International Journal of Scientific Research and Engineering Development-–Volume 4 Issue 3, May-June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Dynamic Code Coverage

Keyur Shah, Dr. B. Sathish Babu
(Computer Science and Engineering, RV College of Engineering, Bangalore
Email: keyurshah.cs17@rvce.edu.in)
(Computer Science and Engineering, RV College of Engineering, Bangalore
Email: bsbabu@rvce.edu.in)

----------**********----------

Abstract:

In today's digital world, with rapid growth in the number of components developed for a single software platform, innumerable lines of code are written. Testing such components is quite an arduous task. Testing an application is an integral part of the development cycle. Following up with the behaviour of code in different environments pile up to this already arduous task. Code coverage analysis helps the tester to find the redundancy or efficacy of these written test suites with the help of visually interactive reports.

This paper discusses how JaCoCo can facilitate the building of code coverage reports with outstanding features. It discusses the challenges addressed during the integration of it. Also, the use of RestFul APIs to facilitate the delivery of such reports to the desired user and providing platform-independent solutions with the help of docker.

Keywords — **JaCoCo, software testing, report, API, platform-independent, code-coverage, docker.**

----------**********----------

I. INTRODUCTION

Software testing is an important activity for supporting the full Software Development Life Cycle and is a vital aspect of software quality. SDLC stands for Software Development Life Cycle, and it refers to the steps involved in researching, designing, implementing, and maintaining an application. Quality Assurance (QA) is an important component of the SDLC in any development business. QA is used to analyse an application's quality-related elements and to validate its behaviour in various system setups. Grasping with the benefits of testing can assist development businesses in focusing on day-to-day tasks rather than application issues. Testing is usually useful in determining if entire software requirements are implemented appropriately, i.e if they are implemented in accordance with the established requirements. It aids in the identification of defects/bugs and ensures that they are identified/addressed prior to the software deployment stage and also shows that the software/application appears to be operating according to specifications and that the established performance criteria are satisfied. The most important reasons for the importance of testing in the SDLC is that it always aids in the verification of appropriate integration and interaction of each component in the system.

II. MANUAL TESTING VS AUTOMATED TESTING

There are two types of testing: manual testing and automated testing. Human testers manually examine codes and faults in software during manual testing. Automated testing, on the other hand, is the process of executing a system using computer programmes. Automation testing is the practice of performing test cases by repeating pre-defined

operations utilising tools, scripts, and software. The goal of test automation is to replace manual human work with automated methods or equipment. Because automated testing is carried out with the help of a tool, it takes less time to conduct exploratory tests and more time to maintain test scripts while also enhancing overall test coverage. Manual testing has the advantage of allowing a human mind to extract insights from a test that an automated testing machine would overlook. For huge projects that require testing the same regions, again and again, automated testing is the best option. Now in order to evaluate the automated test cases and their efficacy, multiple metrics are adopted, one of them is code coverage.

III. CODE COVERAGE

Code Coverage refers to the number of lines, methods, or classes of code that are executed during an automated test run. The term "dynamic code coverage" refers to evaluating the test suite's coverage during the program's execution phase. It takes into consideration components that interact with the application during execution, such as a SQL query or an API call. Code Coverage is used to observe if the test suites were able to cover the complete application or whether there were any missing branches or sections that were left untouched during the process run. It's a crucial indicator for determining test suite efficacy and redundancy. It also encourages a higher percentage of coverage, ensuring that only high-quality code is released to production. Low Dynamic Code Coverage might lead to more unexplained production difficulties, and vice versa. As a result, the coverage report helps to reduce the chance of problems being detected later in the development process.

IV. STATE OF THE ART DEVELOPMENT

The readily available solutions over the internet require in-depth knowledge of the application to be able to implement coverage solutions. It is a time-consuming process and hence requires a lot of effort.

A. RESTful API delivery System

A RESTful API is a design approach for an application programme interface (API) that accesses and utilises data via HTTP requests. That information may be utilised with the data types: PUT, DELETE, POST and GET which correspond to activities such as reading, modifying, creating, and removing resources. A RESTful API, also known as a RESTful web service or REST API, is based on representational state transfer (REST), a communication architecture style and technique popular in web service development. This REST API helps us to issue a GET request to the containerized application to generate the reports and fetch the reports.

B. Docker Shell interactive modules

The docker container envelopes the resources required to run the docker image. After this container is deployed, the only way to interact with this container is through the docker shell. As a part of the project, some modules need to be developed to interact with the application that is within the docker container.

V. OVERVIEW OF VARIOUS CODE COVERAGE TOOLS

OpenClover

A powerful, free Java code coverage suite with over 20 criteria to define the product's reliability. It has all of the same features as the original Clover (the server edition). Developers who worked on Clover from 2012 to 2017 are leading the OpenClover project. OpenClover is a source code instrumentation tool that supports Java, Groovy, and AspectJ. Fine control over the breadth of coverage measurement, test optimization, and complex reports are only a few of its characteristics. Ant, Maven, Gradle, Grails, Eclipse, IntelliJ IDEA,

Bamboo, Jenkins, Hudson, Griffon, SonarQube, and AspectJ are all supported by OpenClover.

JCov

JCov is a tool that has been created and used with Sun JDK (and later Oracle JDK) since version 1.1 of the Java language. JCov is a Java code coverage tool that may be used to measure and report on Java code coverage. The GNU Public License governs the distribution of JCov (version 2, with the Classpath Exception). In 2014, JCov became open-source as part of the OpenJDK code tools effort.

Serenity

Serenity is a free toolkit for calculating and reporting Java code coverage. Major code metrics are examined in addition to coverage, including cyclometric complexity, stability, abstractness, and distance from the main. The data from the report is saved in an object database and made available using Jenkins/Hudson. Visually, the interface resembles that of the Eclipse IDE.

Cobertura

Cobertura is a free Java tool that determines how much code is visited by tests. It can help to figure out which areas in the Java software don't have enough test coverage. It is built on top of jcoverage.

Testwell CTC++

It is a platform that helps to build workflows and apps at a faster rate with features like a visual no-code designer, powerful low-code tools, beautiful UX and collaborative development. Build once deploy anywhere feature by designing app once and deploying it anywhere. It takes more time for processing larger amounts of data and therefore less suitable for that. Testwell CTC++ is a C, C++, Java, and C# code coverage tool. This tool was first developed in 1989 at Testwell in Finland. Verifysoft Technology, based in Offenburg, Germany, has been providing support and development since 2013. All levels of code coverage are analysed by Testwell CTC++, up to Modified condition/decision coverage and Multi condition Coverage. All compilers are supported by the tool.

Parasoft Jtest

Parasoft Jtest is one of the testing tools suite's products. Jtest enables the user to develop Java-based apps faster with less risk, better assistance, and analysis. It's used for manual and automated testing, as well as unit testing and code coverage. Its report gives a clear image of the code that has been covered, reducing risks.

Gretel

Carls Howells of the University of Oregon created Gretel, a free code coverage tool for Java. BCEL (Byte Code Engineering Library) is required to instal Gretel JVM 1.3 or later. It also has the unique capability of removing coverage instrumentation for code that has previously been covered, resulting in improved performance.

VI. RESEARCH METHODOLOGY

Existing coverage solutions require the configuration of individual projects from scratch and mostly platform dependent solution, in some cases even dependent on the IDE they are implemented. These solutions require that the user be completely familiar with the component on which these solutions will be implemented. Therefore, an easy-to-implement and operate methodology is required which can generate dynamic coverage reports without hindering the execution of ongoing application/process.

The project deals with the design and development of a component that generates coverage reports without hindering any of the ongoing processes. Any user can generate these reports and can obtain a structured form of these reports in a zip format. The application uses a microservice architecture to separate the different modules of the application hence allowing independent access to each user without

interference from the work being performed on any other module. The thorough summary generated after each analysis is deployed can be used by the experts to determine the efficacy of the test cases. The whole system is built in Java using the STS IDE.

The project has 2 crucial objectives:

- Build a platform-independent solution.
- Build a concise, visually interactive format of report that can be analysed by the user.

The project methodology consists of the following steps:

STEP 1: Identifying files that require dynamic code coverage.
STEP 2: Identifying component that is utilized for comparison between dead code and active piece of code.
STEP 3: Constructing REST methods to invoke report generation.
STEP 4: Retrieving methods that procure the generated method.

- API Design specification using core JAX-RS API.
- Creating an artifact id for the API which can be incorporated as dependencies in the pom.xml files.
- Building jar files for the project directory and containing them to a docker image.
- Building and deploying Docker container.
- API endpoints interact with the jacocoagent *.jar files, extract source and class files for target repository and create a report based on sourcefiles and target classfiles.
- The above process requires interacting with shell (to create an intermediate executable), StringBuilder is implemented to interact with shell via java.
- Automating deployment of containers as microservices via Jenkins.
- On event when endpoints are hit, the reports generated via the *.jar files must be zipped and stored on the local machine of the user as well as the in the container.

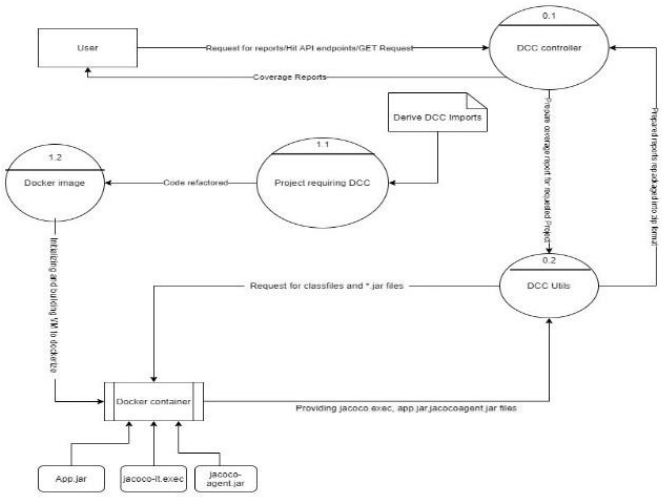

Figure 1: Overall Architecture of an Application

VII. LOW-LEVEL ARCHITECTURE DESIGN

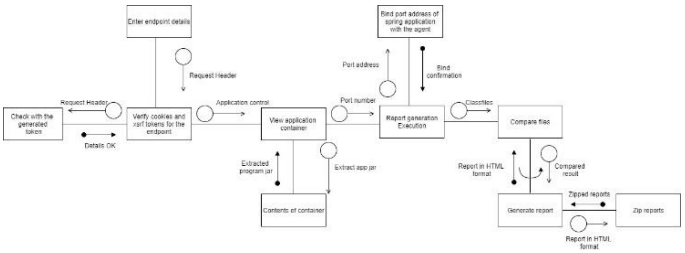

Figure 2: Structure chart for Dynamic Code Coverage

The structure chart, as seen in figure 2, begins with the User hitting the API endpoints. The user is validated after the user's login information matches the data in the White staff table in the database since the application is accessible only by authorised employees. The user must address the appropriate tokens to access the tools. The tokens used for security and authentication are cookies and SSRF. Once the details are verified, application control is handed over to the DCC controller, further it's transferred to the application container. The CLI utils access the docker container and extract the application jar files. These jar files are delivered along with the jacoco lib jar files. The

system issues a port number for output binding to the TCP server. In case if the port is already issued to another process, the application fails and a ticket has to be raised. Now, on successful binding, the DCC utils utilize the extracted application jar files contents and fetch the source and target class files. These are compared with the help of an executable called jacoco-it.exec. This is generated with the help of jacocoagent.lib and jacococli.lib. The jacococli.lib bridges the interaction between the ports and agents hooks provided by the jacocoagent.lib during the on-the-fly instrumentation. The reports are then generated. These reports are visually interactive and provide a detailed analysis of code coverage. These reports are henced delivered to the requested user in a zip format that can be stored on the local machine.

VIII. CONCLUSION

Coverage reports play an essential role in accessing the system correctness and test case efficiency with respect to the implemented codebase. This project achieved the initial objectives of the realization of a visually interactive report that can be accessed by the user to analyse the amount of dynamic code coverage. The component build is independent of the OS platform is deployed and delivered. The JaCoCo tests do not take long at all to generate as they are created while the unit tests are running. By taking a few minutes to incorporate DCC user can make the code better covered and tested. Keep in mind though, 100% code coverage does not necessary reflects effective testing, as it only reflects the amount of code exercised during tests.

IX. FUTURE SCOPE

The scope of every project is limited but can be extended with future improvements. Based on the limitations in the first build of the system, one can jot down possible methods or techniques or algorithms to overcome them in the next build. This will ensure that the product can be made commercially standard and of market quality. The following are possible future enhancements for the system –

- A mechanism to provide line level coverage report instead of method level coverage.
- Implementation of the same mechanism to support applications built in different languages.

REFERENCES

[1] H. Cox, "Differential coverage: : automating coverage analysis," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 424-429, doi: 10.1109/ICST49551.2021.00054.

[2] Baeldung. 'Intro to JaCoCo', July 30, 2020. https://www.baeldung.com/jacoco

[3] E. Viglianisi, M. Dallago and M. Ceccato, "RESTTESTGEN: Automated Black-Box Testing of RESTful APIs," 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, pp. 142-152, doi: 10.1109/ICST46399.2020.00024.

[4] Kinnary Jangla. 'Accelerating Development Velocity Using Docker: Docker Across Microservices', 2020, 3rd Edition, ISBN-13 (pbk): 978-1-4842-3936-0.

[5] S. Mysari and V. Bejgam, "Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-4, doi: 10.1109/ic-ETITE47903.2020.239.

[6] K. Guntupally, R. Devarakonda and K. Kehoe, "Spring Boot based REST API to Improve Data Quality Report Generation for Big Scientific Data: ARM Data Center Example," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 5328-5329, doi: 10.1109/BigData.2018.8621924.

[7] Andre Arcuri. 'Journal first presentation of an experience report on applying software testing academic results in industry: we need usable automated test generation'. In Proceedings of the 40th International Conference on Software Engineering (ICSE '18). Association for Computing Machinery, 2018, New York, NY, USA, 1065. DOI:https://doi.org/10.1145/3180155.3182555.

[8] Ripon K. Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R. Prasad. 2018. Bugs.jar: a large-scale, diverse dataset of real-world Java bugs. In Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18). Association for Computing Machinery, 2018, New York, NY, USA, 10–13. DOI:https://doi.org/10.1145/3196398.3196473.

[9] Isha, A. Sharma and M. Revathi, "Automated API Testing," 2018 3rd International Conference on Inventive Computation Technologies (ICICT), 2018, pp. 788-791, doi: 10.1109/ICICT43934.2018.9034254.

[10] Amanda Schwartz, Daniel Puckett, Ying Meng, Gregory Gay, 'Investigating faults missed by test suites achieving high code coverage', Journal of Systems and Software, Volume 144, 2018, Pages 106-120, ISSN 0164-1212, <https://doi.org/10.1016/j.jss.2018.06.024>.

[11] C. Chang and N. Lin, "Constraint-Based Test Case Generation for White-Box Method-Level Unit Testing," 2016 International Computer Symposium (ICS), 2016, pp. 601-604, doi: 10.1109/ICS.2016.0123.

[12] B. I. Ismail et al., "Evaluation of Docker as Edge computing platform," 2015 IEEE Conference on Open Systems (ICOS), 2015, pp. 130-135, doi: 10.1109/ICOS.2015.7377291.

[13] L. Ma, C. Zhang, B. Yu and H. Sato, "An Empirical Study on Effects of Code Visibility on Code Coverage of Software Testing," 2015

IEEE/ACM 10th International Workshop on Automation of Software Test, 2015, pp. 80-84, doi: 10.1109/AST.2015.23.

[14] D. Tengeri, Á. Beszédes, T. Gergely, L. Vidács, D. Havas and T. Gyimóthy, "Beyond code coverage — An approach for test suite assessment and improvement," 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015, pp. 1-7, doi: 10.1109/ICSTW.2015.7107476.

[15] L. Pierce and S. Tragoudas, "Unreachable code identification for improved line coverage," Sixteenth International Symposium on Quality Electronic Design, 2015, pp. 345-351, doi: 10.1109/ISQED.2015.7085450.

[16] P. S. Kochhar, F. Thung and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015, pp. 560-564, doi: 10.1109/SANER.2015.7081877.

[17] S. Raemaekers, A. van Deursen and J. Visser, "The Maven repository dataset of metrics, changes, and dependencies," 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 221-224, doi: 10.1109/MSR.2013.6624031.

[18] Alexandre Perez. Dynamic Code Coverage with Progressive Detail Levels. , 2013, CoRR, abs/1306.4546.

[19] H. Li, "RESTful Web service frameworks in Java," 2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), 2011, pp. 1-4, doi: 10.1109/ICSPCC.2011.6061739.

[20] S. S. Gokhale and R. E. Mullen, "Dynamic code coverage metrics: a lognormal perspective," 11th IEEE International Software Metrics Symposium (METRICS'05), 2005, pp. 10 pp.-33, doi: 10.1109/METRICS.2005.17.

