6156: Topics in Software Engineering

Team: TestInProd
Nigel Schuster (ns3158), Will Pascucci (wmp2108)

Testing is an essential part of software engineering, and robust test practices help projects find critical bugs and prevent failures. Testing surfaces a wide range of issues and often prevents regressions. Test practices have evolved as much as software engineering has evolved over the past few decades, beginning with manual testing, evolving into automated test suites, and growing now into a critical aspect of DevOps processes. Nowadays, we have a wide range of tools available to enhance our testsuite, including Selenium, Snapshot Testing, Mutation Testing and more. Yet, the fundamental practice of writing a testcase has barely evolved, since the beginning of testing. Our team hopes to improve the user experience of developing testcases for a target audience of all software engineers.

Developer's perception of testing as another task on their respective to-do lists and the amount of additional work required to create full test cases inspires our research question: can a testing framework leverage developer's existing interaction with their programs to reduce the effort required to build test cases? We aim to answer this question by building on a test framework, recruiting developers to use the new framework, and gauge their opinion on the testing framework and thoughts on its use in larger scale applications.

The goal of team "TestInProd" is to augment an existing test framework to add features that will improve the developer experience of testing. For unit and integration testing one still selects concrete scenarios that should be tested, mocks peripheral components, and writes the code to test the scenario. This valuable manual effort is perceived as onerous, and is often left until later in the development process. Techniques such as Fuzz or Metamorphic Testing provide a step ahead, but often only serve in a supplemental nature.

We propose to fundamentally alter the approach of writing testcases. Our first change is to generate testcases from on past executions. We base this on the following observation: Before writing a testcase a developer usually tests the code manually (excluding functional testing). Given a successful execution, the captured input allows us to immediately generate a regression testcase. A developer can choose to alter the code afterwards, but he has a working testcase available already. The second change we propose regards improving the testsuite. Instead of the developer using code coverage or mutation testing separately to improve his testsuite, we want to utilize the tools directly to proactively suggest improvements. Finally, we do not believe that testing is another step in developing a software component, but it is intertwined with the development of the actual code. Therefore we propose a testing framework that continuously provides the developer with information and is not just run periodically.

Our approach differs from existing frameworks that auto-generate test cases in that the framework will capture developer's inputs as opposed to just creating automatic test cases. With

captured developer inputs, we hope to then leverage approaches like metamorphic testing using those inputs.

Our approach will also function on the scale of working with teams by helping each developer without the test cases interfering with others. Generated test cases will not override one another and will be added to increase test coverage. The framework will not react differently to multiple developers working on one class than it would to one developer working on that class multiple times.

We hope to implement these features in Python and the Python testing framework PyTest, a popular, open-source testing tool. Python is a popular language that allows for deep code introspection, has many open source test tools available, and will be easier to evaluate than more esoteric or specific languages. We would write our application as a wrapper to PyTest, offering our features on top of the functionality provided by PyTest. Through approaches like custom annotations, we will make it easier for a developer to mark certain functions, classes, and modules to capture successful test runs, suggest expansions of test coverage, and provide potential improvements.

Our evaluation strategy will focus on the developer experience in using the framework. We will recruit participants from Columbia's CS program through word of mouth and engage those participants in using our framework. Participants will be taught the basic functionality and use of the modified PyTest and will then be asked to write test cases for a provided sample application. We will record developer's experiences in using the framework and rate their UX. Developer's will be asked to fill out a survey on their experience and rate it against their general experience in testing.

We will deliver a python-based testing framework that implements our proposed features, a survey that evaluates the improvements in the perception of testing achieved by our framework, and an analysis of the results of our survey. We will demo the usage of our framework on simple test classes and functions to show the test cases generated by our framework. Currently, both members of the team will contribute to the development of the framework without a rigid delineation in responsibilities. The survey will cover the participant's experience with the framework and the impact it had on their testing experience.

The framework will be developed in an iterative capacity, building on a working framework in PyTest and adding features throughout the project lifecycle. Code, documentation, and commit history will be hosted on GitHub in a public repository. The developer survey will be written using Google surveys which will be given to the developers participating in the testing immediately after they have finished working with the framework in a controlled setting which will involve developers working with traditional testing and our framework.