The Future of Programming Environments

Andreas Zeller
Saarland University

Past of programming environments

A Tool Set

Tools evolve
Tools integrate

Tools work together

Tools work together

Bugs
Changes
Error
Navigation
Chats
e-mail
Models
Specs
Code
Traces
Profiles
Tests
e-mail
Bugs
Effort
Navigation
Changes
Chats
Programmers who changed this function also changed…
Eclipse Preferences

Your task – extend Eclipse with a new preference:

Preference Code

Preferences are stored in the field $\textbf{fkeys}[]$:

```
public final OverlayPreferenceStore OverlayKey[] {
    new OverlayPreferenceStore.Overlay()
}
```

What else do you need to change?

Eclipse Code

<table>
<thead>
<tr>
<th>27,000 files</th>
<th>20,000 classes</th>
<th>200,000 methods</th>
</tr>
</thead>
</table>

12,000 non-Java

What else do you need to change?

funded by IBM Eclipse Innovation Grant
Mining Associations

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#42</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#752</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#9872</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#11386</td>
<td>fKeys[], initDefaults(), ...</td>
</tr>
<tr>
<td>#20814</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#30989</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#41999</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
<tr>
<td>#47423</td>
<td>fKeys[], initDefaults(), ..., plugin.properties, ...</td>
</tr>
</tbody>
</table>

Classical Data Mining

Classical association mining finds **all rules**:

- Helpful in understanding general patterns
- Requires high support thresholds
- Takes time to compute (3 days and more)
**Mining on Demand**

Alternative – mine only matching rules:

- Mine only rules related to the situation $\Sigma$, i.e. $\Sigma \Rightarrow X$

- Mine only rules which have a singleton as consequent, i.e. $\Sigma \Rightarrow \{x\}$

Average runtime of a query: 0.5 seconds

**Evaluation**

- The programmer has changed one entity. Can eROSE suggest related entities?

- Evaluation using last 1,000 transactions of eight open-source CVS repositories

- Training: all transactions before evaluation

**Precision vs. Recall**

What EROSE finds

What it should find

Correct prediction

False positives

False negatives

High precision = returned entities are relevant = few false positives

High recall = relevant entities are returned = few false negatives

**Results**

<table>
<thead>
<tr>
<th>Entities</th>
<th>Recall</th>
<th>Precision</th>
<th>Top 3</th>
<th>Recall</th>
<th>Precision</th>
<th>Top 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>0.15</td>
<td>0.26</td>
<td>0.53</td>
<td>0.17</td>
<td>0.26</td>
<td>0.54</td>
</tr>
<tr>
<td>GCC</td>
<td>0.28</td>
<td>0.39</td>
<td>0.89</td>
<td>0.44</td>
<td>0.42</td>
<td>0.87</td>
</tr>
<tr>
<td>GIMP</td>
<td>0.12</td>
<td>0.25</td>
<td>0.91</td>
<td>0.27</td>
<td>0.26</td>
<td>0.90</td>
</tr>
<tr>
<td>JBOSS</td>
<td>0.16</td>
<td>0.38</td>
<td>0.69</td>
<td>0.25</td>
<td>0.37</td>
<td>0.64</td>
</tr>
<tr>
<td>JEDIT</td>
<td>0.07</td>
<td>0.16</td>
<td>0.52</td>
<td>0.25</td>
<td>0.22</td>
<td>0.68</td>
</tr>
<tr>
<td>KOFFICE</td>
<td>0.08</td>
<td>0.17</td>
<td>0.46</td>
<td>0.24</td>
<td>0.26</td>
<td>0.67</td>
</tr>
<tr>
<td>POSTGRES</td>
<td>0.13</td>
<td>0.23</td>
<td>0.59</td>
<td>0.23</td>
<td>0.24</td>
<td>0.68</td>
</tr>
<tr>
<td>PYTHON</td>
<td>0.14</td>
<td>0.24</td>
<td>0.51</td>
<td>0.24</td>
<td>0.36</td>
<td>0.60</td>
</tr>
<tr>
<td>Average</td>
<td>0.15</td>
<td>0.26</td>
<td>0.64</td>
<td>0.26</td>
<td>0.30</td>
<td>0.70</td>
</tr>
</tbody>
</table>
Results

<table>
<thead>
<tr>
<th>Entities Files</th>
<th>Recall</th>
<th>Precision</th>
<th>Top 3</th>
<th>Recall</th>
<th>Precision</th>
<th>Top 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>0.15</td>
<td>0.26</td>
<td>0.53</td>
<td>0.17</td>
<td>0.26</td>
<td>0.54</td>
</tr>
<tr>
<td>GCC</td>
<td>0.28</td>
<td>0.39</td>
<td>0.89</td>
<td>0.44</td>
<td>0.42</td>
<td>0.87</td>
</tr>
<tr>
<td>GIMP</td>
<td>0.12</td>
<td>0.25</td>
<td>0.91</td>
<td>0.27</td>
<td>0.26</td>
<td>0.90</td>
</tr>
<tr>
<td>JBOSS</td>
<td>0.16</td>
<td>0.38</td>
<td>0.69</td>
<td>0.25</td>
<td>0.37</td>
<td>0.64</td>
</tr>
<tr>
<td>JEDIT</td>
<td>0.07</td>
<td>0.16</td>
<td>0.52</td>
<td>0.25</td>
<td>0.22</td>
<td>0.68</td>
</tr>
<tr>
<td>KOFFICE</td>
<td>0.08</td>
<td>0.17</td>
<td>0.46</td>
<td>0.24</td>
<td>0.26</td>
<td>0.67</td>
</tr>
<tr>
<td>POSTGRE</td>
<td>0.13</td>
<td>0.23</td>
<td>0.59</td>
<td>0.23</td>
<td>0.24</td>
<td>0.68</td>
</tr>
<tr>
<td>PYTHON</td>
<td>0.14</td>
<td>0.24</td>
<td>0.51</td>
<td>0.24</td>
<td>0.36</td>
<td>0.60</td>
</tr>
<tr>
<td>Average</td>
<td>0.15</td>
<td>0.26</td>
<td>0.64</td>
<td>0.26</td>
<td>0.30</td>
<td>0.70</td>
</tr>
</tbody>
</table>

- eROSE predicts 15% of all changed entities (files: 26%)
- In 64% of all transactions, eROSE's topmost three suggestions contain a changed entity (files: 70%)

Code Changes

From: Brian Kahne <bkahne@ibmoto.com>
To: DDD Bug Report Address <bug-ddd@gnu.org>
Subject: Problem with DDD and GDB 4.17

When using DDD with GDB 4.16, the run command correctly uses any prior command-line arguments, or the value of "set args". However, when I switched to GDB 4.17, this no longer worked: If I entered a run command in the console window, the prior command-line options would be lost. [....]
What was Changed

```
$ diff -r gdb-4.16 gdb-4.17
diff -r gdb-4.16/COPYING gdb-4.17/COPYING
5c5
< 675 Mass Ave, Cambridge, MA 02139, USA
   ---
> 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
282c282
< Appendix: How to Apply These Terms to Your New Programs
   ---
> How to Apply These Terms to Your New Programs

...and so on for 178,200 lines (8,721 locations)
```

General Plan

- Decompose diff into changes per location (= 8,721 individual changes)
- Apply subset of changes, using PATCH
- Reconstruct GDB; build errors mean unresolved test outcome
- Test GDB and return outcome
- Delta debugging narrows down difference

Isolating Changes

```
diff -r gdb-4.16/gdb/infcmd.c gdb-4.17/gdb/infcmd.c
1239c1278
< "Set arguments to give program being debugged when it is started."
   ---
> "Set argument list to give program being debugged when it is started."
```

- Documentation becomes GDB output
- DDD expects Arguments, but GDB outputs Argument list

The Failure Cause

```
• Result after 98 tests (= 1 hour)
```

```
```
```
```
```
```
```
```
Map bugs to code locations
Eclipse Imports

71% of all components importing compiler show a post-release defect

71% of all components importing compiler show a post-release defect

14% of all components importing UI show a post-release defect

Joint work with Adrian Schröter • Tom Zimmermann

Eclipse Bugs

Compiler code • Internals • Core functionality

GUI code • Standard Java classes • Help texts

Correlation with failure

import org.eclipse.jdt.internal.compiler.lookup.*;
import org.eclipse.jdt.internal.compiler.*;
import org.eclipse.jdt.internal.compiler.ast.*;
import org.eclipse.jdt.internal.compiler.util.*;
...
import org.eclipse.pde.core.*;
import org.eclipse.jface.wizard.*;
import org.eclipse.ui.*;

Correlation with success

import org.eclipse.jdt.internal.compiler.lookup.*;
import org.eclipse.jdt.internal.compiler.*;
import org.eclipse.jdt.internal.compiler.ast.*;
import org.eclipse.jdt.internal.compiler.util.*;
...
import org.eclipse.pde.core.*;
import org.eclipse.jface.wizard.*;
import org.eclipse.ui.*;
Predicting failure-prone packages

- Relate defect density to imports
- Base: Eclipse bug and version databases (Bugzilla, CVS)
- 36% of all packages had post-release defects
- Prediction using support vector machine

Where do bugs come from?

Is it the Developers?

Does experience matter?

Bug density correlates with experience!
How about Testing?

Does code coverage predict bug density?

Yes – the more tests, the more bugs!

History?

I found lots of bugs here. Will there be more?

Yes!

How about Metrics?

Do code metrics predict bug density?

Yes! (but only with history)

Problem Domains?

Do imports predict bug density?

Yes! (but only with history)
What makes code buggy in the first place?

Software Archives

- contain full record of project history
- maintained via programming environments
- automatic maintenance and access
- freely accessible in open source projects

"Which modules should I test most?"
“How long will it take to fix this bug?”

“This requirement is risky”

Empirical Studies

- Compares what we believe with what we observe
- Standard practice in modern science
- Recent addition to software engineering

Empirical SE

<table>
<thead>
<tr>
<th>Measure data</th>
<th>Build model that explains the data</th>
</tr>
</thead>
</table>

| Use model for predictions | Test *predictive power* |
Predicting Effort


Predicting Maintainability

Maintainability = $171 - 5.2 \ln(V) - 0.23V(G) - 16.2 \ln(L) + 50\sin\left(\sqrt{2.4C}\right)$

Size of vocabulary
McCabe complexity
Percentage of comment lines


Obtaining Data
Studies


Make this Actionable!

Assistance

Future environments will
• mine patterns from program + process
• apply rules to make predictions
• provide assistance in all development decisions
• adapt advice to project history

Web 2.0

“Road map”!
Future environments will
- mine patterns from program + process
- apply rules to make predictions
- provide assistance in all development decisions
- adapt advice to project history