Symflower-Local

Generierte Tests im Entwicklungsworkflow

Programm / Ausschreibung	IWI, IWI, Basisprogramm Ausschreibung 2023	**Status**	abgeschlossen
Projektstart	15.02.2023	**Projektende**	31.05.2024
Zeitraum	2023 - 2024	**Projektlaufzeit**	16 Monate
Keywords			

Projektbeschreibung

Symflower beschäftigt sich seit seiner Gründung 2018 mit der Entwicklung einer Symbolic-Execution, um automatisiert Unit-Tests für Softwareentwickler zu erstellen und diese im Entwicklungsprozess produktiver zu machen. Die R&D Phase, um diese Symbolic Execution zu entwickeln, hat länger gedauert als ursprünglich veranschlagt. Einerseits konnte das Tech Team dafür erst Anfang 2020 komplettiert werden und andererseits sind wir auf einige technische Hürden gestoßen, welche die Entwicklung verzögert haben.

Mit Stand Jänner 2023 sind wir kurz davor, erste Umsätze aus der Serienüberführung der Resultate aus dem letzten Basisprogramm (siehe eCall-Nr. 35401754) zu generieren. Sehen aber starken Bedarf darin, weiter experimentelle Entwicklung zu betreiben, um das volle Potential unserer Symbolic Execution ausschöpfen zu können und den Markt breiter zu adressieren.

Wir konnten 2019 den Top-Tech-Investor eQventure für Symflower gewinnen. Durch deren Investment erhalten wir aktive Unterstützung sowie Coaching von den beiden Tricentis (erstes Software-Tech-Unicorn Österreichs) Gründern Franz Fuchsberger und Wolfgang Platz. Im November 2022 wurde eine weitere Finanzierungsrunde von eQventure und dem Oberösterreichischen Hightechfonds über ein Wandeldarlehen vorgenommen.

Im Q3 2022 konnten wir unsere Marketing-Abteilung neu ausrichten und sind hier nun bestens für unseren Go-To-Market ausgestattet.

Mit Februar 2023 komplettiert Mathias Holzinger als CEO das Management-Team von Symflower.

Um Softwareentwicklern größtmöglichen Wert zu stiften, muss sich Symflower-Local in den natürlichen Entwicklungs-Workflow eines Softwareentwicklers eingliedern. Das bedeutet, dass Tests zum richtigen Zeitpunkt generiert werden und generierte Tests bereits bestehende Tests berücksichtigen müssen. Ist das nicht der Fall, kann die erzeugte Menge an Tests den Softwareentwickler überfordern. Um diese zentrale Problemstellung zu lösen, müssen Antworten auf folgende Forschungsfragen erarbeitet werden:

- Wie müssen sich generierte Tests in gängige Entwicklungs-Workflows integrieren, um die Arbeit der Entwickler effektiv zu unterstützen und zu beschleunigen?
- Wie geht man mit bestehenden, manuell erstellten Tests um und wie spielen diese mit generierten Tests zusammen?
- Wie können bestehende Test-Konventionen eines Projektes erkannt und genutzt werden, damit sich die generierten Tests nahtlos in bestehende Test-Suiten einfügen?
- Wie können die Resultate der Symflower-Testgenerierung den Entwickler beim Schreiben von Code und beim Debugging anleiten?
- Wie können Softwareentwickler mit tausenden von generierten Tests effizient umgehen?

Des Weiteren haben die bisherigen bezahlten POCs ergeben, dass generierte Tests idealerweise schon direkt, d.h. während der Entwickler Code schreibt, erzeugt werden sollten.

Endberichtkurzfassung

Symflower generates unit and integration tests for Java projects through the application of symbolic execution. Within this project we reached the following results:

Integration into existing developer workflows

With supporting all major editors, operating systems and architectures, we were set for a major goal of this project: the integration into existing developer workflows. Since there is no single workflow for developers, we distilled all workflows down the most important directions where generated tests can help:

Write tests before implementing (especially important for TDD)
Write tests after implementing
Highlight problems through tests while implementing

We succeed in supporting all these directions by implementing the following approaches:

Generate test suites and test templates through shortcuts, actions, context menus and scripts
Generate single tests through a `create test` code lens (directly inside of the source code)
Duplicate single tests through a `duplicate test` code lens
Filter and select generate tests through a `add test` code lens
Improve maintainability by adding tests directly into existing test suites
Highlight problems with wiggle lines directly inside of the source code
Detect existing coding conventions and configurations and apply them for analyzes and generations

Support the versatile Java ecosystem

Another major goal of this project is to support the versatile Java ecosystem. In the first year we most notably accomplished to lay a foundation for all future additions by implementing a flexible configuration store with layering, grouping, hierarchies and attributes that can deal with every source and combination of configuration. To evaluate this configuration store multiple basic components were added for dealing with builds and dependencies of the most common Java tooling `Maven` and `Gradle` as well as the test frameworks`JUnit 4` and `JUnit 5`. All of them apply to a majority of projects of the Java ecosystem. Additionally, we were able to introduce basic support for all necessary environmental details to allow the evaluation of workflows of software developers that use the Java application framework “Spring”. Specifically we were able to do a vertical slice of Spring controller integration tests which have one of the highest complexity in the whole Java ecosystem.

Projektkoordinator

- E&M Software Service GmbH

Projektpartner

- Software Competence Center Hagenberg GmbH