Testing Plug-in Architectures

Arie van Deursen
@avandeursen

Joint work with Michaela Greiler
Margaret-Anne Storey
The TU Delft
Software Engineering Research Group

Education

• Programming, software engineering
• MSc, BSc projects

Research

• Software architecture
• Software testing
• Repository mining
• Collaboration
• Services
• Model-driven engineering
• End-user programming
Crawljax: Automated Testing of Ajax Applications

Plug-in Architectures

Create series of tailored products by combining, configuring, & extending plug-ins
Dear @github, please let us build plugins
#githubplugins
aosabook.org/blog/2012/09/d...
“The number one reason people give us for not upgrading to the latest version of WordPress is fear that their plugins won’t be compatible.”

Indigo

62 Projects
46 Million Lines of Code

http://www.eclipsesource.com/blogs/author/irbull/
List of Eclipse-based software

The Eclipse platform can be extended by adding different plug-ins, for example:

- **Adobe Flex Builder**, Adobe IDE based on Eclipse for building Flex applications for the Flash Platform
- **AnyLogic**, a simulation modeling tool developed by XJ Technologies
- **Aptana**, Web IDE based on Eclipse (commercial and community version)
- **Avaya Dialog Designer**, a commercial IDE to build scripts for voice self-service applications
- **BioEclipse**, a visual platform for chemico- and bioinformatics
- **BIRT Project**, open source software project that provides reporting and business intelligence capabilities for rich client and web applications
- **Bonita Open Solution** relies on Eclipse for the modeling of processes, implementing a BPMN and a Web form editors
- **Borland JBuilder 2007**, based on Eclipse
- **ColdFusion Builder** is a plug-in for Eclipse that provides an Integrated Development Environment and editor for the ColdFusion Programming Language
- **eCLIPSE** is a plug-in that integrates with Eclipse to test Web, Java, .NET, Siebel, SAP and Oracle applications
- **EasyEclipse**, bundled distributions of the Eclipse IDE
- **g-Eclipse**, an integrated workbench framework to access the power of existing Grid infrastructures
- **OpenArchitectureWare**, a distributed infrastructure

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.

*OCFL, Oracle Enterprise Pack for Eclipse*
Underneath: OSGi

- Routers, Modems, Gateways, Control Panels, Phones, Cars, Trains, Trucks, Healthcare devices...
What are the test implications?

How should we test plug-in architectures?
Plug-in Testing Issues to Consider

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
- Search-based through interaction space
- ...

See related work (slide 5) of #sast2012
Teste de Linha de Produto de Software Baseado em Mutação de Variabilidades. Marcos Quinaia (UNICENTRO), Johnny Ferreira (UFPR), Silvia Vergilio (UFPR).
What do Eclipsers Think about Testing?
Research Questions

1. What testing practices are prevalent in the Eclipse community?

2. Does the plug-in nature of Eclipse have an impact on software testing?

3. Why are certain practices adopted, and why are others not adopted?

4. Are there additional compensation strategies used to support testing of plug-ins?
## 25 Interviews

<table>
<thead>
<tr>
<th>Domain</th>
<th>Project and/or Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDEs, Eclipse Distribution</td>
<td>Yoxos, EclipseSource</td>
</tr>
<tr>
<td>SOA</td>
<td>Mangrove, SOA, Inria</td>
</tr>
<tr>
<td>GUI Testing Tool</td>
<td>GUIDancer, Bredex</td>
</tr>
<tr>
<td>Version Control Systems</td>
<td>Mercurial, InlandSoftware</td>
</tr>
<tr>
<td>Modeling</td>
<td>xtext, Itemis</td>
</tr>
<tr>
<td>Persistence layer</td>
<td>IMP, University of Amsterdam</td>
</tr>
<tr>
<td>Domain Specific Language</td>
<td>CDO</td>
</tr>
<tr>
<td>BPM Solutions</td>
<td>Spoofax, TU Delft</td>
</tr>
<tr>
<td>GUI Testing Tool</td>
<td>GMF, BonitaSoft</td>
</tr>
<tr>
<td>Coverage Analysis</td>
<td>Q7, Xored</td>
</tr>
<tr>
<td>Modeling</td>
<td>EclEmma</td>
</tr>
<tr>
<td>BPM Solutions</td>
<td>EMF, Itemis</td>
</tr>
<tr>
<td>Scientific data acquisition</td>
<td>RCP product, AndrenaObjects</td>
</tr>
<tr>
<td>Runtime platform</td>
<td>OpenGDA, Kichacoders</td>
</tr>
<tr>
<td>Task Management system</td>
<td>RAP, EclipseSource</td>
</tr>
<tr>
<td>Embedded Software</td>
<td>Mylyn, Tasktop</td>
</tr>
<tr>
<td>RCP product</td>
<td>MicroDoc</td>
</tr>
<tr>
<td></td>
<td>EclipseSource</td>
</tr>
</tbody>
</table>
Grounded Theory

- Systematic procedure to discover theory from (qualitative) data
- Coding
- Code, concept, category
- Memoing
- Theoretical sensitivity
- Theoretical sampling
- Constant comparison
- Saturation


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.”

<table>
<thead>
<tr>
<th>P</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, P4</td>
<td>The best</td>
<td>Unit test are the best</td>
</tr>
<tr>
<td>P1</td>
<td>Legacy code</td>
<td>Legacy code can be problematic to tests</td>
</tr>
<tr>
<td>P1</td>
<td>Fast, easy to execute</td>
<td>They are fast, and easy to execute</td>
</tr>
<tr>
<td>P1</td>
<td>Misused as Integration test</td>
<td>Unit tests grows into PDE test</td>
</tr>
<tr>
<td>P1</td>
<td>Refactoring easy</td>
<td>Refactoring is easier when you have a well design unit test</td>
</tr>
<tr>
<td>P4</td>
<td>More important than Integration test</td>
<td>Are more important</td>
</tr>
<tr>
<td>P4</td>
<td>Limited applicability</td>
<td>Leave parts out that cannot be tested, Remote calls</td>
</tr>
<tr>
<td>P4</td>
<td>High amounts</td>
<td>You can have many</td>
</tr>
<tr>
<td>P4</td>
<td>Frequent execution</td>
<td>Can be executed often</td>
</tr>
<tr>
<td>P8</td>
<td>Only complex stuff</td>
<td>Only for complex stuff like state machines</td>
</tr>
<tr>
<td>P13</td>
<td>Comfort for refactoring</td>
<td>It gives you a certain level of comfort to know that when you make a change and you break something that would be apparent in your test case</td>
</tr>
<tr>
<td>P16</td>
<td>Limited applicability</td>
<td>For code within browser</td>
</tr>
<tr>
<td>P20</td>
<td>Limited applicability</td>
<td>For code within browser</td>
</tr>
</tbody>
</table>

**Concept 2.2. Plug-in characteristic**

To what specific test practices does the plug-in nature lead?

| 2.2.1 | No influence | The plug-in characteristic has no influence on testing. |
| 2.2.2 | Modularization | The plug-in mechanism is used for modularizing test suites. |
| 2.2.3 | Extension points | Test strategies for Eclipse extensions and extension points are adopted. |
| 2.2.4 | Registration untested | The plug-in and extension point registration mechanisms are untested. |
| 2.2.5 | Plug-in testability | Eclipse plug-ins can be hard to test if they do not expose their (internal) functionality. |
| 2.2.6 | Eco-system integration | Plug-ins are exercised in the context of the Eclipse runtime environment. |
| 2.2.7 | GUI based | Automated GUI testing is used to test GUI based Eclipse applications. |
| 2.2.8 | No eclipse integration | Tests do not require the Eclipse or OSGi runtime. |
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

Also 12 concepts, 100 codes, 100 memos

Triangulation

1. Resonance @ EclipseCon

2. Survey among 151 developers
Practices: Unit testing is popular

“Unit testing is where you find the most bugs”

“Ultimately, unit test are our best friends”

“At least 70% of our test effort is spent on unit testing.”
Other forms of testing are less popular

“We think that with a high test coverage through unit tests, integration tests are not necessary.”

“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.”
Findings 1: Practices

• Common practice to have no separate test teams

• Eclipsers are proud of their unit testing

• Eclipsers tend to dislike system, integration, UI, and acceptance testing
  – Substantially less automation
Automated or Manual?
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 effort during `product line engineering’

• Integration testing on demand:
  – Bug occurring in the field

• No test effort aimed at integration faults per se
  – Versions, configurations, interactions, ...
Testing combinations or versions?

43% don’t test integration of different products
only 3% test this thoroughly

55% don’t test for platform versions
only 4% test this thoroughly

63% don’t test for dependency versions
only 10% test this thoroughly
Barriers

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

“And you never know, once you write a good test, then it will become obsolete with the next version of Eclipse”

“Especially for plug-ins, we would need some best practices.”
Findings 3: Barriers

• **Responsibility** for integration unclear
• **Requirements** for composite unclear
• Lack of **ownership**
• Insufficient **plug-in knowledge**
• **Set-up** of test infrastructure too complicated
• **Test execution** too long
• Poor **testability** of the platform
Community Testing (I)

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.”
Community Testing (II)

“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 writes a patch has the opportunity to run the same tests [...]”
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.”
Findings 4: “Compensation Strategies”

• Community plays key role in finding and reporting issues

• Downstream testing (manual and automatic) provides additional tests of upstream framework.

• Open test infrastructure facilitates patching
Summary: Findings

1. (Automated) unit testing is widely adopted; Integration, system, UI and acceptance testing are much less automated

2. The plug-in nature has little direct impact on test practices

3. Barriers to adopt techniques include unclear ownership, accountability, and test effort & execution time

4. Limited integration testing is compensated by community
Scope

• Beyond the participants:
  – Challenged results in survey among 150 Eclipsers

• Beyond Eclipse:
  – Open source, developer centric, plug-in architecture, services, ...

• Beyond the people:
  – Repository mining, code analysis
(Eclipse) Implications

1. Community tolerance for failures determines (integration) test effort

2. Need to strengthen community

3. Need to strengthen plug-in architecture with “self testing” capabilities

4. Test innovations must address adoption barriers
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>Reproducibility</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>Generalizability</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>

"Ameaças à validade"
Impact for Your (SAST 2012) Paper?

2. Experience in Organizing Test Teams
3. Teste de Linha de Produto de Software Baseado em Mutação de Variabilidades
4. Execução Determinística de Programas Concorrentes Durante o Teste de Mutação
5. Uso de análise de mutantes e testes baseados em modelos: um estudo exploratório
6. Framework para Teste de Software Automático nas Nuvens
7. Usando o SilkTest para automatizar testes: um Relato de Experiência
8. Automating Test Case Creation and Execution for Embedded Real-time Systems
9. Controlando a Diversidade e a Quantidade de Casos de Teste na Geração Automática a partir de Modelos com Loop
10. Geração Aleatória de Dados para Programas Orientados a Objetos
Conclusions (1)
Increasing Dynamism

• We must accept that many deployed compositions are in fact untested.

• As the level of dynamism grows, we need to move from a priori testing to “in vivo” testing.

Rethink what your test approach might mean in a run time setting.
Conclusions (2)
In Vivo / Run Time / On line Testing

• Continuous assertion & health checking
• Active stimuli upon configuration change
• Learn relevant stimuli from past behavior

First steps in research, little adoption so far

[ DevOps is just the beginning ]
Conclusions (3):
Involving the Tester in Your Research

• To understand the problem to begin with
  – Qualitative study to start research

• To evaluate your results

• More painful, ... and more rewarding!
Summary

• A Grounded Theory study is a great way to understand what people are struggling with.

• Integration testing in plug-in architectures is hard and possibly costly.

• A cooperative community is a invaluable.

• We must and can strengthen post-deployment testing and reporting.
Further Reading

Michaela Greiler, Arie van Deursen & Margaret-Anne Storey.


Full report: TUD-SERG-2011-010