td>90</td>
</tr>
<tr>
<td>P10</td>
<td>test manager</td>
<td>C</td>
<td>20-50</td>
<td>Eclipse plug-in, RCP product</td>
<td>closed</td>
</tr>
<tr>
<td>P11</td>
<td>developer</td>
<td>O</td>
<td>7-11</td>
<td>Eclipse plug-in</td>
<td>710</td>
</tr>
<tr>
<td>P12</td>
<td>project lead</td>
<td>O</td>
<td>1-2</td>
<td>Eclipse plug-in</td>
<td>12 &amp; 56</td>
</tr>
<tr>
<td>P13</td>
<td>project lead</td>
<td>O</td>
<td>5-7</td>
<td>Eclipse plug-in</td>
<td>2000</td>
</tr>
<tr>
<td>P14</td>
<td>developer</td>
<td>C</td>
<td>5</td>
<td>RCP product</td>
<td>350</td>
</tr>
<tr>
<td>P15</td>
<td>project lead</td>
<td>O</td>
<td>20</td>
<td>RCP product</td>
<td>850</td>
</tr>
<tr>
<td>P16</td>
<td>developer</td>
<td>O</td>
<td>7-10</td>
<td>Eclipse plug-in</td>
<td>1500</td>
</tr>
<tr>
<td>P17</td>
<td>developer</td>
<td>C/O</td>
<td>5-6</td>
<td>Eclipse plug-in</td>
<td>2500</td>
</tr>
<tr>
<td>P18</td>
<td>project lead</td>
<td>C</td>
<td>4</td>
<td>RCP product</td>
<td>100</td>
</tr>
<tr>
<td>P19</td>
<td>developer</td>
<td>C/O</td>
<td>6-9</td>
<td>Eclipse plug-in</td>
<td>2500</td>
</tr>
<tr>
<td>P20</td>
<td>developer</td>
<td>O</td>
<td>7-10</td>
<td>RCP product</td>
<td>1000</td>
</tr>
<tr>
<td>P21</td>
<td>developer</td>
<td>C</td>
<td>4-10</td>
<td>RCP product</td>
<td>80-100</td>
</tr>
<tr>
<td>P22</td>
<td>developer</td>
<td>C</td>
<td>3-5</td>
<td>Eclipse distribution</td>
<td>140</td>
</tr>
<tr>
<td>P23</td>
<td>project lead</td>
<td>C</td>
<td>5-7</td>
<td>RCP product</td>
<td>closed</td>
</tr>
<tr>
<td>P24</td>
<td>developer</td>
<td>C</td>
<td>8</td>
<td>RCP product</td>
<td>400</td>
</tr>
<tr>
<td>P25</td>
<td>project lead</td>
<td>C</td>
<td>7-12</td>
<td>RCP product</td>
<td>closed</td>
</tr>
<tr>
<td>Domain</td>
<td>Project and/or Company</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDEs, Eclipse Distribution</td>
<td>Yoxos, EclipseSource</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOA</td>
<td>Mangrove, SOA, Inria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI Testing Tool</td>
<td>GUIDancer, Bredex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version Control Systems</td>
<td>Mercurial, InlandSoftware</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modeling</td>
<td>xtext, Itemis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence layer</td>
<td>IMP, University of Amsterdam</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain Specific Language</td>
<td>CDO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPM Solutions</td>
<td>Spoofax, Delft University of Technology</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI Testing Tool</td>
<td>GMF, BonitaSoft</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverage Analysis</td>
<td>Q7, Xored</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modeling</td>
<td>EclEmma</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPM Solutions</td>
<td>EMF, Itemis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scientific data acquisition</td>
<td>RCP product, AndrenaObjects</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime platform</td>
<td>OpenGDA, Kichacoders</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Management system</td>
<td>RAP, EclipseSource</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded Software</td>
<td>Mylyn, Tasktop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCP product</td>
<td>MicroDoc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EclipseSource</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Code</td>
<td>Comment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>-------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1, P4</td>
<td>The best</td>
<td>Unit test are the best</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Legacy code</td>
<td>legacy code can be problematic to tests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Fast, easy to execute</td>
<td>They are fast, and easy to execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Misused as Integration test</td>
<td>Unit tests grows into PDE test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Refactoring easy</td>
<td>refactoring is easier when you have a well design unit test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>More important than Integration test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Limited applicability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>High amounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Frequent execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Only complex stuff</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td>Comfort for refactoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P16</td>
<td>Limited applicability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P20</td>
<td>Limited applicability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Image of Challenges of Unit Tests](image)

- Highly Coupled Code Parts
  - Working with frameworks
  - Misused as integration tests
  - Legacy Code
  - Misleading Coverage
- Limited applicability
  - Application concurrency
  - Application within Application
  - Client-Server

---

### Challenges of Unit Tests

**Highly Coupled Code Parts**

- Working with frameworks
- Misused as integration tests
- Legacy Code
- Misleading Coverage

**Limited applicability**

- Application concurrency
- Application within Application
- Client-Server

---

**Presentation Notes**

- Integration tests on a daily basis
- He skips two slides about database setup and environment
- Talks around 3 minutes about integration tests
- Favorite topic
- He presents it as it is not easy

**Utility**

- In the IDE? SET-UP
- Example: TOOLS/UTILITY
  - Shows that he can even make screen shots (UTILITY) during test execution to debug root cause analysis (TASK during TESTING)
From Code to Category (Hoda et al)

Figure 1. a: Levels of data abstraction in GT b: Emergence of category *Lack of Customer Involvement* from concepts.
Resulting Theory

Theory comprises four main categories:

1. Testing practices used by Eclipsers
2. Impact of the plug-in characteristic
3. Factors affecting test practice adoption
4. The role of the community
Unit testing is popular

“Unit testing is where you find the most bugs”

“We don’t have UI tests, because we can cover that with unit tests”

“At least 70% of our test effort is spent on unit testing.”
Findings 1: Test Practices

- Unit testing popular:
  - 1000s of unit tests running in few minutes
  - Coverage in lines
  - Nightly builds

- PDE Runner used for tests requiring ecosystem

- Requirements:
  - Issue tracker / stories
  - (e)Fitnesse

- GUI testing used for functional testing
  - Automation limited (maintainability)
  - Mostly manual

- Test-Driven Development:
  - too hard due to ecosystem

- Models/UML: *not mentioned*
# Eclipse Testing Study

## 2. Test Activities

When filling in this survey, please relate the answers always to the current or most recent Eclipse-based software project you have been part of.

- **Unit Testing**: typically comprises a small executable like a method, class, or several related classes.
- **Integration Testing**: testing with the intent of finding bugs in component (plug-in) interactions.
- **System Testing**: tests execute the entire system.

### 4. Please estimate the relative effort spent on each test technique.
(The total effort spent should not exceed ~100%.)

<table>
<thead>
<tr>
<th></th>
<th>90-100%</th>
<th>75-90%</th>
<th>50-25%</th>
<th>25-10%</th>
<th>&lt;10%</th>
<th>0%</th>
<th>I don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Integration Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>GUI Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>System Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

### 5. Please indicate the level of test automation?
(If you do not practice a technique please check "not applicable").

<table>
<thead>
<tr>
<th></th>
<th>fully automated</th>
<th>main effort test automation</th>
<th>main effort manual</th>
<th>only manual testing</th>
<th>I don't know</th>
<th>not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Integration Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>GUI Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>System Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
Automated or Manual?

- **Unit Testing**
  - Fully automated: 70%
  - Main effort test automation: 20%
  - Main effort manual: 10%
  - Only manual testing: 0%
  - I don't know: 0%

- **Integration Testing**
  - Fully automated: 80%
  - Main effort test automation: 10%
  - Main effort manual: 5%
  - Only manual testing: 5%
  - I don't know: 0%

- **GUI Testing**
  - Fully automated: 90%
  - Main effort test automation: 5%
  - Main effort manual: 4%
  - Only manual testing: 1%
  - I don't know: 0%

- **System Testing**
  - Fully automated: 10%
  - Main effort test automation: 30%
  - Main effort manual: 60%
  - Only manual testing: 0%
  - I don't know: 0%
Cross plug-in testing is optional

“We do bug-driven cross plug-in testing”

“We have no automated tests for cross plug-in testing, but we do manual testing.”
Version testing is minimal

“A lot of people put version ranges in their bundle dependencies, and they say they can run with 3.3 up to version 4.0 of the platform.”

“But I’m willing to bet that 99% of the people do not test that their stuff works.”
Findings 2: Plug-ins

• Testing deferred to application engineering
  – No special test effort at ‘product line engineering’ level

• Integration testing on demand:
  – Feature that can’t be tested in isolation
  – Bug occurring in the field

• No test effort aimed at integration faults per se
  – Versions, configurations, ...
Why are Techniques Unpopular?

“It’s complicated to integrate JUnit with the build. Another framework? I didn’t want to take the trouble”

“The Capture and Replay nature of QF-tests was too rigid when the system was evolving”

“We haven’t been 100% satisfied with capture-replay: too much is captured.”
Organizational Factors

“It’s very hard for people to motivate themselves if you do not have enough time to do something more carefully.”
Findings 3: Adoption of Practices

Organizational factors
Organization
- Tester status
- Integration responsibility
- Reuse strategy
Project / Application
- Availability of required information
- Recognizable benefits
- Lack of time

Technical factors
Technology
- Execution time
- Setup time
- Maturity of tooling
Application
- Testability w.r.t technique
- Coupling (ecosystem)
Organizational Problems Experienced

- Testing less appreciated than development activities
- Unclear who is/feels responsible for overall quality
- Restricted controllability of foreign plug-ins
- Unclear or unknown design documents or end user requirem.
- Practice too time consuming
- Unclear who is/feels responsible for performing this test practice
- No recognizable benefits of this test practice
- Test technique is perceived as less important
- Lack of time for this test practice
Technical Impediments Experienced

- Long test execution time
- Immature tooling or missing test infrastructure
- High maintenance effort
- Hard to test code tightly coupled to Eclipse
- Hard to test highly coupled, or legacy code
- Difficult to set-up test execution environment
- Lack of knowledge or expertise
“We use self-hosting as test technique. That is, we use our software regularly”

“...the real tests are coming from the users, who are doing all kind of different things with [project 9].”
Community Testing

Testing is done by the user community. [...] We have more than 10,000 installations per month. If there should be a bug it gets reported immediately.”

“The community helps to test the system for different operating systems, and versions. They are very active with that.”
Downstream Testing

“We’re a framework. If the user downloads a new version and lets his application run with it, then this is already like a test.”

“They have extensive unit tests, and so I am quite sure that when I break something, somebody downstream very rapidly notices and reports the problem.”
Community Testing

“If there are problems, people definitely report them, so you do find out about problems.”

“Yes, for the GEF part we find bugs and we report bugs.”
“I would say the majority of the bug reports come from the community. [...] We have accepted more than 800 patches.”

“We make all infrastructure available, [...], so that somebody who write a patch has the opportunity to run the same tests [...]”
Findings 4: “Compensation Strategies”

• Community plays key role in finding, reporting, and fixing issues
• Downstream testing (manual and automatic) provides additional tests of upstream framework.
• Open test infrastructure facilitates patching
Involvement of the Community

- Automated testing (including community, downstream project)
- Manual testing (including GUI, combinatorial testing)
- Providing bug fixes
- Providing bug reports or feature requests
- Giving feedback, and foster discussions
The Emerging Picture:

• Eclipsers are proud of their unit testing
• But dislike integration testing:
  – too slow to execute,
  – too many possibilities
  – too hard to write
• Instead:
  – Eclipsers rely on community to report issues
  – Product line engineers rely on application engineers to do downstream testing
Implications?

• Can upstream projects simplify downstream testing?

• Can the community reward downstream testing?

• Can upstream testing be better supported?
Chapter VI

Rigor In Grounded Theory Research: An Interpretive Perspective on Generating Theory From Qualitative Field Studies

Susan Gasson,
Drexel University, USA

ABSTRACT

This chapter presents a set of principles for the use of Grounded Theory techniques in qualitative field studies. Some issues and controversies relating to rigor in Grounded Theory generation are discussed. These include: inductive theory generation and emergence, how theoretical saturation may be judged, the extent to which coding schemes should be formalized, the objectivist-subjectivist debate, and the assessment of quality and rigor in interpretive research. It is argued that Grounded Theory is often criticized for a lack of rigor because we apply positivist evaluations of rigor to research that derives from an interpretive worldview. Alternative assessments of rigor are suggested, that emphasize reflexivity in the inductive-deductive cycle of substantive theory generation.
<table>
<thead>
<tr>
<th>Issue of concern</th>
<th>Positivist View</th>
<th>Interpretive View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representativeness</strong> of findings</td>
<td><em>Objectivity</em>: free from researcher bias</td>
<td><em>Confirmability</em>: conclusions depend on subjects, not on researcher</td>
</tr>
<tr>
<td><strong>Reproducibility</strong></td>
<td><em>Reliability</em>: findings can be replicated</td>
<td><em>Auditability</em>: process is consistent &amp; stable over time</td>
</tr>
<tr>
<td><strong>Rigor</strong> of method</td>
<td><em>Internal validity</em>: statistically significant</td>
<td><em>Credibility</em>: finding relevant and credible to people we study</td>
</tr>
<tr>
<td><strong>Generalizability</strong> of findings</td>
<td><em>External validity</em>: domain of generalizability</td>
<td><em>Transferability</em>: how far can findings be transferred to other contexts?</td>
</tr>
</tbody>
</table>
Thanks!

• Monday: Reflections on *impact & empirical methods*

• Yesterday: Reflections on *test automation for the web*

• Today: Reflections on *compositional testing*