Intrinsic software redundancy for self-healing software systems and automated oracle generation

Antonio Carzaniga, Alberto Goffi, Alessandra Gorla,* Andrea Mattavelli,
Nicoló Perino, Mauro Pezzè,† Paolo Tonella‡

Faculty of Informatics, University of Lugano, Via Giuseppe Buffi, 13, 6900 Lugano
{*name.surname*}@usi.ch, alessandra.gorla@imdea.org, tonella@fbk.eu

Abstract: Software systems are intrinsically redundant. We identify the sources of intrinsic software redundancy in good design practices, and suggest how to exploit intrinsic software redundancy to augment software systems with self-healing capabilities and to automatically generate test oracles.

Reliability is becoming a necessity for many software systems used in everyday life, where failures may interrupt important services with severe economical and social consequences. In system and software engineering, reliability is traditionally approached by adding some form of redundancy to overcome the consequences of faults that are unavoidable in human artifacts. RAID (Redundant Array of Independent Disks) [PGK88], HDFS (Hadoop Distributed File Systems) and N-version programming [Avi85] are good examples of the use of redundancy for improving hardware, data and software reliability, respectively. In these approaches, redundancy is added *deliberately* to the systems to improve reliability, and comes with additional costs that may be acceptable for some systems, like safety critical systems, but may not meet the requirements of other domains, like many software applications used in everyday life. We observe that many software systems are characterized by an *intrinsic* form of redundancy that derives from good design practices: Design for reusability creates many distinct APIs to improve compatibility with different uses, and this results in a variety of implementations of the same functionality; Performance optimisation results in different methods implementing the same functionality with different, optimised code; Backward compatibility preserves the old versions of the reimplemented functionalities thus offering redundant methods. Redundancy is massively present in modern software systems: Our manual inspection of several popular libraries including Apache Ant, Google Guava, Joda Time, Eclipse SWT, GraphStream, Apache Lucene, GoogleMaps and YouTube identified over 4,700 redundant methods, with an average of 5 redundant methods per class [CGPP10, CGM+13].

Redundancy is present at many abstraction levels. So far, we have exploited the redundancy intrinsically present at the method call level both to build self-healing software systems

*Alessandra Gorla is with the IMDEA Software Institute, Madrid, Spain
†Mauro Pezzè is also with the University of Milano-Bicocca, Milano, Italy
‡Paolo Tonella is with Fondazione Bruno Kessler, Trento, Italy

and to generate semantically relevant test oracles. Once identified a set of redundant methods, we add self-healing capabilities by automatically deploying a mechanism that substitutes a failing method with a redundant one to avoid the failure. We call such approach *automatic workaround*. The design of automatic workarounds requires a mechanism to reveal failures, a method to roll back to a correct state and a method to execute a redundant method. We approach the different problems relying on assertions embedded in the code, an optimised rollback mechanism and a source to source code transformation, respectively [CGPP10, CGM+13]. We automatically generate semantically relevant test oracles by comparing the results of executing the method calls in the original program with the results of executing equivalent methods in the same context, and we call such oracles *cross-checking oracles* [CGG+14]. Given a set of redundant methods, we generate cross-checking oracles by cloning the program state before executing a method call, executing the method call in the original program and the corresponding redundant method calls on the cloned state, and comparing the results and the obtained states. In this way cross-checking oracles can reveal discrepancies between the executions of methods that should produce equivalent results and reveal failures related to the program semantics.

The automatic synthesis of both self-healing mechanisms and automated oracles requires a set of redundant program elements as input. We can automatically synthetize redundant methods without expensive formal specifications by exploiting search-based techniques. We use a genetic algorithm for synthetizing a method call equivalent to a given method for an initial scenario (usually one or few test cases). We then look for a counterexample that, if found, gives us a new scenario to search for a redundant method, and, if not found, validates the redundancy of the original and the identified methods. We can automatically synthetize a large amount of redundant method sequences by applying the approach to all methods in the target software system [GGM+14].

References

[Avi85] Algirdas Avizienis. The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, 1985.

[CGG+14] Antonio Carzaniga, Alberto Goffi, Alessandra Gorla, Andrea Mattavelli, and Mauro Pezzè. Cross-Checking Oracles From Intrinsic Software Redundancy. In *Proceedings of the 2014 International Conference on Software Engineering*, 2014.

[CGM+13] Antonio Carzaniga, Alessandra Gorla, Andrea Mattavelli, Nicoló Perino, and Mauro Pezzè. Automatic recovery from runtime failures. In *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[CGPP10] Antonio Carzaniga, Alessandra Gorla, Nicolò Perino, and Mauro Pezzè. Automatic Workarounds for Web Applications. In *Proceedings of the 18th International Symposium on the Foundations of Software Engineering*, 2010.

[GGM+14] Alberto Goffi, Alessandra Gorla, Andrea Mattavelli, Mauro Pezzè, and Paolo Tonella. Search-based Synthesis of Equivalent Method Sequences. In *Proceedings of the 22nd International Symposium on Foundations of Software Engineering*, 2014.

[PGK88] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Record*, 17(3):109–116, 1988.