WCRE 2013
Koblenz, Germany

Automatic Discovery of Function Mappings between Similar Libraries

Cédric Teyton, Jean-Rémy Falleri, Xavier Blanc

Software Engineering (http://se.labri.fr)
*LaBRI, Université de **Bordeaux**, France*

Libraries & Maintenance

- Software massively use 3rd-party libraries
  - → Robust and efficient services
  - → Reuse = gain of time
- Libraries can be replaced
  - → Library Migration, happens in practice [1]
  - → Many motivations : more features, more convenient, not outdated
  - → Ex : switch MySQL database to H2

[1] Teyton et al, Mining Library Migration Graphs, WCRE 2012

Library Migration

- Full abandon of a source library in favor of a target library

TODO : adapt the project code base

- Hypothesis : the target library provides an undefined subset of similar functions from the source library

Migration : Tame a new API

- How To :
  1. Locate the function calls to the source library
  2. For each function :
     a) Figure out of what it does
     b) Search in the target library a similar function
     c) Adapt the source code

- Problem :

Two libraries = Two API structures and designs

Migration : Tame a new API

Example : (Apache commons → Google guava)

```
-import org.apache.commons.lang.Validate;
+import com.google.common.base.Preconditions;

public long getProblemVersion(String id) {
-    Validate.notNull(id);
+    Preconditions.checkArgument(id != null);
}
```

Validate.notNull(Object) is similar to Preconditions.checkArgument(Boolean)

→ How obvious is that ?

→ Textual similarity is not relevant

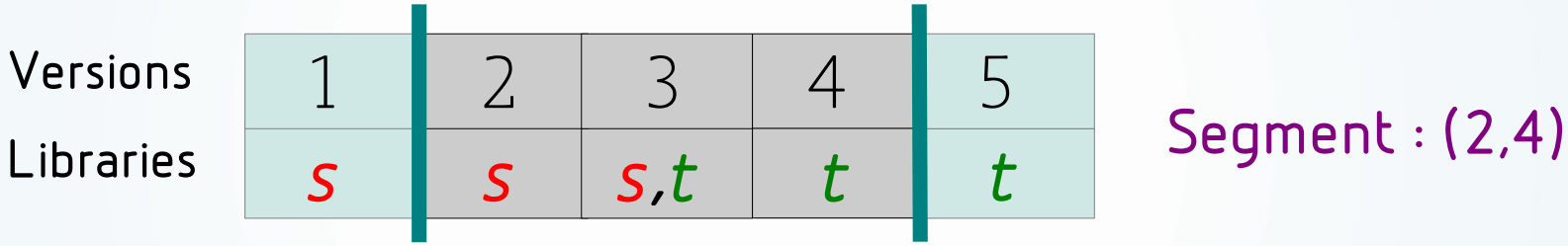
Difficult and time-consuming
to discover function mappings

Proposed Solution

- Identify software projects that already performed a given library migration : ***s*** → ***t***
- Analyze with precision their source code changes during the migration
- Extract function mappings :
  → a couple ***(x*** ↔ ***y)*** of functions, ***x*** ∈ ***s*** and ***y*** ∈ ***t***
  → ***x*** can be replaced by ***y*** and vice-versa (*"they somehow do the same thing"*)

(1) Segment Identification

- Does a software contain a migration ***s*** → ***t*** ? when?
- A migration segment is the smallest interval of project versions (V_i,V_j) where a migration happened



- Find segment has a cost : version download + static analysis of source code
  - → But... 100+ or 1000+ of versions

(1) Segment Identification

- We propose a divide and conquer based algorithm to efficiently identify migration segments

- Goals :
  - → Reduce the search space of versions
  - → Reduce the computation time

(2) Function mappings Extraction

- For a segment (V_i,V_j), analyze each pair versions (V_k,V_{k+1}), with $i < k < j-1$

- Compute source code diffs

- Identify migration hunks :
  - → Sequence of lines that contain :
    1. removed references to the source library
    2. added references to the target library
  - → A cartesian product between the 2 sets forms candidate function mappings

(2) Function mappings Extraction

Example : 3 Migrations Hunks

```java
public class Bar {

 public void test() {
  Log.getLog("MyLogger");
  something(3);
 }

 public void something(int i) {
  Log.getLog("MyLogger");
  if (i > 0) {
   Log.fatal("Error");
  }
 }
}
```

(a) Bar.java - version 1

```java
public class Bar {

 public void test() {
  Logger.getLogger("MyLogger");
  other(3);
 }

 public void other(int i) {
  Logger.getLogger("MyLogger");
  if (i > 0) {
   Logger.error("Error");
  }
 }
}
```

(b) Bar.java - version 2

```
@@ -4,2 +4,2 @@
-  Log.getLog("MyLogger");
-  something(3);
+  Logger.getLogger("MyLogger");
+  other(3);
@@ -8,2 +8,2 @@
- public void something(int i) {
-  Log.getLog("MyLogger");
+ public void other(int i) {
+  Logger.getLogger("MyLogger");
@@ -11 +11 @@
-   Log.fatal("Error");
+   Logger.error("Error");
```

(c) Diff Bar.java 1-2

2 function mappings :

Log.getLog(String) → Logger.getLogger(String) - Score : 2

Log.fatal(String) → Logger.error(String) - Score : 1

Idea : code updates performed in a similar location

Evaluation

- Setup
  - → Java Software
  - → 4 library migrations (8 since bi-directional)
  - → JSON, I/O, Lang and Mock domains
  - → 11,592 randomly selected OSS projects

- Data obtained
  - → 36 migration segments (hard to find data!)
  - → 285 migration hunks
  - → 2 persons to manually correct the mappings
  - → 115 correct function mappings, 113 wrong (50% precision)

Results online : http://www,labri.fr/perso/cteyton/Matching

Evaluation : Questions

1. Is our segment extraction algorithm faster than an exact algorithm ?

2. Is our mapping hunk construction technique a relevant solution ?

3. Can we improve precision using a filter ?

Results online : http://www,labri.fr/perso/cteyton/Matching

Evaluation : D&C Algorithm (1)

- Are we faster than an exact algorithm ? (that finds all the segments)
- Measured on 10 projects that contain segments
- Our algorithm :
  - → finds the same number of segments
  - → computes 85 % faster in time
- Theoretical but rare cases where our algorithm fails (cf. paper)

Evaluation : Hunk construction (2)

- Diff on files VS. Diff on class methods (AST-level)[2]
- Results

	Files	Methods	
correct mappings	115	92	(-23)
wrong mappings	113	245	(+132)
precision	*50 %*	*27 %*	*(-23 %)*

- Build migration hunks from source-level diff seems more relevant

[2] Schäfer,et Al. Mining framework usage changes from instantiation code, ICSE 2008

Evaluation : Filtering (3)

- Filtering technique to improve precision of mappings
  → Only keep mappings where both functions have their best score with each other

- Results
  → Can get better precision (up to 79%), but recall decreases (55% of rules found)

- Filters still to be improved
  → But lack of data is not ideal for filters

Conclusion

- Library Migration is a tedious operation
  → Many efforts to adapt the code
- Observe adaptive changes from software that did the job
  → Migration segment extraction + Migration hunks
- Early good results (but actual recall is missing)
- Next step :
  → Automatic code update, integration with “wrappers” [2]

[2] Bartolomei et al, Swing to SWT and back : Patterns for API migration by wrapping – ICSM 2010