Constructing Software Knowledge Graph from Software Text

Sirui Li (u5831882)
Supervisor: Dr. Zhenchang Xing
Team Member: Hongwei Li, Jiamou Sun
COMP8755 Individual Computing Project
Semester 1, 2018
What is Software Text?

- API documentation
- Stack Overflow Post
- Bug Report
Motivation (Formative Study)

• Accessibility issue of API usage directives (API caveats)

Summary: Our formative study shows that many programming issues could actually be avoided if developers were aware of relevant API caveats in API documentation. Unfortunately, developers mostly discover API caveats post mortem after something wrong happened, rather than bewaring of the API caveats beforehand to avoid the mistakes in the first place.
Goal

To tackle the accessibility of API caveats by mining an API caveats knowledge graph from multiple sources of API documentation.
Approach

1. Preprocess input documentation
2. Build an API skeleton graph from semi-structured API reference documentation
3. Extract API caveat sentences from API textual descriptions
4. Construct the API-caveats knowledge graph by linking API caveat sentences to relevant APIs
Approach (Preprocess input documentation)

• Input documentation: API reference documentation and API tutorials
e.g. Android API reference, Android Developer Guides

• Software-specific tokenizer (Ye et al) + Stanford CoreNLP
e.g. “void setOnBufferAvailableListener(Allocation.OnBufferAvailableListener
callback) Set a notification handler for USAGE_IO_INPUT”
Approach (Build an API skeleton graph )

• From API reference documentation
  1. Entities: classes, interfaces, fields, methods and parameters.
  2. Relations: containment, inheritance/implementation, field data type, method return type, method parameter type, and method-thrown-exception.

```
public class ThreadGroup
extends Object implements Thread.UncaughtExceptionHandler

java.lang.Object
  \rightarrow java.lang.ThreadGroup
```
Each pattern is defined by a regular expression:
1. Error/Exception: You must store a strong reference to the listener, otherwise it will be susceptible to garbage collection.
2. Recommendation: you are better off using JobIntentService, which uses jobs instead of services ...
3. Alternative: You must call release() when you are done using the camera, otherwise it will remain locked and be unavailable to applications.
4. Imperative: Do not confuse this method with activity lifecycle callbacks such as onPause(...) 
5. Note: Note: For all activities, you must declare your intent filters in the manifest file.
6. Conditional: When using a subclass of AsyncTask to run network operations, you must be cautious...
7. Temporal: This may be null if the service is being restarted after its process has gone away.
8. Affirmative: The identifier does not have to be unique in View, but it should be positive.
9. Negative: Any activities that are not declared there will not be seen by the system and will never be run.
10. Emphasis: Only objects running on the UI thread have access to other objects on that thread.
Approach (Build API Caveats Knowledge Graph)

1. Co-reference Resolution
   a) Stanford CoreNLP
   b) Declaration-based Resolution: “this method”, “this class”, etc.
      e.g. Activity.onActionModeStarted: “Activity subclasses overriding this method should call the superclass implementation. If you override this method you must call through to the superclass implementation.”

2. Linking API Caveats to API Entities
   a) Hyperlink based
      e.g. “It throws ActivityNotFoundException if ...” is hyperlinked to the reference page of “ActivityNotFoundException”, we can then link this sentence to the “ActivityNotFoundException”
   b) Declaration based
      e.g. View.setId(int): “Do not pass a resource ID” and “The identifier should be a positive number”.
   c) Open linking (Subject-Verb-Object triples)
      e.g. “You must call release() ...” links with “release()”
Evaluation

• RQ1: What is the abundance of different subcategories of API caveats in API documentation?
• RQ2: Can our approach accurately extract API caveat sentences, re-solve co-references in these sentences, and link API caveat sentences to API entities?
• RQ3: Can our API caveats knowledge graph and API caveats search improve the accessibility of API caveats, compared with traditional documentation search?
Evaluation (RQ1: The Abundance of API Caveat)

- ACFOR scale (Abundant, Common, Frequent, Occasional, Rare) \(^1\)

<table>
<thead>
<tr>
<th>Subcategory</th>
<th>Numbers</th>
<th>Frequency</th>
<th>Average Abundance Frequency</th>
<th>ACFOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error/Exception</td>
<td>11,973</td>
<td>0.045</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Recommendation</td>
<td>19,209</td>
<td>0.072</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Alternative</td>
<td>4,032</td>
<td>0.015</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Imperative</td>
<td>6,410</td>
<td>0.024</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Note</td>
<td>8,183</td>
<td>0.031</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Conditional</td>
<td>70,404</td>
<td>0.263</td>
<td>0.100</td>
<td>A</td>
</tr>
<tr>
<td>Temporal</td>
<td>8,479</td>
<td>0.032</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Affirmative</td>
<td>61,952</td>
<td>0.231</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Negative</td>
<td>36,136</td>
<td>0.135</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Emphasis</td>
<td>41,113</td>
<td>0.153</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Total:</td>
<td>267,891</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conditional, Affirmative, Negative and Emphasis API caveats are dominant in our knowledge graph. The fact that most of API caveats do not have explicit caveat indicators could help to explain why API caveats are hard to notice in the API descriptions, especially the lengthy ones.

\(^1\)http://en.wikipedia.org/wiki/Abundance_(ecology).
Evaluation (RQ2: The Quality of API Caveats Knowledge Graph)

• **Accuracy of Extracting API Caveat Sentences**
  a) Cohen’s kappa metric: 88% -> almost perfect agreement
  b) Accuracy: 100%

• **Accuracy of Co-reference Resolution**
  a) Cohen’s kappa metric: 97% -> almost perfect agreement
  b) Accuracy: 74.22%

• **Accuracy of Caveat-Sentence-API Linking**
  a) Cohen’s kappa metric: 99.74% (declaration-based), 98.70% (open linking) -> almost perfect agreement
  b) Accuracy: 99.48% (declaration-based), 98.44% (open linking)
Evaluation (RQ3: The Improvement of API Caveats Accessibility)

- This means that apps targeting Android 7.0 (API level 24) and higher cannot share private files by name, and attempts to share a "file://" URI will result in a FileUriExposedException to be thrown.
- Therefore, attempts to pass a file:// URI trigger a FileUriExposedException.
- If an intent containing a file URI leaves developer's app, the app fails with a FileUriExposedException exception.
- For more recent apps targeting Android 7.0 (API level 24) and higher, passing a file:// URI across a package boundary causes a FileUriExposedException.
- FileUriExposedException will be thrown to applications.
- FileUriExposedException The exception that is thrown when an application exposes a file:// Uri to another app.
Evaluation (RQ3: The Improvement of API Caveats Accessibility)

Participants: 12 third- and fourth-year under-graduate students from our school (None of them have Android development experience.)

1. Control Group: use Google search engine to search for API caveats on the Android Developers website
2. Experimental Group: use our search tool to search for API caveats in the Android API caveats knowledge graph we construct.

Experiment Procedure: An simple application records their answers, the time and the difficulty of the question and participants’ confidence in the submitted answer using the 5-point Likert scale.

Data Analysis: question-completion-time statistics and the question-difficulty and answer-confidence ratings of the two groups
Evaluation (RQ3: The Improvement of API Caveats Accessibility)

Table 6.4: Performance of Control Group versus Experimental Group

<table>
<thead>
<tr>
<th>Control Group</th>
<th>AveQCT (second)</th>
<th>AveCP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>170.8</td>
<td>55.83</td>
</tr>
<tr>
<td>P2</td>
<td>132.3</td>
<td>32.5</td>
</tr>
<tr>
<td>P3</td>
<td>143.9</td>
<td>37.5</td>
</tr>
<tr>
<td>P4</td>
<td>63.4</td>
<td>15.83</td>
</tr>
<tr>
<td>P5</td>
<td>214.4</td>
<td>49.83</td>
</tr>
<tr>
<td>P6</td>
<td>129.2</td>
<td>54.16</td>
</tr>
<tr>
<td><strong>Ave±stddev</strong></td>
<td><strong>142.33±50.02</strong></td>
<td><strong>40.94±15.42</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experimental Group</th>
<th>AveQCT (second)</th>
<th>AveCP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P7</td>
<td>94.3</td>
<td>57.5</td>
</tr>
<tr>
<td>P8</td>
<td>70.9</td>
<td>54.16</td>
</tr>
<tr>
<td>P9</td>
<td>89</td>
<td>73.33</td>
</tr>
<tr>
<td>P10</td>
<td>76.2</td>
<td>65.82</td>
</tr>
<tr>
<td>P11</td>
<td>124.6</td>
<td>65</td>
</tr>
<tr>
<td>P12</td>
<td>61.8</td>
<td>61.66</td>
</tr>
<tr>
<td><strong>Ave±stddev</strong></td>
<td><strong>86.13±22.26</strong></td>
<td><strong>62.91±6.76</strong></td>
</tr>
</tbody>
</table>

The participants of the experimental group complete the questions faster than those of the control group (86.13±22.26 seconds versus 142.33±50.02 seconds), and the API caveats that the experimental group finds are more accurate than those found by the control group (62.91±6.76% versus 40.94±15.42%). The Wilcoxon Rank Sum Test shows that both the difference between question-completion time and the difference of correct-API-caveats percentage between the two groups are statistically significant at p-value < 0.05.
Evaluation (RQ3: The Improvement of API Caveats Accessibility)

The objective performance results and the “surprising” subject ratings reveal the accessibility issue of API caveats in API documentation which could create an illusion of already knowing the right API usage. As our knowledge graph makes the API caveats more easily accessible, beware of the API caveats would make the developers realize that using an API properly may not be as easy as it looks. This improved awareness of API caveats could make the developers more cautious when using an API, and thus potentially avoiding some mistakes in the first place.
Questions?
Suggestions?
Thank you! 😊