Software Evolution Analysis and Visualization

Harald Gall
University of Zurich
Department of Informatics
http://seal.ifi.uzh.ch
"The study of products is vastly more important than the study of production, even for understanding production and its methods."

Karl Popper
On the study of products...
Goals & Questions

- What can we learn about
  - Software and its structure
  - Change impact and propagation
  - Developer contributions and efforts
  - Team structure and social networks
  - Change smells, trends and hot spots
  - Faults and defects
Mining Software Repositories...

- Code base
  - Which entities co-evolve?
  - Do code and comments co-evolve?
- Bugs and Changes
  - Who should fix this bug?
  - How long will it take to fix this bug?
  - Predicting bugs from cached bug history
  - When do changes induce fixes?
- Project and Process
  - Project memory for software development
- Software Expertise
  - Identifying expertise from changes and bug reports
Techniques

Standard Quality Characteristics

Software Quality Models
Software Process

Software Analysis
Reverse Engineering
Feature Analysis

Software Evolution Analysis

Code Duplication Analysis

Reengineering Patterns
Architecture Reflexion
Developer Patterns
Software Evolution Metrics
Software Visualization

Software Artifacts Analysis
Analyses & Visualizations

Changes and bug fixes
Developer tasks & patterns
Social networks
Release History Database

Related work
- Hipikat, Cubranic et al.
- softChange, German
- Kenyon, Bevan et al.
- s.e.a.l. Evolizer, Gall et al.
The *gestalt* of Fractal Figures

(a) One developer  (b) Few balanced developers  (c) One major developer  (d) Many balanced developers

Fractal Value \( = 1 - \sum_{a_i \in A} \left( \frac{nc(a_i)}{NC} \right)^2 \), with \( NC = \sum_{a_i \in A} nc(a_i) \)
How many developers per entity?
How many bugs per entity?

Marco D’Ambros, Michele Lanza and Harald C. Gall, Fractal Figures: Visualizing Development Effort for CVS Entities
Who should fix this bug?

- Apply machine learning algorithms to open bug repository
- Learn the kinds of reports that each developer resolves
- A classifier suggests developers who should resolve the bug
- Precision: 57% in Eclipse, 75% in Firefox

How long will it take to fix this bug?

- Automatically predicting the fixing effort, i.e., the person-hours spent on fixing an issue
- Effort data from JBoss project
- Quality of predictions
  - issues: close to actual effort
  - bugs: beating naive predictions

Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, Andreas Zeller, *How Long Will It Take to Fix This Bug?*
When do changes induce fixes?

- Fix-inducing changes
- Which change properties may lead to problems?
- How error-prone is my product?
- How can I filter out problematic changes?
- Can I improve guidance along related changes?

--> Fridays (Eclipse) or Sundays (Mozilla)

<p>| Table 5: Distribution of fixes and fix-inducing changes across day of week in ECLIPSE |</p>
<table>
<thead>
<tr>
<th>% of revisions</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\text{fix})$</td>
<td>18.4</td>
<td>20.9</td>
<td>20.0</td>
<td>22.3</td>
<td>24.0</td>
<td>14.7</td>
<td>16.9</td>
<td>20.8</td>
</tr>
<tr>
<td>$P(\text{bug})$</td>
<td>11.3</td>
<td>10.4</td>
<td>11.1</td>
<td>12.1</td>
<td>12.2</td>
<td>11.7</td>
<td>11.6</td>
<td>11.4</td>
</tr>
<tr>
<td>$P(\text{bug} \cap \text{fix})$</td>
<td>4.6</td>
<td>4.8</td>
<td>4.6</td>
<td>5.2</td>
<td>5.6</td>
<td>4.5</td>
<td>4.5</td>
<td>4.9</td>
</tr>
<tr>
<td>$P(\neg \text{bug} \cap \neg \text{fix})$</td>
<td>74.9</td>
<td>73.5</td>
<td>73.5</td>
<td>70.8</td>
<td>63.4</td>
<td>78.1</td>
<td>76.0</td>
<td>72.7</td>
</tr>
<tr>
<td>$P(\text{bug} \mid \text{fix})$</td>
<td>25.1</td>
<td>22.9</td>
<td>23.3</td>
<td>23.5</td>
<td>23.2</td>
<td>30.3</td>
<td>26.4</td>
<td>23.7</td>
</tr>
<tr>
<td>$P(\text{bug} \mid \neg \text{fix})$</td>
<td>8.2</td>
<td>7.1</td>
<td>8.1</td>
<td>8.8</td>
<td>8.7</td>
<td>8.4</td>
<td>8.6</td>
<td>8.1</td>
</tr>
</tbody>
</table>

<p>| Table 6: Distribution of fixes and fix-inducing changes across day of week in MOZILLA |</p>
<table>
<thead>
<tr>
<th>% of revisions</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\text{fix})$</td>
<td>42.5</td>
<td>46.5</td>
<td>49.7</td>
<td>45.9</td>
<td>48.4</td>
<td>50.2</td>
<td>61.1</td>
<td>48.5</td>
</tr>
<tr>
<td>$P(\text{bug})$</td>
<td>39.1</td>
<td>44.1</td>
<td>41.2</td>
<td>40.8</td>
<td>46.2</td>
<td>44.9</td>
<td>26.4</td>
<td>41.5</td>
</tr>
<tr>
<td>$P(\text{bug} \cap \text{fix})$</td>
<td>19.4</td>
<td>23.6</td>
<td>22.8</td>
<td>21.6</td>
<td>26.9</td>
<td>19.6</td>
<td>13.2</td>
<td>21.9</td>
</tr>
<tr>
<td>$P(\neg \text{bug} \cap \neg \text{fix})$</td>
<td>37.8</td>
<td>33.0</td>
<td>31.9</td>
<td>34.9</td>
<td>32.3</td>
<td>24.5</td>
<td>25.7</td>
<td>31.9</td>
</tr>
<tr>
<td>$P(\text{bug} \mid \text{fix})$</td>
<td>45.7</td>
<td>50.8</td>
<td>45.8</td>
<td>47.1</td>
<td>55.6</td>
<td>39.1</td>
<td>21.6</td>
<td>45.2</td>
</tr>
<tr>
<td>$P(\text{bug} \mid \neg \text{fix})$</td>
<td>34.1</td>
<td>38.3</td>
<td>36.7</td>
<td>35.5</td>
<td>37.3</td>
<td>37.3</td>
<td>33.9</td>
<td>38.1</td>
</tr>
</tbody>
</table>

Code Ownership & Co-Evolution
Who is the code owner?

CVS check-in analysis combined with Formal Concept Analysis (FCA)

*ownership = 30% of revisions*
Which entities co-evolve?

Software Evolution Metrics

Fan-in
- invoke
- access

Class/module metrics
- files, directories,
- packages, ...
- global variables,
- NOM, NOA, ...

Change dependencies
- change couplings
- bugs, issues

Fan-out
- invoke
- access

Martin Pinzger, Harald C. Gall, Michael Fischer, and Michele Lanza, *Visualizing Multiple Evolution Metrics*
Mozilla Module DOM: 0.92 -> 1.7
Multiple Evolution Metrics

Kiviat graph:
26 metrics
7 Mozilla modules
7 subsequent releases
ChangeDistilling

Source Code Change Extraction
Change Analysis

- Current change history analysis rely on versioning systems (e.g., CVS)
- Extracting source code changes by means of text diffs has problems
  - determine enclosing entity (e.g., method)
  - kind of statement which changed (e.g., return statement)
  - kind of change (i.e., insert, delete, move, update)
Change Analysis

CVS diff

```
<    if (d != null) {
<        d.foo();
<        d.bar();
<    }
---
>    d.foo();
>    d.bar();
```

- 3 Body changes
- 2 Statement parent changes
- 1 Statement delete
- Change significance?

CVS log: “lines: +2 -4”
Change Distilling

- Identifying change types and change patterns
- Eliminate mass changes and other noise
Examples of Change Types

- Classification of 35 change types

<table>
<thead>
<tr>
<th>Body part changes:</th>
<th>significance:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Object State</td>
<td>low</td>
</tr>
<tr>
<td>Condition Expression Change</td>
<td>medium</td>
</tr>
<tr>
<td>Removed Functionality</td>
<td>crucial</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration part changes:</th>
<th>significance:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Modifier Delete</td>
<td>low</td>
</tr>
<tr>
<td>Parameter Renaming</td>
<td>medium</td>
</tr>
<tr>
<td>Return Type Update</td>
<td>crucial</td>
</tr>
</tbody>
</table>
Potential of Change Type Analysis

- Stability of interfaces
- Change impact
- Code and Comments
- Changes due to bug fixes
- Many or significant changes
Which changes are significant?
Developer Networks

Communication Structures in Software Teams
What are the developer networks?

- Conway’s law
- Inter-team collaboration
- Ownership changes
- Key personalities in social networks
  - connectors vs. communicators
  - gatekeepers, influencers, innovators, leaders and communicators as trendsetters
- Information for project manager vs. newcomer
Integration of Data Sources

- iQuest
  - MHonArc
    - html
  - Mails
- Importer
  - Bugzilla
    - xml/html
  - Bugs
  - CVS
  - ViewVC
- Ownership Calculator
  - File Size Calculator
- Interaction Paths
  - Person Matching
  - Person Finder → Allocation
  - Person Matching
  - Possible Follow-Ups
    - Follow-Ups
  - Project Consolidation
  - Person Matching

University of Zurich
Department of Informatics
seal.ifi.uzh.ch
Scenario: newcomer Kevin

(a) April 2004

(b) April 2004

(c) June 2004

(d) June 2004
Scenario: key person Rafael leaving

(a) April 2005

(b) July 2005

(c) July 2005

(d) November 2005
Scenario: distributed teams

Sub-Group @us.ibm.com

Sub-Group @ch.imb.com
SNA Cockpit

<table>
<thead>
<tr>
<th>File</th>
<th>View</th>
<th>Graph Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
33
```
Evolizer

A platform for harvesting and provisioning of software evolution data
The Architecture of Evolizer

- Plug-in architecture
- Layers
  - Repositories
  - Data importers
  - Data integrators
  - Data providers
  - Data consumers
Data Models in Evolizer

- Data Models provide an interface to the information harvested from a software repository
  - One model per repository
  - Models can integrate other models
Version Control Model
Bridging the Gap
Bridging the Gap
Evolizer Tools

- ChangeDistiller: change types and significance
- ArchView: evolution metrics
- SNA Cockpit: developer networks
- Evolution Browser
- Comment Analyzer: code and comments
- Clone Evolution
Conclusions
Résumé

- Analyzing *software evolution* is a multi-source/-view/-dimension/-stakeholder challenge
  - Technical: resides in modeling and handling various kinds of information
  - Conceptual: answering interesting questions and presenting the results (visually)
- Mining software repositories has been embraced by both the *software evolution* and the *empirical software engineering* community
- Social networks are a key
Developers, developers, developers
References

- Anvik, J., Hiew, L., and Murphy, G. C., Who should fix this bug?. In Proceeding of the 28th international Conference on Software Engineering, Shanghai, China, May 2006
- John Anvik, Gail C. Murphy, Determining Implementation Expertise from Bug Reports, Proceedings of the Fourth International Workshop on Mining Software Repositories, May 2007
- Davor Cubranic, Gail C. Murphy, Janice Singer, Kellogg S. Booth, Hipikat: A Project Memory for Software Development, IEEE Transactions on Software Engineering, v.31 n.6, p.446-465, June 2005
References /2

- Gursimran Singh Walia, Jeffrey C Carver, Nachiappan Nagappan, The Effect of the Number of Inspectors on the Defect Estimates Produced by Capture-Recapture Models, ICSE 2008 (to appear)
- J. Zliwerski, T. Zimmermann, and A. Zeller, When Do Changes Induce Fixes?, Proc. of Int'l Workshop on Mining Software Repositories (MSR 2005), Saint Louis, Missouri, USA, May 2005
- Thomas Zimmermann, Nachiappan Nagappan, Predicting Defects using Social Network Analysis on Dependency Graphs, ICSE 2008 (to appear)