Mining Software Repositories for Traceability Links

Huzefa Kagdi
Jonathan I. Maletic
Bonita Sharif
Research Goal

Uncover/discover traceability links between source code and other types of software artifacts using a Mining Software Repositories (MSR) type of approach
Merits of a Change-Based Approach

- Actual changes to artifacts rather than estimations
- The practice of commits and commit messages embody part of the developer’s knowledge and experience
- We feel that version history may contain domain-specific “hidden” links that program-analysis methods may fail to uncover
A KDE Subversion Commit

Metadata

```
Changed paths:
  M /trunk/KDE/kdegames/ktron/ktron.cpp
  M /trunk/KDE/kdegames/ktron/main.cpp
move scope
```

Differences

```
ktron.cpp r472030:r472031

--- ktron.cpp (revision 472030)
+++ ktron.cpp (revision 472031)
@@ -27,7 +27,6 @@
   #include <kmessagebox.h>
   #include <kaction.h>
   #include <kstdgameaction.h>
-#include <kapplication.h>
   #include <kstatusbar.h>
  
  // Settings
```
Traceability of Interest

- Uncovering traceability between source code and other artifacts:
  - User manuals, *e.g.*, HTML and XML/docbook
  - Build management, *e.g.*, automake and makefile
  - HowTo guides, *e.g.*, FAQs
  - Software distribution, *e.g.*, ChangeLogs and README
  - Progress monitoring, *e.g.*, TODO and STATUS

- *Software Informalisms* in open source systems [Scacchi'02]
Examples: Traceability Links

- \{kalzium.cpp, pse.cpp\} → \{index.docbook\}

- \{kalziumtip.cpp\} → \{detailinfodlg.cpp\} → \{Makefile.am\} → \{kalzium.cpp, kalzium.h\}

- \{TODO\} → \{pse.cpp\}

Notations
- \{..,..,..\} files in the same change-set/revision
- \(\rightarrow\) change-set commit order
Refining the Problem

- Is the presence of different types of documents in a single change-set sufficient to infer traceability links between them?

- How do we account for related documents with potential traceability links committed in a series of change-sets?
Hypothesis

If a group of specific artifacts (of different types) are co-changed together, *frequently*, then there is a high probability that they have a traceability link between them.
Our Approach

- Heuristics are used to group potentially related change-sets

- Sequential-pattern mining [Agrawal'92] analyzes related change-sets to mine for highly frequent co-occurring changed files - change patterns

- Change patterns are then analyzed to uncover patterns that contain source code files and other types of files - traceability patterns
Heuristics based on Metadata

- **Time Interval**
  - Change-sets committed in a given time duration are placed in a single group

- **Committer**
  - Change-sets committed by a given committer are placed in a single group

- **Committer + Time Interval**
  - Change-sets committed by the same committer within the same time interval are placed in a single group
Mining Ordered Patterns

- Related change-sets are grouped based on a grouping heuristic.

- Ordering of change-sets in a group is based on their revision numbers.
  - Change-sets with larger revision numbers occur after those with lesser revision numbers.

- Sequential Pattern Discovery Algorithm (SPADE) [Zaki'02] is applied to mine frequent partial sequences of changed files from a set of groups [Kagdi'06].
  - Each group of related change-sets forms an transaction.
  - Each change-set forms an event.
Examples: KDE Traceability Patterns

- Found in five days
  - \{kalzium.cpp, pse.cpp\} → \{index.docbook\}

- Repeated by five different developers
  - \{kalziumtip.cpp\} → \{detailinfodlg.cpp\} → \{Makefile.am\} → \{kalzium.cpp, kalzium.h\}

- Ten different developer-day combinations
  - \{TODO\} → \{pse.cpp\}
Evaluation

- KDE (K Desktop Environment) is used as a subject system
- First mine a portion of the version history for traceability patterns - training-set
- Next mine a later part of the version history for traceability patterns - evaluation-set
- Assess how accurately the candidates generated from the training-set predict changes that occur in the evaluation-set
## Change and Traceability Patterns

### Training-set

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>CP</th>
<th>TP</th>
<th>TP/CP%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>5,839</td>
<td>1,851</td>
<td>37.10</td>
</tr>
<tr>
<td>Committer</td>
<td>718</td>
<td>54</td>
<td>7.52</td>
</tr>
<tr>
<td>CommitterDay</td>
<td>2,372</td>
<td>277</td>
<td>11.68</td>
</tr>
</tbody>
</table>

### Evaluation-set

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>CP</th>
<th>TP</th>
<th>TP/CP%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>1112</td>
<td>143</td>
<td>12.86</td>
</tr>
<tr>
<td>Committer</td>
<td>304</td>
<td>26</td>
<td>8.55</td>
</tr>
<tr>
<td>CommitterDay</td>
<td>835</td>
<td>8</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Assessment Metrics

- **Coverage**
  - The percentage of traceability patterns in the evaluation-set for which there is at least one candidate (correct or incorrect) recommended from the training-set

- **Recall**
  - The percentage of traceability patterns in the evaluation-set that are correctly recommended from the training-set

- **Precision**
  - The percentage of correct candidates suggested after a change in a change-set of a covered pattern
  - Minimum, maximum, and average for the evaluation-set
## Coverage, Recall, and Precision Results

<table>
<thead>
<tr>
<th>Heuristic Queries</th>
<th>Coverage (%)</th>
<th>Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>44</td>
<td>8</td>
</tr>
<tr>
<td>Committer</td>
<td>57</td>
<td>11</td>
</tr>
<tr>
<td>CommitterDay</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heuristic Queries</th>
<th>Precision (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>Day</td>
<td>50</td>
</tr>
<tr>
<td>Committer</td>
<td>55</td>
</tr>
<tr>
<td>CommitterDay</td>
<td>100</td>
</tr>
</tbody>
</table>
Related Work

- Traceability between artifacts in bug repositories and source code [Canfora’06, Cubranic’05, Sliwerski’05]

- Evolutionary couplings of source code [Gall’98, Zimmermann’04, Yang’04, German’04, Kagdi’06]

- Approaches typically analyzing a single software version for recovering traceability links e.g., Antoniol, Cleland-Huang, Egyed, Hayes, Marcus, Spanoudakis and Zisman
Conclusions

- A heuristic based approach that uses frequent-pattern mining for mining traceability links was presented

- Showed on KDE that these traceability links can be uncovered and used with high precision

- This work compounded with existing approaches expands the area of traceability link recovery
Future Work

- Additional heuristics for grouping related change-sets such as textual similarity of commit messages

- Fine-grained traceability link (e.g., class and method levels) using srcML

- Integrating our tools into Subversion
MSR Survey

Kagdi, Collard, Maletic

*JSME* March/April 2007

MSR in the context of software maintenance – 56 pages