Linking Source Code to Untangled Change Intents

Xiaoyu Liu¹  LiGuo Huang¹  Chuanyi Li¹,²  Vincent Ng³

¹ Dept. of Computer Science and Engineering, Southern Methodist University
² State Key Laboratory for Novel Software Technology, Nanjing University
³ Human Language Technology Research Institute, University of Texas at Dallas
Motivating Example

1. read commit message sentences

1. read relevant issue report

identify link

Issue report: Issue 13
Title:
master compilation

Description:
Can you explain how to compile the master repository?
When I try to do it I get
[INFO] --- maven-source-plugin:2.1.2:jar (default) @
gmail4j --
......

Comment 1:
I just removed the plugin. I'll try to login and let you
know. Can I ask here another question or do I need to
open another issue? It's a question on api usage...
Key Challenges

Change intents are tangled together in commit message

Manually link changed source code to change intents take lots of human efforts
Key Challenges

Change intents are tangled together in commit message
  • **Solution**: Untangle change intents

Manually link changed source code to change intents take lots of human efforts
  • **Solution**: An automated approach
Existing Software Artifacts Linking Approaches

Manual approaches

- **Problem:** Time-consuming, labor-intensive, and requires a great deal of experience

Information Retrieval (IR)-based approaches

- Use a IR-based model: measures similarity between changed source code and untangled change intent
- **Problem:** Changed entities extracted from source code could be very different from what is described in commit messages and other related software documents
Propose approaches to address the task of linking changed source code to untangled change intent at the sentence level
Our Approaches

**AutoCILink-P**
- Automatically identify links by using **manually defined patterns**

**AutoCILink-ML**
- Automatically identify links by applying **supervised learning**
Pattern-based Link Identification System (AutoCILink-P)

Untangling change intents

Issue 13: quick fix. # moved the declaration of StringBuilder inside toString() method. # added missing javadoc to provide long startTime instance variable

• Issue 13: quick fix.
• # moved the declaration of StringBuilder inside toString() method.
• # added missing javadoc to provide long startTime instance variable

Text Preprocessing

Link identification using regular expression

If no link

Link identification using vocabulary similarity
Pattern-based Link Identification System (AutoCILink-P)

Enriched untangled change intents

Commit: ae66810
Commit message:
Issue 13: quick fix. # moved the declaration of StringBuilder
toString inside toString() method. # added missing javadoc
to private long startTime instance variable.

Issue report: Issue 13
Title:
master compilation

Description:
Can you explain how to compile the master repository?
When I try to do it I get

Link identification using regular expression

If no link

Link identification using vocabulary similarity
Pattern-based Link Identification System (AutoCILink-P)

Extracting terms from enriched change intents and changed entities from changed source code

“# moved the declaration of StringBuilder inside toString() method.”

Terms: “declaration”, “string”, “builders”, ...

Changed file:
SimulationRun.java

Diff: ......
+ eventHandlerMap.put( SimulationEvent.-
  TYPE_ACQUIRE_JOB_NOTIFICATION_EVENT, new AcquireJobNotificationEventHandler(job-Executor) );
......

Changed entities:
“event”, “handler”, “map”, “simulation”, “type”, “job”, ...

1. Text Preprocessing
2. Link identification using regular expression
3. Link identification using vocabulary similarity

If no link
Pattern-based Link Identification System (AutoCILink-P)

Generating regular expressions

- \.*\s*<verb>\(.*?\)<entity>\(.*\)
- \.*\s*<entity>\(.*?\)<verb>\(.*\)

<verb>: collected from change types (e.g., move)
<entity>: noun words in changed source code identifiers, comments or string literals (e.g., builder)
Generating regular expressions

\[^.*\s*<verb>(.*?)<entity>(.*)\]

\[^.*\s*<entity>(.*?)<verb>(.*)\]

“moved the declaration of stringbuilder inside toString() method”
Pattern-based Link Identification System (AutoCILink-P)

Vocabulary similarity between enriched untangled change intent terms and changed source code entities

\[ \text{sim}(i, c) = \frac{\sum_{t \in V, e \in V} \omega(t, i, V) \times \omega(e, c, V)}{\sqrt{\sum_{t \in V} \omega(t, i, V)^2} \times \sqrt{\sum_{e \in V} \omega(e, c, V)^2}} \]

\[ \text{m}(d, c) = \frac{\sum_{t \in V, e \in V} \omega(t, d, V) \times \omega(e, c, V)}{\sqrt{\sum_{t \in V} \omega(t, d, V)^2} \times \sqrt{\sum_{e \in V} \omega(e, c, V)^2}} \]

\[ \text{sim}(\text{Link}(i, c)) = \max(\text{sim}(i, c), \bigcup_{x=1}^{n} \text{sim}(d_x, c)) \]

**Text Preprocessing**

1. Link identification using regular expression

2. If no link

3. Link identification using vocabulary similarity
AutoCILink-\textit{P}'s Key Weakness

Regular expressions are not always precise

\begin{verbatim}
Commit: 18eb05d
Commit message:
... Implementation of jobs for simulation process engine.
... Performance can be increased by increasing
    maxWaitTime for job executor too. ...

Changed file:
SimulationRun.java
Diff: ......
+ eventHandlerMap.put( SimulationEvent.-
    TYPE_ACQUIRE_JOB_NOTIFICATION_EVENT, new
    AcquireJobNotificationEventHandler( job-Executor ) );
    ......

Changed file:
SimulationAcquireJobsRunnable.java
Diff: ......
+ millisToWait *= waitIncreaseFactor;
+ if (millisToWait > maxWait) {
+    millisToWait = maxWait; ......
\end{verbatim}
AutoCILink-P’s Key Weakness

Impreciseness of regular expressions

- **Solution**: employs a learning-based link classification system to weigh the importance of matched regular expressions.
Create training instances for each changed source code and enriched untangled change intent pair in the training dataset.

Each instance annotated as linked or not linked depending on whether there is a link.

Each instance is represented using 6 features.
Type 1: Regular Expression Features encode the presence or absence of the regular expression in the training set.

Type 2-4: Three types of Vocabulary Features
Three Types of Vocabulary Features

Type 2: Vocabulary Pair Features

Type 3: Vocabulary Similarity Features

Type 4: Term Unmatched Features

"Implementation of jobs for simulation process engine"

Changed file: SimulationRun.java
Diff: ......
+ eventHandlerMap.put( SimulationEvent.TYPE_ACQUIRE_JOB_NOTIFICATION_EVT,
  AcquireJobNotificationEventHandler(job);
......

(process, job)
Three Types of Vocabulary Features

Type 2: Vocabulary Pair Features

Type 3: Vocabulary Similarity Features

Encode the vocabulary similarity scores between an enriched untangled change intent and changed source code

Type 4: Term Unmatched Features
Three Types of Vocabulary Features

Type 2: Vocabulary Pair Features

Type 3: Vocabulary Similarity Features

Type 4: Term Unmatched Features


Entities: “job”, “notif”, “time”, “event”, “executor”, “handle”, …

3 out of 7 (42.9%) matched, so the percentage of new terms unmatched in changed entities (1-42.9%=57.1%) falls in the range of [50%, 60%)
Features

**Type 5: Code Import Features** encode the percentage of terms unmatched in each imported code module changed entities.

**Type 6: Untangled Change Intent Count Features** encode the number of untangled change intents in corresponding commit message.
Supervised learning-based link classification system (AutoCILink-ML) uses the SVM learning algorithm to train a link classifier.
Supervised learning-based link classification system (AutoCILink-ML)

1. Create Training Instances
2. Train Link Classifier
3. Classify Linked or Not-linked in Test Set

AutoCILink-ML determines whether the given untangled change intent and changed source code file are "linked" or "not linked".
Empirical Evaluation

Datasets: Open Source Java Projects from GitHub

19 Projects from GitHub, 572 untangled change intents, 2739 changed source code files, 1025 “linked” code-intent pairs (70.1%), and 1288 “not linked” code-intent pairs (29.9%).

Five-Fold Cross Validation

Metrics

Recall, Precision, F1-score

Accuracy: percentage of code-intent pairs correctly classified

Baseline Systems

IR-based Systems: LSI, VSM, Association-based

Majority Classifier: a greedy approach simply classify into the majority class (i.e., “linked”)

Untangled Intent Count Classifier: classify using threshold based on untangled change intents count
**Overall Performance**

1. How effective is AutoCILink in linking changed code to untangled change intents?

<table>
<thead>
<tr>
<th></th>
<th>Linked, F1</th>
<th>Not linked, F1</th>
<th>Avg. Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Majority</td>
<td>LSI</td>
<td>VSM</td>
</tr>
<tr>
<td></td>
<td>82.4</td>
<td>73.2</td>
<td>80.4</td>
</tr>
<tr>
<td></td>
<td>78.4</td>
<td>36.2</td>
<td>70.4</td>
</tr>
<tr>
<td></td>
<td>62.4</td>
<td>66.5</td>
<td>64.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RQ2. Which system is more accurate in linking changed code to untangled change intents, AutoCIILink-ML or AutoCIILink-P.
Which feature types have the largest impact on the performance of AutoCILink-ML?

The most important features are *Untangled Change Intent Count Features* and *Term Unmatched Features*, as their removal results in a 18.1% and 12.2% drop in accuracy, respectively.