Automatically Extracting Instances of Code Change Patterns with AST Analysis

Matías Martínez, Laurence Duchien and Martin Monperrus

Context

A source code change pattern represents a kind of recurrent modification in software.
The automatic extraction of pattern instances is essential to measure their empirical importance.

Objective

Our goal is to present an automated process to define source code change patterns and quantify them from software versioning history.

Contributions

An approach to:

- specify source code change patterns with an abstraction over AST differencing
- automatically recognize concrete pattern instances based on the analysis of abstract syntax trees

Methodology

Representing software versioning history at the AST level

For a given pair of consecutive versions of a source code file from the versioning history, we compute the AST of both versions. We then apply a AST differencing algorithm [1] to extract the changes. We use an AST change taxonomy [2] that defines 41 source changes types, such as “Condition Expression Change”.

Representing change patterns at the AST level

Change pattern = {L,R,U}

- L list of micro-patterns
- R relation map
- U list of undesired changes

Example:

Bug fix pattern “Addition of Precondition Check with Jump” [3]

L: {m1=(“Statement Insert”(ct), “If”(et), *(pt)) , m2=(“Statement Insert”,“Return”, “If”) }

R: {m2.pt = m1.et} **U**: {Ø}

A micro-pattern is an abstraction over AST changes

micro-pattern = (ct, et, pt)

- **ct** change types (mandatory)
- **et** source code entity related to the change
- **pt** entity where the change takes place

The relation map R is a set of relations between entities (et) involved in the micro-patterns of L.

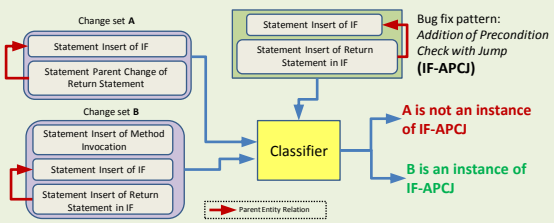
The list of undesired changes U represents AST changes that must not be present in the pattern instances.

Searching instances of AST change patterns

AST change classifier decides whether a given pattern is present or not inside an set of AST changes.

The classification procedure has three phases:

- change mapping
- identification of change relations
- exclusion of AST hunks containing undesired changes



Evaluation

We represent 18 code change patterns from a bug fix pattern catalog [3].
A “bug fix pattern” is a kind of change that often fixes bugs.
We mined instances of those patterns from 6 Java open source project totaling 23,597 Java revisions (Java file pairs).

Bug fix pattern	#
Change of If Condition Expression-IF-CC	4444
Addition of a Method Declaration-MD-ADD	4443
Addition of a Class Field-CF-ADD	2427
Addition of an Else Branch-IF-ABR	2053
Change of Method Declaration-MD-CHG	1940
Removal of a Method Declaration-MD-RMV	1762
Removal of a Class Field-CF-RMV	983
Addition of Precond. Check with Jump-IF-APCJ	667
Addition of a Catch Block-TY-ARCB	497
Addition of Precondition Check-IF-APC	431
Addition of Switch Branch-SW-ARSB	348
Removal of a Catch Block-TY-ARCB	343
Removal of an If Predicate-IF-RMV	283
Change of Loop Predicate-LP-CC	233
Removal of an Else Branch-IF-RBR	190
Removal of Switch Branch-SW-ARSB	146
Removal of Try Statement-TY-ARTC	26
Addition of Try Statement-TY-ARTC	18
Total	21,234

Future work

Integration of change context information in the change pattern.
Evaluation about precision and recall of AST change classifier.

References

[1] B. Fluri, M. Wursch, M. Pinzger, and H. Gall, “Change distilling: Tree differencing for fine-grained source code change extraction,” IEEE Transactions on Software Engineering, vol. 33, pp. 725 –743, nov. 2007.
[2] B. Fluri and H. Gall, “Classifying change types for qualifying change couplings,” in Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on, pp. 35 –45, 0-0 2006.
[3] K. Pan, S. Kim, and E. J. Whitehead, “Toward an understanding of bug fix patterns,” Empirical Software Engineering, vol. 14, pp. 286–315, Aug. 2008.