Bledea Mihaela Alexandra Boancă Tudor Bretan Cezar Alexandru

EvoMaster: Evolutionary Multi-context Automated System Test Generation

https://ieeexplore.ieee.org/document/8367066

EvoMaster was created due to the lack of available tools for white-box system testing of enterprise applications, particularly RESTful web services. Most of the work done so far in literature has been concentrating on black-box testing. The strong desire to develop such a tool was the driving force behind the creation of EvoMaster. The tool consists of two main components: a core process that handles the main functionalities and a drive process. It supports multiple search algorithms for generating test suites, with the MIO (Many Independent Objective) algorithm being the default one. Each test will be composed of one or more HTTP calls and the generated test files are self-contained, as the EvoMaster driver can automatically start the System Under Test (SUT) before executing the tests. The core process is written in Kotlin, a language that can compile into JVM bytecode.

Numerous automated tools are available that can generate unit tests for various types of applications, including smartphone apps and HTML pages. However, most of these tools are black-box in nature, meaning that they do not consider the internal workings of the server-side code. Currently, only a limited number of white-box testing tools are available for download and use, which is why EvoMaster seeks to provide developers with these features. EvoMaster is designed to generate tests at the system level using evolutionary techniques, specifically the MIO algorithm. Its current focus is on RESTful APIs that run on JVMs. The tool also aims to support search-based software testing for databases, which is currently under development. As it is said in the paper, there is no available white-box tool that addresses enterprise/web applications, so what it brings new, is the feature of generating white-box, system-level test cases for those.

EvoMaster underwent validation by assessing its performance on three RESTful APIs, including two open-source APIs and one provided by an industrial partner. These APIs ranged in size from 2,000 to 10,000 lines of Java code. During the evaluation, the tool uncovered 38 unique bugs, resulting in server errors and internal crashes (HTTP 5xx codes) due to the way HTTP calls were generated. However, the statement code coverage for the tested SUTs was only between 20% and 40%.

1. **Approach and motivation**
2. **Aim and the novelty**
3. **How the tool was validated**