# Locating Bugs in C/C++ Programs by Generating Directed Unit Tests

| | | |
|---|---|---|
| Project Type: | **Research Project (NPRG070)** | |
| Study Program: | **Computer Science - Software Systems** | |
| Student: | **bc. Roman Vašut** | bohdanqq@proton.me |
| Supervisor: | **doc. RNDr. Pavel Parízek, Ph.D.** | parizek@d3s.mff.cuni.cz |

## Motivation and Background

Debugging large and complex software systems is really a tedious and time-consuming activity. For an observed error, developers usually start by reproducing the error in a controlled environment, and try to determine the root cause of the error, i.e. the faulty statements (bugs) in the program source code. Then developers modify the source code to fix the respective bugs and validate the patch is correct. In practice, all these steps take a lot of time and effort by developers, especially in case of applications with long executions where developers have to manually inspect huge log files or use interactive debuggers to single-step through long fragments of program execution traces. Many researchers are therefore working on various automated techniques (and related tool support) to make the whole process of debugging much faster and easier for developers.

In particular, the supervisor of this project has recently started an initiative focused on techniques and tools for debugging large systems that include long-running services and computations (jobs). The proposed methodology involves the following steps (to be performed repeatedly):

- Dynamic runtime analysis of the subject application to monitor its execution that reproduces the error and to record necessary information about the execution (e.g., values of method call arguments).
- Generating directed unit tests for specific application components and methods that are suspicious (i.e., candidates for containing the bugs), using information provided by the dynamic analysis.
- Execution of the generated tests to navigate towards the likely root cause of the runtime error in question, e.g. through discovering some differences between passing and failing tests in terms of method call arguments and input program states.

One student is already working on a prototype implementation of the necessary infrastructure for all steps of this debugging methodology, in the scope of his master thesis, but only in the context of programs written in Java, using available tools and libraries such as DiSL [1] and JUnit [2].

## Project Description

The goal of this project is to create a significant part of the necessary infrastructure for C/C++ applications. Specifically, it means (1) adapting relevant concepts from Java platform and technology stack to the C/C++ world, (2) design and implementation of several necessary modules, and (3) evaluating the whole approach on a middle-sized C/C++ program as a case study. Popular tools available for C/C++, such as LLVM [3], Valgrind [4] or Pin [5], and unit test frameworks (e.g., Google Test [6] or CppUnit [7]) will be used.
The prototype has to support just a subset of C/C++ language constructs (e.g., methods taking arguments just of numeric and string types) and selected features of the standard library (e.g., with very limited support for STL and collections). Optionally, if time permits, support for additional, more advanced features of C/C++, will be implemented. The required core parts ("minimal viable product") and subgoals are the following: usage of dynamic analysis to record information during program execution, generating tests for procedures that take parameters of simple types, and demonstration of basic functionality on some case study. We also expect that, during the course of this project, it will be necessary to solve many technical challenges associated with the C/C++ programming languages, tools, and runtime environment.
Some design and technical decisions will be partially inspired by the ongoing work on a similar infrastructure for debugging Java programs.

## Project Output (Deliverables)

The main part of deliverables will be the prototype implementation of several core modules needed to evaluate the proposed debugging methodology. This should, in particular, include some kind of a plugin for a dynamic analysis framework of choice, a module that can generate unit tests for C/C++ methods and classes, a runner for generated tests tailored for our use case, and scripts that will orchestrate the whole process.
Another part of the project output will be technical documentation, which should also discuss the main technical challenges and our solutions.
We expect that the implemented prototype (framework) will be one part of contribution presented in a future publication, but more work still would have to be done before the paper can be written and submitted.
The complete source code of the prototype implementation will be later released to the public as open-source software (e.g., on GitHub).

## Time Schedule and Milestones

We plan to start by thorough research of available technologies, their features and limitations. Then the work on core modules will be performed in the order specified below. We assume that each module should take between 1-2 months.

- Usage of dynamic analysis to record information about program execution (months 2-3).
- Generation of directed unit tests using a popular framework of choice  (months 3-4).

- Runner for unit tests and scripts that will orchestrate everything together (months 5-6).

In the last few months (7-9), the project team will focus on writing the documentation and preparation of the demo (case study). This also includes a scheduled reserve of 1-2 months for the case of unexpected implementation challenges.

## Team (Cooperation)

The project team will have these members:

- Roman Vašut - master student at the Faculty of Mathematics and Physics, Charles University. Project leader. Responsible for the main part of design and implementation.
- Pavel Parízek - Faculty of Mathematics and Physics, Charles University. Project supervisor.
- Denis Leskovar - master student at the Faculty of Mathematics and Physics, Charles University. Working already on the prototype infrastructure for Java-based systems. His main responsibility in this project will be consultations of the design, technical help, and code reviews.

All work on this project will be stored in repositories on the faculty GitLab.

## References

[1] DiSL, a domain-specific language for Java bytecode instrumentation. https://disl.ow2.org/.
[2] JUnit, testing framework for Java and the JVM, https://junit.org/.
[3] The LLVM Compiler Infrastructure, https://llvm.org/.
[4[ Valgrind, a dynamic analysis framework, https://valgrind.org/.
[5] Pin, a dynamic binary instrumentation framework, http://www.intel.com/software/pintool.
[6] Google Test, https://github.com/google/googletest.
[7] CppUnit, https://sourceforge.net/projects/cppunit/.