Notice of the Final Oral Examination
for the Degree of Master of Science

of

GIOVANNI MURGUIA

BSc (Instituto Tecnológico y de Estudios Superiores de Monterrey, Mexico, 2008)

“Autonomic Test Case Generation of Failing Code Using AOP”

Department of Computer Science

Tuesday, August 25, 2020
10:30 A.M.
Remote Defence

Supervisory Committee:
Dr. Hausi Müller, Department of Computer Science, University of Victoria (Supervisor)
Dr. Alex Thomo, Department of Computer Science, UVic (Member)

External Examiner:
Dr. Nikitas Dimopoulos, Department of Electrical and Computer Engineering, UVic

Chair of Oral Examination:
Dr. Margo Matwychuk, Department of Anthropology, UVic

Dr. Stephen Evans, Acting Dean, Faculty of Graduate Studies
Abstract

As software systems have grown in size and complexity, the costs of maintaining such systems increases steadily. In the early 2000's, IBM launched the autonomic computing initiative to mitigate this problem by injecting feedback control mechanisms into software systems to enable them to observe their health and self-heal without human intervention and thereby cope with certain changes in their requirements and environments. Self-healing is one of several fundamental challenges addressed and includes software systems that are able to recover from failure conditions. There has been considerable research on software architectures with feedback loops that allow a multi-component system to adjust certain parameters automatically in response to changes in its environment. However, modifying the components' source code in response to failures remains an open and formidable challenge.

Automatic program repair techniques aim to create and apply source code patches autonomously. These techniques have evolved over the years to take advantage of advancements in programming languages, such as reflection. However, these techniques require mechanisms to evaluate if a candidate patch solves the failure condition. Some rely on test cases that capture the context under which the program failed—the patch applied can then be considered as a successful patch if the test result changes from failing to passing. Although test cases are an effective mechanism to govern the applicability of potential patches, the automatic generation of test cases for a given scenario has not received much attention. ReCrash represents the only known implementation to generate test cases automatically with promising results through the use of low-level instrumentation libraries.

The work reported in this thesis aims to explore this area further and under a different light. It proposes the use of Aspect-Oriented Programming (AOP)—and in particular of AspectJ—as a higher-level paradigm to express the code elements on which monitoring actions can be interleaved with the source code, to create a representation of the context at the most relevant moments of the execution, so that if the code fails, the contextual representation is retained and used at a later time to automatically write a test case. By doing this, the author intends to contribute to fill the gap that prevents the use of automatic program repair techniques in a self-healing architecture.
The prototype implementation engineered as part of this research was evaluated along three dimensions: memory usage, execution time and binary size. The evaluation results suggest that (1) AspectJ introduces significant overhead with respect to execution time, (2) the implementation algorithm causes a tremendous strain on garbage collection, and (3) AspectJ incorporates tens of additional lines of code, which account for a mean size increase to every binary file of a factor of ten compared to the original size. The comparative analysis with ReCrash shows that the algorithm and data structures developed in this thesis produce more thorough test cases than ReCrash. Most notably, the solution presented here mitigates ReCrash's current inability to reproduce environment-specific failure conditions derived from on-demand instantiation. This work can potentially be extended to apply in less-intrusive frameworks that operate at the same level as AOP to address the shortcomings identified in this analysis.