Error models for the representative injection of software defects

Anna Lanzaro[1], Roberto Natella[1], Stefan Winter[2], Domenico Cotroneo[1], Neeraj Suri[2]

[1] DIETI, Federico II University of Naples, Italy
{anna.lanzaro, roberto.natella, cotroneo}@unina.it
[2] DEEDS Group, Dept. of CS, TU Darmstadt, Germany
{sw, suri}@cs.tu-darmstadt.de

Abstract: This paper considers the representativeness of injected error models for ascertaining software defects.

Business- and safety-critical systems are more and more relying on software. Therefore, while in the past these systems were mainly threatened by hardware faults, they are today increasingly exposed to *software faults*, as demonstrated by recent severe software-related accidents [WDS$^+$10]. It is a matter of fact that, despite careful engineering and rigorous quality assurance, critical systems are deployed with residual (unknown) software defects. This problem is exacerbated by the massive reuse of legacy and off-the-shelf software components [Wey98, Voa98]: When a component is reused in a new context, the system may use parts of the component that were previously seldom used and only lightly tested, or may interact with the component in unforeseen ways, thus exposing residual software faults in the component that had not been discovered before.

It thus becomes important to adopt software fault tolerance strategies, in order to prevent such residual defects in less critical parts from affecting more critical parts of a system. *Software fault injection* (SFI) is an experimental approach to assess the dependability of software-intensive systems in the presence of faulty software components, and to guide the development of software fault tolerance mechanisms and algorithms. SFI deliberately introduces faults in software components [Voa98, KS08] for:

- **Validating fault-tolerance mechanisms**: SFI can evaluate error detection and handling mechanisms (such as assertions and exception handlers) against component faults, and to add and to improve such mechanisms if necessary.
- **Aiding FMECAs (Failure Mode, Effects, and Criticality Analysis)**: Developers can quantify the impact of a faulty component on the overall system (e.g., in terms of catastrophic system failures), and mitigate risks by focusing testing efforts on the most critical components or by revising the system design.
- **Dependability benchmarking**: SFI helps developers to choose among alternative systems or components the one that provides the best dependability and/or performance in the presence of other, faulty, components.

Existing methods for the injection of *representative* software faults (i.e., the errors generated by injected faults match the effects of real residual software faults in a component) consist in the corruption of the code of software components [NCDM13]. Unfortunately, such code mutation suffers from practical disadvantages, including the need for source code (which may be impossible to obtain for proprietary third-party components) and the ability to mutate binary code (which has been proven to be very difficult due to the semantic gap between source-level faults and their corresponding binary translation).

Therefore, practitioners often restrict themselves to a projection (error model) of the emulated defects' effects at the interfaces between software components. *Interface error injection* overcomes the limitations of code mutations, by mimicking the effects (i.e., *errors*) produced by faulty components through the injection of *exceptional or invalid values* at the component's interface [WSSM11].

The research question we address is whether existing error models for interface injections are representative projections of residual defects commonly found in software systems. For this purpose, we propose a method for analyzing how faults in software components manifest as errors at the interfaces of software components (*error propagation*) [LNW+14]. The method injects faults in the software component under analysis through code mutations, and it instruments and executes the software component to identify the effects of injected faults on the program that uses the component, including the corruption of data structures shared between the program and the component, and erroneous return values from function calls. A case study with widely used software libraries reveals that existing interface error models are not suitable for emulating software faults, and provides useful insights for improving the representativeness of interface error injections.

Acknowledgments: Work supported by the projects *SVEVIA* (PON 02_00485_3487758), *TENACE* (PRIN n.20103P34XC), *BMBF EC-SPRIDE*, and *LOEWE-CASED*.

References

[KS08] K. Kanoun and L. Spainhower. *Dependability Benchmarking for Computer Systems*. Wiley-IEEE Computer Society, 2008.

[LNW+14] A. Lanzaro, R. Natella, S. Winter, D. Cotroneo, and N. Suri. An Empirical Study of Injected versus Actual Interface Errors. In *Proc. ISSTA*, pages 397–408, 2014.

[NCDM13] R. Natella, D. Cotroneo, J.A. Durães, and H.S. Madeira. On Fault Representativeness of Software Fault Injection. *IEEE Trans. Softw. Eng.*, 39(1):80–96, 2013.

[Voa98] J.M. Voas. Certifying off-the-shelf software components. *IEEE Computer*, 31(6):53–59, 1998.

[WDS+10] W.E. Wong, V. Debroy, A. Surampudi, H. Kim, and M.F. Siok. Recent catastrophic accidents: Investigating how software was responsible. In *Proc. SSIRI*, pages 14–22, 2010.

[Wey98] Elaine J Weyuker. Testing component-based software: A cautionary tale. *IEEE Softw.*, 15(5):54–59, 1998.

[WSSM11] S. Winter, C. Sârbu, N. Suri, and B. Murphy. The impact of fault models on software robustness evaluations. In *Proc. ICSE*, pages 51–60, 2011.