Continuous Testing in Eclipse

David Saff, Michael D. Ernst
MIT CSAIL
eTX 2004, Barcelona, Spain
Continuous testing:
inspired by continuous compilation

• Continuous compilation, as in Eclipse, notifies the developer quickly when a *syntactic error* is introduced:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Error]</td>
<td></td>
<td>Syntax error on token &quot;a&quot;, &quot;]&quot; expected</td>
</tr>
<tr>
<td>![Warning]</td>
<td></td>
<td>The method decode(String) from the type URLDecoder is deprecated</td>
</tr>
</tbody>
</table>

• Continuous testing notifies the developer quickly when a *semantic error* is introduced:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Error]</td>
<td></td>
<td>Test failure: testArithmetic(ct.test.MainTestSuite)</td>
</tr>
<tr>
<td>![Warning]</td>
<td></td>
<td>The method decode(String) from the type URLDecoder is deprecated</td>
</tr>
</tbody>
</table>
Outline

• Continuous testing: defined and motivated
• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design
• Next steps
Outline

• Continuous testing: defined and motivated

• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design

• Next steps
Continuous testing

• Continuous testing uses excess cycles on a developer's workstation to continuously run regression tests in the background as the developer edits code.
Goals of continuous testing

Continuous testing:

• No longer forces the developer to decide whether to test and what tests to run.
• Prevents long-standing regression errors.*
• Makes developer confident, not annoyed.

* Saff, Ernst, ISSRE 2003: Reducing wasted development time via continuous testing
Continuous testing made students more productive

<table>
<thead>
<tr>
<th>Treatment</th>
<th>N</th>
<th>Completed assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>No tool</td>
<td>11</td>
<td>27%</td>
</tr>
<tr>
<td>Continuous compilation</td>
<td>10</td>
<td>50%</td>
</tr>
<tr>
<td>Continuous testing &amp; continuous compilation</td>
<td>18</td>
<td>78%</td>
</tr>
</tbody>
</table>

$p < .03$

* Saff, Ernst, ISSTA 2004: An experimental evaluation of continuous testing during development
### Students appreciated continuous testing

<table>
<thead>
<tr>
<th>I would use continuous testing…</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>…for the rest of the course</td>
<td>94%</td>
</tr>
<tr>
<td>…for my own programming</td>
<td>80%</td>
</tr>
<tr>
<td>I would recommend the tool to others</td>
<td>90%</td>
</tr>
</tbody>
</table>
Outline

• Continuous testing: defined and motivated
• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design
• Next steps
Design principles, 1 of 2

• Reuse
  – Whenever possible, plug in and reuse

• Future reuse
  – When reuse is impossible, copy and paste to show where Eclipse could be more flexible
Design principles, 2 of 2

• Consistent experience
  – Don’t change expected behavior
  – Build on current developer metaphors

• Minimal distraction
  – Don’t swamp benefits by sapping attention

• Testability
  – Add testing-specific API’s when necessary
Outline

• Continuous testing: defined and motivated
• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design
• Next steps
Outline

• Continuous testing: defined and motivated
• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design
• Next steps
Eclipse auto-building: Static structure

- Project -> * Builder
- * Source file
  - * Java builder
  - * RMI builder
Eclipse auto-building: Dynamic behavior

Project * Builder *

Source file * Builder

Marker creates Java builder RMI builder

Marker changes * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Marker * Marker

Problem view updates

Build Manager notifies

Auto-build Thread runs

Auto-build Thread starts
Eclipse launching: Static structure

Launch config

Launch config type

has classes

Launch project

Application

JUnit

Runtime workbench
Eclipse launching: Dynamic behavior (JUnit)

Remote test runner

Launch project

Classpath

Eclipse JVM

Launch config

Launch config type

JUnit

Launch project

Test runner client

Test runner GUI

Socket

updates

Launched JVM
Continuous Testing
Static structure

Project

Source file

Builder

Launch config

Launch config type

Testing metadata

Launch project

Continuous testing

Java builder

CT builder

* when changes

* has classes
Continuous Testing
Dynamic behavior

Places we had difficulty

Testing multiple asynchronous units is hard

Problem icon selection hacks internal classes

Eclipse JVM
Suggestions for Eclipse

• JUnit integration:
  – Display results from multiple simultaneous test runs
  – Allow plug-ins to contribute prioritization

• Problems view:
  – More flexibility in icons

• Tools for testing asynchrony
  – It’s hard to create deterministic unit tests
Outline

• Continuous testing: defined and motivated
• Eclipse plug-in:
  – Design principles
  – User interface design: demo
  – Software design
• Next steps
Next steps:
split into individual plug-ins

Current plug-in

- Prioritize tests
- Associate launches with projects
- Run tests when project changes
- Create markers based on test failures
Next steps: feature enhancements

- Extend to Plug-in Development Environment
- Prioritize based on which methods, classes, etc. changed
- Use hot-swapping JVM to reduce start-up time
- Increase resolution: associate suite with package? class? method?
Next steps: test factoring

- **User-supplied test:**
  - Method Call
  - Expected Result

- **Factored tests:**
  - Method Call
  - Expected Result
  - Mock Object
  - Expected Result

* Saff, Ernst, PASTE 2004: Automatic mock object creation for test factoring
Further reading

• Model of developer behavior
  – Saff, Ernst, ISSRE 2003: Reducing wasted development time via continuous testing

• Controlled student experiment
  – Saff, Ernst, ISSTA 2004: An experimental evaluation of continuous testing during development

• Test factoring
  – Saff, Ernst, PASTE 2004: Automatic mock object creation for test factoring
Conclusion

- Plug-in is publicly available at http://pag.csail.mit.edu/~saff/continououstesting.html
- Many are using and enjoying continuous testing: give it a try!
- Eclipse was an excellent platform for meeting our design goals.
- Research and implementation continues