What Can Changes Tell about Software Processes

Barbara Russo, Maximilian Steff

Free University of Bolzano-Bozen, Italy

WeTSOM June, 3rd, 2014
The problem

- Investigate code change histories to understand software processes
- Histories of code changes reflect the manner in which process activities are carried out
Starting point

- Activities made to complete a developer’s task might not occur in close temporal proximity
- They are related by what in common they change
The method

- We represent software development processes with colored commit graphs and study histories in these graphs.
- We color graphs by assigning semantics of different nature to commits (e.g., bug fix commit)
- We color graphs with either commit labels or large architectural or size change.
- To color commits by large architectural changes, we introduce a measure of architectural change.
The problem

We study the order of occurrence of different colors in commit graphs to understand the precedence among them:

▶ Do large architectural changes precedes or follow large churn?
▶ Do bug fixes precede large architectural change or churn?
▶ Do large architectural changes precede bug fixes?

The Proof of concept

We extend a non-parametric precedence score to histories in commit graphs
Commit Graph

It is a directed graph whose nodes are commits and edges are determined by files changed in commits.
Commit Histories

- CG is made of a set of commit histories
- In this research: progenies and ancestry
- Progeny is the set of children and grandchildren of C
- Ancestry is the set of parents and grandparents of C
Coloring commits

We color commits as:

- Large churn (churn jump)
- Large architectural change (architectural jump)
- Bug fix
- Combination of them
- None of them

Jump: measures off one standard deviation
Architectural change

The architecture at a commit:

- Node: class and interface
- Node label: class names
- Link: Inheritance, association and use dependency
Architectural change

The Hido-Kashima kernel (2009):
- Hashing the node label to incorporate neighbour labels (XOR and ROT): bit label graph
- Radix sort on obtained bit labels
- Kernel: Jacard index of two architectures

Measure of architectural change, Nakamura, & Basili, 2005,

\[ L(A) = \frac{d(A_f, A) - d(A, A_0)}{d(A_f, A) + d(A, A_0)} \]
Gamma score for commit histories

An extension of the precedence gamma score to graph histories

\[ I_{A,B} = \frac{P - Q}{P + Q} \]

where \( P \) (\( Q \)) is the number of color-A (B) commits with at least one color-B (A) commit in their progeny
Proof of concept

A case study of Spring Framework and Eclipse JDT
Eclipse JDT

Architectural changes do not follow bug fixes as churn does
Spring

Architectural changes and churn have a similar trend and opposite to number of bug fixes.
Change impact

At a first sight, one would imply that:

- In Eclipse, architectural changes are not performed for fixing bugs and
- In Spring, changes are not made for fixing bugs
Change impact

This is implication might not be correct:

- The two plots can only illustrate trends on individual commits. They do not show change dependencies.
- Changes in commits can originate from preceding changes or affect future changes.
Gamma score

We use the extended gamma score to progenies and ancestries
Progeny precedence analysis

<table>
<thead>
<tr>
<th>Color A</th>
<th>Color B</th>
<th>Spring</th>
<th>JDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>churn jump</td>
<td>arch jump</td>
<td>0.15</td>
<td>0.09</td>
</tr>
<tr>
<td>bug fix</td>
<td>churn jump</td>
<td>0.23</td>
<td>0.78</td>
</tr>
<tr>
<td>bug fix</td>
<td>arch jump</td>
<td>0.22</td>
<td>0.71</td>
</tr>
<tr>
<td>bug fix</td>
<td>arch &amp; churn jump</td>
<td>0.10</td>
<td>0.84</td>
</tr>
<tr>
<td>arch bug fix</td>
<td>churn jump</td>
<td>-0.01</td>
<td>0.14</td>
</tr>
<tr>
<td>arch bug fix</td>
<td>arch jump</td>
<td>0.04</td>
<td>0.06</td>
</tr>
<tr>
<td>arch bug fix</td>
<td>arch &amp; churn jump</td>
<td>-0.18</td>
<td>0.39</td>
</tr>
<tr>
<td>churn bug fix</td>
<td>churn jump</td>
<td>0.13</td>
<td>0.76</td>
</tr>
<tr>
<td>churn bug fix</td>
<td>arch jump</td>
<td>0.07</td>
<td>0.68</td>
</tr>
<tr>
<td>churn bug fix</td>
<td>arch &amp; churn jump</td>
<td>-0.20</td>
<td>0.81</td>
</tr>
</tbody>
</table>
Interpretation

- Architectural and churn changes are not completely synchronous in both systems.
- Follow-up effect (e.g., refactoring): the probability that jumps occur in progenies of bug fixes is statistically higher than they occur in ancestries (especially for Eclipse JDT).
- The two systems are definitely different in fixing bugs that do not require architectural changes (last row).
- In Eclipse JDT, bug fix commits that do not have architectural changes likelier precedes large changes in both architecture and churn.
Conclusions

- We represent software development processes with colored commit graphs and study histories of changes in these graphs.
- The graph gives an holistic representation of the development process together with the evolution of the product.
- Graph theory can be exploited to examine patterns in the commit graph substructures (e.g., to determine activities performed for specific bugs).
Reflections

- A correct interpretation of the differences or a causality analysis requires a deeper investigation of the type of systems and their maintenance process.
- Bug fixes can accomplish other tasks.
  - Herzig and Zeller, (2013) found that 16.6% of all source files are incorrectly associated to bug fixes; future work will explore the Herzig and Zeller algorithm on commit histories.
Reflections

- HK hashing works fine until the module label is modified during the change process (in our case, this does not occur).
- Commit graphs can be colored in different ways depending on the information available in the repositories.
Thank you!
Questions?
Related Work

Research focus on

- Impact analysis
- Change classification
- Change measurement
Impact analysis

Typically studied for causality of changes or prediction of other code measures. Examples:

- Change impact graph to detect and visualize the propagation of function changes and help developers to localize bugs (German et al., 2009)
- Local modification to classes’ dependencies to investigate change causality (Herzif and Zeller, 2011)
  - It is affected by how one interprets the local changes and how frequently structural snapshots are captured.
  - The major changes keep occurring.
  - Herraiz et. al. challenged this result on a different data sets
Change classification

Literature mainly focuses on syntactical types or types determined by the task they accomplish.

- Logical coupling as change pattern similarity of files over releases (Gall et al., 1998)
- Number of linked developers working on a file (Cataldo et al., 2009)
Change measurement

Change measures are defined at different granularity levels (e.g., files, classes, methods, or commit, versions, releases).

- The majority focus on churn
- Other measures based on syntactical dependencies (Nakamura and Basili, 2005)