**Test case generation method for Go language**

https://ieeexplore.ieee.org/document/7936140

The research paper proposes an approach for generating test cases with code coverage adequacy in Go language using a Genetic Algorithm (GA). The developed tool, called Goceng, consists of four main components: an analyzer, test data generator, test code generator, and executor. The analyzer collects information from the source code, such as function declarations, imports, types, and constants. The test data generator generates input values based on the collected information, and the test code generator produces the test code including the invocation of the function under test. The executor runs the generated test code using the go test tool to collect coverage profiles. The researchers evaluated the approach on eight different Go source codes and found that it facilitated the process of unit testing by reducing the effort and time required for test case generation. In what concerns the experiments, they performed 10 times generation at maximum crossover duration time (5 minutes). The same process was repeated for the other 7 source codes. The paper discusses the design and implementation of Goceng, as well as the experimental results demonstrating its applicability in achieving full code coverage. However, it also mentions some weaknesses, such as the difficulty in determining which paths will be covered by the generated test cases. Overall, the approach shows promise in automating test data generation and code coverage in Go language, with potential for further enhancements and applications.

# RESTest: automated black-box testing of RESTful web APIs

<https://doi.org/10.1145/3460319.3469082>

The paper introduces RESTest, a black-box testing framework for RESTful web APIs. The framework supports the generation of test cases using various techniques, including fuzzing and constraint-based testing. RESTest takes the API specification in the OpenAPI Specification format as input and generates abstract test cases that can be transformed into executable test cases for specific testing frameworks and programming languages. The paper presents the architecture and workflow of RESTest, describing its components such as test data generators, test case generators, test case mutators, test writers, and test runners. The framework was evaluated through offline and online testing scenarios. In offline testing, RESTest outperformed random approaches in detecting failures by leveraging constraint-based testing. In online testing, RESTest continuously tested multiple APIs, generating thousands of test cases and uncovering failures related to server errors, client errors, and inconsistencies between the API specification and implementation. The paper highlights the effectiveness and potential of RESTest in testing real-world APIs and suggests future work on expanding the framework with additional test data generators and generation strategies.

# An Approach of Automated Test Cases Generation in Database Stored Procedure Testing

<https://ieeexplore.ieee.org/document/5458953>

This paper proposes an approach that combines white-box testing with black-box testing for testing database stored procedures. The approach utilizes path coverage to select a set of testing paths and generates test cases using program slices. The authors constructed a constraint system for test data generation by extracting and replacing predicates in the program slices. They used Z-path coverage testing and depth-first search algorithm to generate test paths for the stored procedures. The constraint system was solved using a genetic algorithm to obtain test data that meets the constraints. The experiments conducted showed that the approach can efficiently generate test cases and achieve path coverage testing for database stored procedures. In the provided excerpt, the results of the experiments are briefly mentioned. The system takes input in the form of information about the database stored procedure to be tested and the desired number of test cases to be generated. The output includes formatted test case files, database states, and system execution logs.

The experiments conducted on MySQL stored procedure testing involved generating automated test cases for 80% of simple stored procedures (up to 20 lines without complex logic). After running the tests using the generated test cases, it was found that 70% of the test cases were executable and provided coverage for general path testing. This approach demonstrated improved efficiency and increased test coverage compared to manual test case generation. However, the experiments also identified some limitations. The approach failed to cover the entire paths within loop bodies, requiring additional manual work to add test data. Additionally, compatibility issues with other testing tools, such as integrating with test data generated by existing tools, were observed. The conclusion acknowledges the need for further research and development in automated test case generation for database stored procedure testing, including addressing challenges related to loop bodies, system and user functions, dynamic test case generation, and more.

# EvoMaster: Evolutionary Multi-context Automated System Test Generation

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

# RESTful API Automated Test Case Generation

https://ieeexplore.ieee.org/document/8009904

<https://ieeexplore.ieee.org/document/7890602>

<https://ieeexplore.ieee.org/document/7102595>

<https://ieeexplore.ieee.org/document/9159077>