Keeping documentation up to date with the source code

Nataliia Stulova

Software Composition Seminar

12 May 2020
$ whoami

2020 Postdoc (UniBe) - NOW :-) 
2019 Postdoc (EPFL) 
2018 PhD in Software, Systems and Computing (IMDEA SW/UPM) 
2013 MSc in Artificial Intelligence (UPM) 
2012 BSc in Systems Analysis (NTUU “KPI”)

my research focus: program specifications

C
-> assert statements

```c
double our_div(double num, double denom) {
    assert(denom != 0);
    return num / denom;
}
```

Racket
-> contracts

```racket
(define/contract (our-div num denom)
  (number? (and/c number? (not/c zero?))) . -> . number?)
  (/ num denom )
```

Java
-> doc comments

```java
/**
 * @param denom Must be non-zero
 */
public Double ourDiv (Double num, Double denom) {
```
previous research: executable specifications

- logic-based programming language (Prolog)
- assertion language
- combined static and dynamic verification frameworks

:- pred ourDiv(Num, Denom, Res)
   : num(Num), num(Denom), nonzero(Denom)
   => num(Res).

ourDiv(Num, Denom, Res) :- Res is (Num / Denom).
executable specifications: run-time verification

Q1: how to **instrument** the source code with specification checks?

Q2: how to **reduce** associated run-time **overhead**?

```prolog
ourDiv(Num, Denom, Res) :-
    check_precondition((num(Num), num(Demon), nonzero(Denom)), PRE),
    Res is (Num / Denom),
    check_postcondition(PRE, (num(Res)), POST).
```
NL specifications: program comments

```java
/**
 * Custom arithmetic division operation.
 *
 * @param num Numerator, any number
 * @param denom Denominator, any non-zero number.
 * @return Returns division result if denominator is not a zero, or infinity
 */

public Double ourDiv (Double num, Double denom) {
    if (!(denom == 0)) {return num / denom;}
    else {return Double.POSITIVE_INFINITY;}
}
```
** Custom arithmetic division operation. 

* @param num Numerator, any number
* @param denom Denominator, any non-zero number.
* @return Returns division result if denominator is not a zero, or infinity

```java
public Double ourDiv (Double num, Double denom) {
    if !(denom == 0) {return num / denom;}
    else {return Double.POSITIVE_INFINITY;}
}
```
current research: code-comment correspondence

Goal: keep comments up to date with the code during change

Challenges:
- how to map code to comments?
- how to detect inconsistencies?
- how to handle NL ambiguity (e.g, synonym use)?
- how to handle non-English words (e.g, abbreviations in method and variable names)
code-comment mapping

- get data
  - comment
  - method

- parse
  - sentences
  - AST

- transform
  - sentence BoWs
  - AST node BoWs

- map
mapping-based change analysis

1) get the change (new code)

2) build the mapping (old code)

3) check: all changes in the mapping?

change: [B, 3]

hit: [B, 3]

miss: [A]
implementation details

Parsing
- Javaparser for source code
- Stanford CoreNLP for comments

Mapping (IR-inspired)
- bag-of-words (BoW) representation of code and comment documents
- cosine similarity for document comparison

Change detection
- GumTreeDiff tool
evaluation details

- **mapping accuracy**: `<code,comment>` pair corpora
  - RepliComment set (method signature to comment mapping tests)
  - FunCom: ~2.1M pairs from Java projects
  - CodeSearchNet: ~2M pairs from open source libraries (Python, Javascript, Ruby, Go, Java, PHP)

- **change analysis**: commit histories
  - pre-processed dataset from USI of 1.3 Billion AST-level changes extracted with GumTreeDiff tool
  - individual projects: JOSS submissions
DEMO!