Research on NLP for RE at Università della Svizzera Italiana (USI): A Report

Arianna Blasi
Università della Svizzera italiana (USI)
Lugano, Switzerland
arianna.blasi@usi.ch

Mauro Pezzè
Università della Svizzera italiana (USI)
Lugano, Switzerland
mauro.pezze@usi.ch

Alessandra Gorla
IMDEA Software Institute
Madrid, Spain
alessandra.gorla@imdea.org

Michael D. Ernst
University of Washington
Seattle, WA
mernst@cs.washington.edu

Abstract

We report the activity of the Software Testing and Analysis Research (STAR) laboratory of USI Università della Svizzera italiana about the use of NLP to automatically generate test cases from documentation in natural language. We first introduce the research contributions of the group to contextualize the work related to NLP. We then summarize our research techniques to automatically generate test oracles from specifications expressed in terms of Javadoc tags using NLP. We conclude by presenting the challenges of shifting the focus on more complex software systems, and on more complex artifacts in natural language.

1 Team Overview

This report describes research to generate inputs and oracles to automatically test software systems. Our research uses Natural Language Processing (NLP) to automatically produce executable specifications from software documentation written in natural language. This complements our other work on automatically generating test inputs for applications with complex structured inputs [BDMP17], interactive and GUI-based applications [MPZ18], concurrent and distributed software systems [TP18], and test cases for exercising software in the field [GMPP17].

This report presents our work on the generation of test oracles, such as assertions for test cases. We focus on the automatic generation of *semantically relevant* oracles, that is, oracles that can reveal failures due to semantic mismatches with respect to software requirements. Such oracles are more powerful than simple implicit and regression oracles [BHM+15] that are commonly generated by automatic testing tools such as Randoop [PLEB07] and Evosuite [FA13].

In 2009 we started investigating redundancy intrinsically present in software systems [CGP09, CGPP15], and we designed a technique that exploits such redundancy to generate test oracles [CGG+14]. Along the same line of research, we designed and experimented with techniques and prototype tools to automatically identify semantically equivalent method calls in Java programs [GGM+14]. We then developed techniques for automatically generating assertions from specifications expressed in natural language. We developed an approach that exploits NLP to automatically infer executable specifications from Javadoc comments, and use such executable specifications as test oracles [GGEP16, BGK+18]. In the next sections, we describe the results that we obtained so far, and our research plans to automatically generate semantically relevant oracles from specifications in natural language with NLP.

2 Past Research on NLP for RE

Test cases can be generated from different sources of information [BP06, BHM+15]: requirements specifications (black-box and model-based testing), source code (white-box testing), possible faults (fault-based testing), former versions and similar code (regression and metamorphic testing). When generating test cases by exploiting requirements specifications, the goal is to identify a finite set of test inputs that properly sample the execution space (partition testing), and a set of assertions (oracles) that check the results of testing the software system. Most of the approaches for automatically generating test cases proposed so far focus on generating test inputs from formal and semi-formal specifications [PY07]. Relatively little work takes advantage of natural language requirements, and it uses simplistic techniques, such as pattern matching, to determine conditions related to nullness of parameters (Tan et al.'s @tComment [TMTL12]), part-of-speech tagging and pattern-matching to generates simple pre- and post-conditions (Pandita et al.'s ALICS [PXZ+12]). Some approaches take advantage of the simplifications induced by the structure of semi-formal specifications to generate test inputs. For instance, Wang et al. automatically derive test cases from use case specifications [WPG+15]. Many techniques use NLP to solve problems related to requirements quality, such as ambiguity [FDE+17], which are not strictly related to testing oracle specific issues.

We have investigated more powerful and effective approaches. As illustrated in the following simple example, the Javadoc tags indicate the scope of the specifications (i.e. whether it is a pre-condition on a parameter, or a post-condition on the method execution result), and this simplifies the task of processing the information for testing. However, Javadoc tags may predicate on program elements that are partially implicit, for instance implicit subjects referring to parameters, and often use developers' jargon, for instance *not null*. Such features pose a challenge for traditional natural language processing:

```
1 /**
2 * Merges the arrays in input
3 *
4 * @param x the first array, not null
5 * @param y the second array, not null
6 * @return an array which is the result of the merge, empty if both arrays are empty
7 * @throws IllegalArgumentEsxception if either array is null
8 */
9 public Object[] merge(Object[] x, Object[] y) throws IllegalArgumentException {...}
```

Listing 1: Sample Javadoc specification of a method

`@param` tags indicate the preconditions on the method input parameters, the `@return` tag (at most one) and `@throws` tags (one for each exception that the method may rise) indicate the postconditions of the method execution. The information expressed with Javadoc tags is useful to determine the correctness of the results of the test executions, but it is necessary to translate it into executable code assertions to act as test oracles.

For example, the translation of the specification expressed in the `@param` tags in Listing 1 is the following executable assertion, which automatically acts as testing oracle:

```
x != null && y!=null
```

the `@throws` tag translation is:

```
(x == null || y==null) ⟶ java.lang.IllegalArgumentException
```

and the `@return` tag translation is:

```
(x.length==0 && y.length==0) ⟶ result.length==0
```

We designed and developed Toradocu [GGEP16] later extended to Jdoctor [BGK+18], a technique that automatically infers executable assertions from comments in Javadoc tags expressed in natural language, as shown in the example. The early Toradocu approach performs simple translations of exceptional postconditions. Jdoctor extends Toradocu to all Javadoc tags and greatly improves the translation abilities of Toradocu, supporting also semantic similarity for interpreting synonyms [KSKW15].

Toradocu and Jdoctor use the Stanford Parser to produce a semantic graph for each sentence. First, they preprocess the text in natural language to deal with the peculiarities of Javadoc comments, which are rarely complete and grammatically sound English sentences. For example, most Javadoc specifications lack punctuation, many

have implicit subjects and verbs, and often intermix mathematical or code notation with English. Also, different types of tags need different preprocesses, and Toradocu and Jdoctor take this into account.

The core idea of Toradocu and Jdoctor is to exploit information already present in the source code to produce ready-to-use executable assertions. This approach does not require any other external intervention or effort from developers. The last experimental results obtained by executing Jdoctor on 6 popular open source Java projects are encouraging: the tool achieves 92% recall and 83% precision on 829 translations [BGK+18]. Also, Jdoctor assertions are officially integrated with Randoop [PLEB07], and in our evaluation they produce test cases that raise fewer false alarms and reveal more defects.

3 Research Plan on NLP for RE

The results of our past work confirm the research hypothesis of our long term research plan: *automatically generating test inputs and oracles from requirements specifications given in natural language with NLP is feasible and effective.*

In the short term, we plan to analyze free, *unstructured* text in Javadoc, beside the specific Javadoc tags that we already support. Moreover, we aim to extract information beyond functional properties. We plan to focus on temporal, security, performance and other non-functional properties. As an example, Figure 1 shows some temporal properties about call protocols[1]. Such information is very useful for reducing the amount false alarms that affect existing testing approaches.

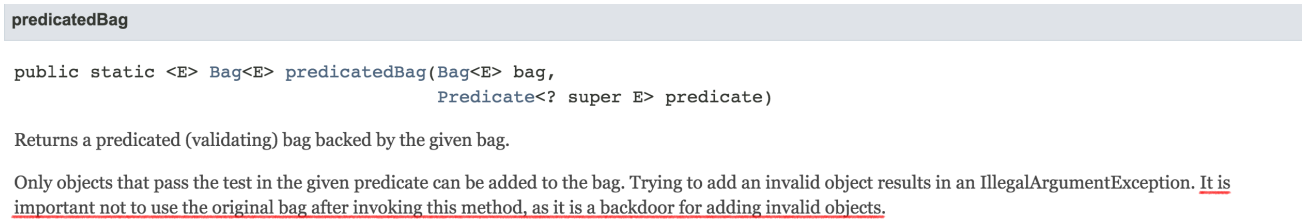

Figure 1: *Javadoc documentation (free text part) of Apache Commons Collections*

We will combine different approaches to interpret various properties expressed in natural language. We will resort to Open Information Extraction to infer information from unstructured text [DCG13, FSE11, SBS+12], and natural language parsing, pattern matching, semantic similarities and machine translation techniques to match documentation with code elements.

Our mid-term plan aims to extend Jdoctor to deal with information coming from other artifacts in natural language, such as wikis, issue trackers, and community forums, which are commonly available for popular applications. Even if these artifacts do not have a narrow scope as Javadoc comments do — i.e., Javadoc refers to a specific method or a specific class — they are still often partially-structured, and thus we believe that our techniques, if properly extended, can deal with them. The software engineering research community already produced some techniques that derive test artifacts from system requirements such as use-case requirements [WPG+15, MPGB18].

Our long term plan is to define and develop a set of techniques to automatically test *human-centric* software systems. Such systems have key features that make them different from traditional software systems: First and foremost, the user is an integral part of the system. Secondly, they often integrate different sensors and physical devices. Lastly, they often rely on machine learning components that drive the decisions of the system based on the observed inputs from sensors. In a nutshell, human-centric software systems can be seen as an evolution of ultra large software systems also called systems of systems [MPS08].

To deal with human-centric software systems we need to radically change the considered scenario and widen the set of techniques that we plan to use, mostly because the expected behavior of the system may be hard to predict, and it is seldom specified in the requirements. So far we studied the problem of automatically generating test inputs and oracles for functional properties of program units (classes and methods) with *deterministic* behaviors.

To address the problem of properly testing human-centric software systems, we need to move from functional properties of software components with deterministic behavior, to properties of subsystems with *non deterministic*

[1] `https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/BagUtils.html`

behavior. Non determinism may derive from concurrency, and may be due to machine learning components that act differently depending on the underlying model they use. Moreover, external physical sensors and users involved in the system may increase the uncertainty of the expected behavior of the system.

Despite these challenges, we still plan to focus our analysis on natural language artifacts, and aim to infer the missing information to test such systems. No matter how complex such systems may be, their requirements still have to be expressed in some form: It could be, for example, classical user stories. We will investigate what kind of artifacts are mostly used to document such type of systems. We will study different ways of contextualizing the fragmented and incomplete information expressed in natural language to solve ambiguities and incompleteness, and we will properly exploit the inferred specification to test these complex systems.

Acknowledgments

This work was partially supported by the Spanish projects DETEST, by the Madrid Regional projects BLUETS and MadridFlightOnChip, and by the Swiss project ASTERIx: Automatic System TEsting of inteRactive software applIcations (SNF-200021_178742). This material is also based on research sponsored by DARPA under agreement numbers FA8750-12-2-0107, FA8750-15-C-0010, and FA8750-16-2-0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[BDMP17] Pietro Braione, Giovanni Denaro, Andrea Mattavelli, and Mauro Pezzè. Combining symbolic execution and search-based testing for programs with complex heap inputs. In *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA '17, pages 90–101. ACM, 2017.

[BGK+18] Arianna Blasi, Alberto Goffi, Konstantin Kuznetsov, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Sergio Delgado Castellanos. Translating code comments to procedure specifications. In *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA '18. ACM, 2018.

[BHM+15] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

[BP06] Luciano Baresi and Mauro Pezzè. An introduction to software testing. *Electronic Notes in Theoretical Computer Science*, 148(1):89–111, 2006.

[CGG+14] Antonio Carzaniga, Alberto Goffi, Alessandra Gorla, Andrea Mattavelli, and Mauro Pezzè. Cross-checking oracles from intrinsic software redundancy. In *Proceedings of the International Conference on Software Engineering*, ICSE '14, pages 931–942. ACM, 2014.

[CGP09] Antonio Carzaniga, Alessandra Gorla, and Mauro Pezzè. Fault handling with software redundancy. In R. de Lemos, J. Fabre, C. Gacek, F. Gadducci, and M. ter Beek, editors, *Architecting Dependable Systems VI*, pages 148–171. Springer, 2009.

[CGPP15] Antonio Carzaniga, Alessandra Gorla, Nicolò Perino, and Mauro Pezzè. Automatic workarounds: Exploiting the intrinsic redundancy of web applications. *ACM Transactions on Software Engineering and Methodologies*, 24(3):16, 2015.

[DCG13] Luciano Del Corro and Rainer Gemulla. Clausie: Clause-based open information extraction. In *Proceedings of the International Conference on World Wide Web*, WWW '13, pages 355–366. ACM, 2013.

[FA13] Gordon Fraser and Andrea Arcuri. Whole test suite generation. *IEEE Transactions on Software Engineering*, 39(2):276–291, 2013.

[FDE+17] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, and Stefania Gnesi. Natural language requirements processing: a 4d vision. 34(6):28–35, 2017.

[FSE11] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1535–1545. Association for Computational Linguistics, 2011.

[GGEP16] Alberto Goffi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè. Automatic generation of oracles for exceptional behaviors. In *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA '16, pages 213–224. ACM, 2016.

[GGM+14] Alberto Goffi, Alessandra Gorla, Andrea Mattavelli, Mauro Pezzè, and Paolo Tonella. Search-based synthesis of equivalent method sequences. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '14, pages 366–376. ACM, 2014.

[GMPP17] Luca Gazzola, Leonardo Mariani, Fabrizio Pastore, and Mauro Pezzè. An exploratory study of field failures. In *Proceedings of the International Symposium on Software Reliability Engineering*, ISSRE '17, 2017.

[KSKW15] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the International Conference on International Conference on Machine Learning*, ICML '15, pages 957–966, 2015.

[MPGB18] Phu X. Mai, Fabrizio Pastore, Arda Goknil, and Lionel C. Briand. A natural language programming approach for requirements-based security testing. In *Proceedings of the International Symposium on Software Reliability Engineering*, ISSRE '18, pages 58–69. IEEE Computer Society, 2018.

[MPS08] Hausi Muller, Mauro Pezzè, and Mary Shaw. Visibility of control in adaptive systems. In *ULSSIS '08: Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems*, pages 23–26. ACM, 2008.

[MPZ18] Leonardo Mariani, Mauro Pezzè, and Daniele Zuddas. Augusto: Exploiting popular functionalities for the generation of semantic gui tests with oracles. In *Proceedings of the International Conference on Software Engineering*, ICSE '18, pages 280–290, 2018.

[PLEB07] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *Proceedings of the International Conference on Software Engineering*, ICSE '07, pages 75–84. ACM, 2007.

[PXZ+12] Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. Inferring method specifications from natural language api descriptions. In *Proceedings of the International Conference on Software Engineering*, ICSE '12, pages 815–825. IEEE Computer Society, 2012.

[PY07] Mauro Pezzè and Michal Young. *Software Testing and Analysis: Process, Principles and Techniques.* Wiley, 2007.

[SBS+12] Michael Schmitz, Robert Bart, Stephen Soderland, Oren Etzioni, et al. Open language learning for information extraction. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 523–534. Association for Computational Linguistics, 2012.

[TMTL12] Shin Hwei Tan, Darko Marinov, Lin Tan, and Gary T. Leavens. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, ICST '12, pages 260–269. IEEE Computer Society, 2012.

[TP18] Valerio Terragni and Mauro Pezzè. Effectiveness and challenges in generating concurrent tests for thread-safe classes. In *Proceedings of the International Conference on Automated Software Engineering*, ASE '18. ACM, 2018.

[WPG+15] Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal. Automatic generation of system test cases from use case specifications. In *Proceedings of the International Symposium on Software Testing and Analysis*, ISSTA '15, pages 385–396. ACM, 2015.