Automated testing platform and lean principles for successful adoption

Uday Jain
Department of Software Engineering
Blekinge Institute of Technology
Karlskrona, Sweden
uday9k@gmail.com

David Fucci
Department of Software Engineering
Blekinge Institute of Technology
Karlskrona, Sweden
Davide.fucci@bth.se

***Abstract*— Lean principles have become an integral part of product development culture within organizations. Evolving open-source community has made it possible for teams to exploit available resources and setup a fully functional automation testing platform that is closely integrated with CICD pipelines without major effort investments. In this paper we discuss one such product stitched using open-source components. We further discuss the advantages and challenges one would face once implementing such a platform in an industrial setting and how lean principals when collaborating with developers can help transform the platform into a more modular and extensive tool that is customized to organizational needs.**

Keywords—Lean principles, Automated testing, Docker, Open-source technology stack, OpenClover, Pitest

I. INTRODUCTION (*HEADING 1*)

Agile methods of software development and testing have been the corner stone for most industrial settings since early 2000s. They have showcased significant efficiency gains in product development especially with rapidly evolving products [1]. However, gradual and industry wide progression towards lean is observed with time [2] [3], owing to major efficiency gains [4] across the entire product lifecycle that stiches together design, development, deployment and testing into small feedback loops when compared against a non-lean product development team. Recent work has shown application of lean principals and practices can provide efficiency gains even in non-software development areas [5].

Empirical studies suggest test form an integral and crucial part of the product development lifecycle and presence of test contribute towards performance gains when using test driven development (TDD) [6]. With growing interest in the area, testing suits have seen major development and a continuous rise in complexity as new testing techniques and evaluation test suits are researched [7] [8] [9] . The rise in complexity has in turn paved the way for development of automated testing and associated tools. Studies have shown automated test suits to provide major benefits through test reusability, repeatability, test coverage and test execution time [10] [11].

As the efficiencies of automated testing were realised [11] [12], the testing community saw a major push towards development of both paid as well as open-sourced tools [13] and architecture [14] to support rapid adoption within the industrial setting [15] [16]. Recent year have seen tight integration of such tools directly with continuous integration and continuous delivery (CICD) [17] to deliver incremental value through automated testing. Advancements in technology solutions and available software solutions today have reduced the initial effort required for teams to adopt automated testing as part of product development lifecycle [18]. Teams can now utilise services such as docker containers and Kubernetes to optimise resource utilization [19] when running large test suits such as system integration tests [20].

As part of this paper we have shown how one can setup a fully functional automated test suit with little resources in terms of both effort and computational power using open sourced tools available. The shown environment can effectively run on a personal laptop and uses open-sourced test automation tools such as Bitnami [21] to host Jenkins [22] server coupled with multiple open-sourced plugins to evaluate relevant performance metrics and produce desired reports. In this paper our intent is to provide a preview of such a setup, exploiting the vast open-sourced offerings to instrument a minimum viable product (MVP). The MVP is containerized and allows for further customization and scalability. We also discuss the benefits and challenges one face when setting up such a product and how it can be improved with additional functionality using a modular approach to design. The work also provides details into how such a system would work in an industrial setting exploring lean principals and its dependence on people and processes for success.

Collaborating with developers in an industrial setting, we found metrics such as code coverage and mutation testing can provide crucial insights for the development and testing team(s). These testing techniques are found to highly correlate with test quality across multiple experimental setup [16] [23] [8] and are crucial to improving quality of test suits, detecting bugs and improving code maintainability [24]. Given the vast availability of associated tools, choices are often individualistic and dependent on level of popularity, configurability and usability within the existing technology stack.

The paper further stresses the importance of processes and collaboration among developers in the industrial setting to minimize constraint on resourcing needs as the environment scales to multiple products and test suits grow over time. Minimizing waste through test suit optimization can help get additional gains to constrain the speed of resource consumption [25]. Multiple test suit optimization techniques such as those using the artificial bee colony algorithm or the cuckoo search algorithm have been evaluated and shown to have significant impact on resourcing needs of product teams [26]. The choice of optimization technique could however depend heavily on individual use cases and test suit structure [27] [25].

II. Related Work

Modern day software development cycles often include small iterative cycles of continuous development and testing. While agile development practices are efficient to organise the development lifecycle, they may suffer from insufficient ways to address design [28]. Both design and development are iterative in nature and continuous care is required to closely monitor integration, making the process fragile. Since lean principals integrate design, development, deployment and testing into a single feedback loop, efficiently integrating agile practices with lean principals into software development and testing can provide continuous value generation across the whole product lifecycle[3] [29] [30].

Recent literature also suggests the claimed benefits of TDD may not be due to its distinctive test-first dynamic, but rather because TDD like processes encourage fine grained, steady steps that improve focus and flow [6] [31]. Several evidence are available to understand efficiency of testing systems given goals and constraints [11], establishing the precedence to automation [10].

With realisation of testing as a part of software development cycle and integration into industrial process, technology stacks have evolved [20] [19] to integrate CICD with testing as a service [17] [32], pushing the limits on time to delivery. Rapid development in open-sourced tools in recent years has made it easier to combine multiple components offered by the community and stich together a comprehensive automated testing environment customised to individual use cases [21]. Integration techniques which were development intensive [33] have now become available to use out of the box [22]. This rapid development has paved the way for automated testing [8] [15] and its further optimizations through multiple types of test case prioritization, test suit minimisation and test suit selection techniques [26] [34] [14].

As a part of this paper, we have used mutation testing along with code coverage reports to evaluate the strengths of our testing suit. Multiple studies have indicated use of coverage metrics and its variants as an indicator of test suite quality [23] [35]. With high popularity among open source community, mutation testing has also proved to play a significant role in identifying defects and help improve test suit quality [24] [16]. Pitest & OpenClover have been identified as a popular open source Jenkins plugin that provide easy to setup mutation testing and code coverage analysis for maven project [36] [37].

III. Methodology

To build a customized environment automating testing of software updates we started with selection of KPI (key performance indicators) that we want to test the performance with each of the commits to GitHub repository. As a part of this exercise, we started with understanding the process of development and different types of tests written. We then evaluated inputs from developers and testers along with product managers (PM) to understand the KPIs that would provide most meaningful inputs for further development, iterations, over the lifecycle of the product. To optimize the metrics, we considered the ones that provide nonoverlapping information and ones that were identified by more than 3 developers in a squad of 5 developers. The goal is to provide simple reports that cover most use cases across the teams.

Leveraging the small size of the team, we conducted interviews and brainstorming sessions to identify the following list of KPIs:

- %Code Coverage
- %Test passed
- % Mutation Coverage
- % Test Strength

Based on the identified KPIs we then outlined most effective tools that generate such reports. Effectiveness was estimated based on following parameters:

- Open-Source availability
- Community support and active contributors
- Frequency of Commits
- Issues identified
- Ease of implementation
- Level of detail

To identify an appropriate technology stack, we outline following parameters:

- Scalable: both horizonal and vertical scalability in terms of computational and storage resources
- Ease of customization
- Compatibility with available tools and resources
- Ease of implementation and iterative development
- Popularity among developers and testers

Since GitHub was the preferred choice to store code base, among the contenders, a docker based Jenkins container that connects to GitHub was found most suitable. Additionally, GitHub can easily connect to any IDE (Integrated Development Environment) that the developers choose to use, a platform for automated testing using the technology stack required little changes to the existing ways of working within the team.

Bitnami offers open-sources Jenkins based docker images that can be customized to suit the needs of the organization. To choose additional plugins based on our identified KPIs, prioritization using parameters listed above was done. We started development using Pitest, OpenClover and Test Result

Analyzer. These plugins efficiently capture the required metrics and provide a detailed report that can be further customized to individual needs. Figure 1 captures the environmental setup of the automation testing platform MVP used as a part of the exercise.

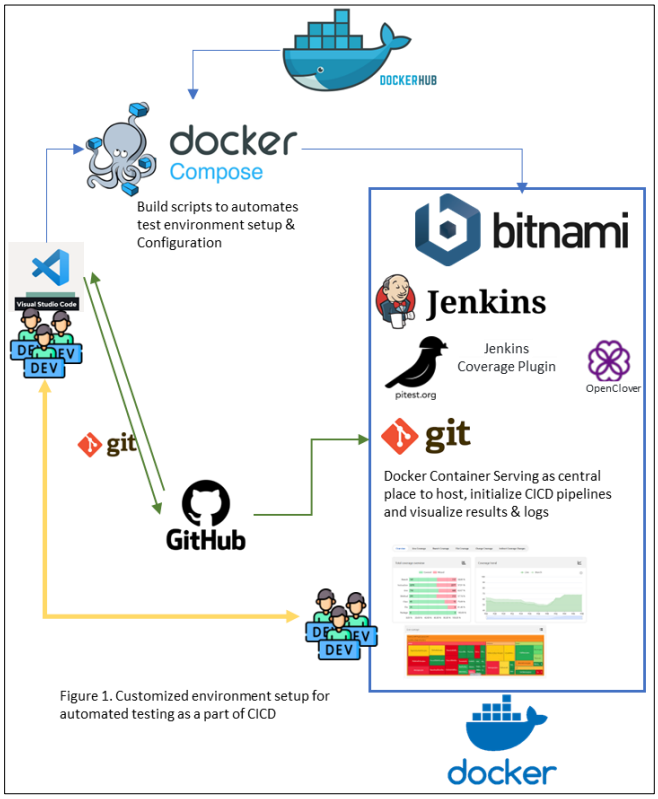


Figure 1. Customized environment setup for automated testing as a part of CICD

The platform (Figure 1) provides extensive flexibility to develop additional customizations and modularizations to ensure stability and security among the systems. Figure 2 outlines the platform vision with its components to be developed as the need and use grow over time.

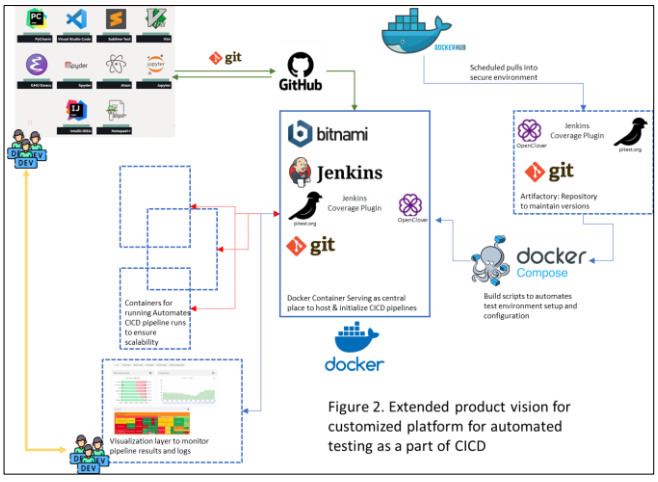


Figure 2. Extended product vision for customized platform for automated testing as a part of CICD

All tests runs were performed on a single laptop with 32GB RAM (16GB available to container running) and 8 processor (4 core available to container running) to ensure comparison among results. To minimize computation time with each build used "DwithHistory" as a default option when running Pitest. To test the environment we used open-source project Joda-Time committed to GitHub repository here.

IV. Results

To build the environment using a docker compose file it took a total of 25 minutes once the images were downloaded. However, to identify right set of combination for package and plugin versions and respective configurations, it took more than 30 iterations.

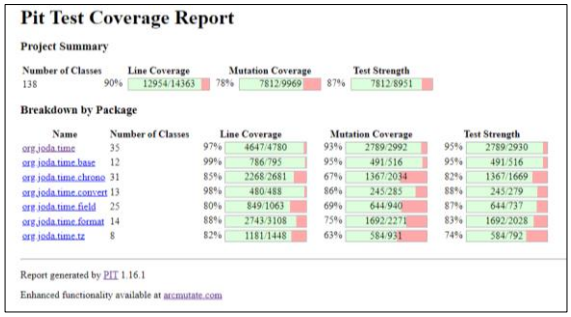
Building the project and running all tests (4212 tests in total, 100% passed) took 9 minutes and 14 seconds when complied to publish OpenClover reports. The build running PIT mutations to publish both OpenClover reports and PIT mutation reports however took 1 hour and 35 minutes of which mutation analysis took 1 hour and 30 minutes. During the run 9969 PIT mutation were generated and 7812 or 78% mutations were killed using existing tests. Among them 1018 mutation had no coverage thereby providing a test strength of 87%. In total 118,302 tests were run (approximating 11.87 tests per mutation) and a line coverage of 12954/14363 (90%) was achieved for mutated classes.

During exploration 2 sets of tests cases were used:

- With all test (*org.joda.time.TestAll.suite*, *org.joda.time.chrono.TestAll.suite*, *org.joda.time.chrono.gj.TestAll.suite*, *org.joda.time.convert.TestAll.suite*, *org.joda.time.field.TestAll.suite*, *org.joda.time.format.TestAll.suite*, *org.joda.time.tz.TestAll.suite*)
- With only *org.joda.time.TestAll.suite* test.

Changing the number of test suits had little impact on the total run time.

Figures 3 & 4 capture the results of Pit Test while Figure 5-8 capture OpenClover results for built with all test cases and Figure 9-12 capture OpenClover results for built with only *org.joda.time.TestAll.suite* test cases.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
138	90% 12954/14363	78% 7812/9969	87% 7812/8951

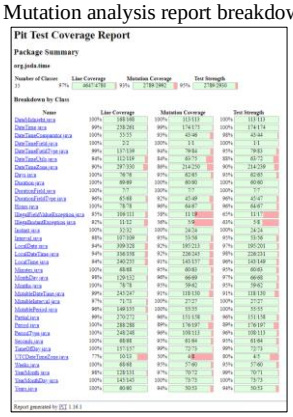
Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
org.joda.time	35	97% 4647/4780	93% 2789/2992	95% 2789/2930
org.joda.time.base	12	99% 786/795	95% 491/516	95% 491/516
org.joda.time.chrono	31	85% 2268/2681	67% 1367/2034	82% 1367/1669
org.joda.time.convert	13	98% 480/488	86% 245/285	88% 245/279
org.joda.time.field	25	80% 849/1063	69% 644/940	87% 644/737
org.joda.time.format	14	88% 2743/3108	75% 1692/2271	83% 1692/2028
org.joda.time.tz	8	82% 1181/1448	63% 584/931	74% 584/792

Report generated by PIT 1.16.1

Enhanced functionality available at arcmutate.com

Figure 3. PIT Mutation analysis report breakdown by Package



Pit Test Coverage Report

Package Summary

org.joda.time

Figure 4. PIT Mutation analysis report breakdown by Class eg. org.joda.time

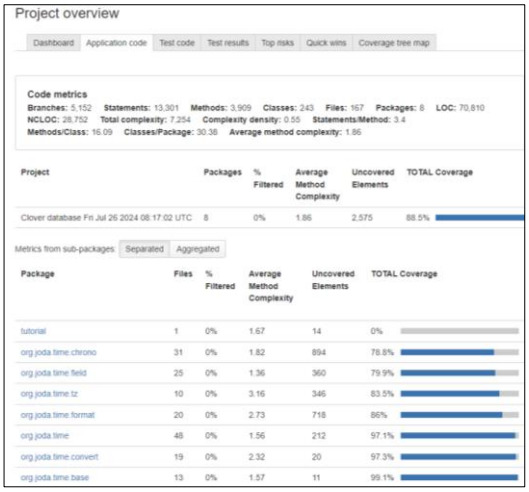


Figure 5. OpenClover report for application code coverage analysis overview when running all tests

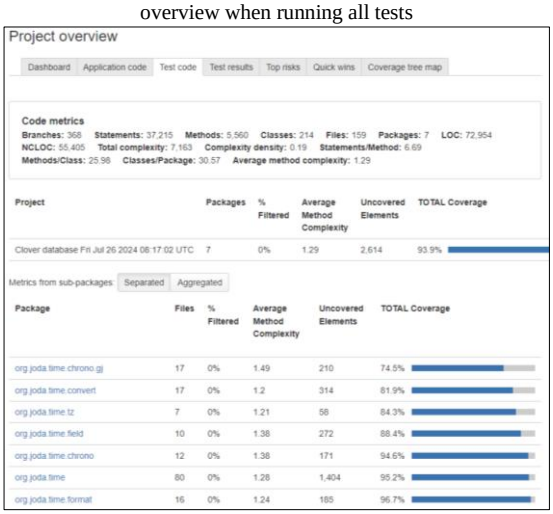


Figure 6. OpenClover report for test code coverage analysis overview when running all tests

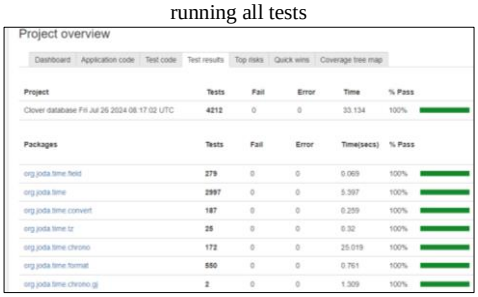


Figure 7. OpenClover report for test results when running all tests

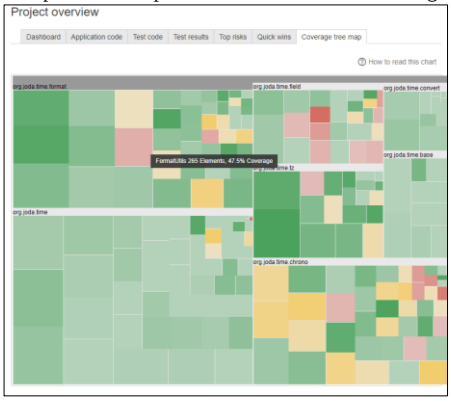


Figure 8. Coverage Tree map from OpenClover report when running all tests

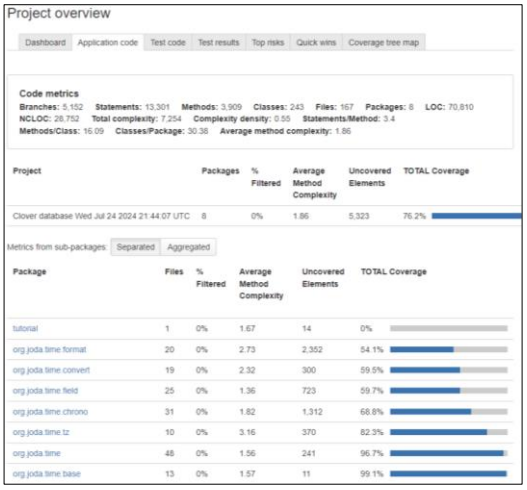


Figure 9. OpenClover report for application code coverage analysis overview when running only *org.joda.time.TestAll.suite()*

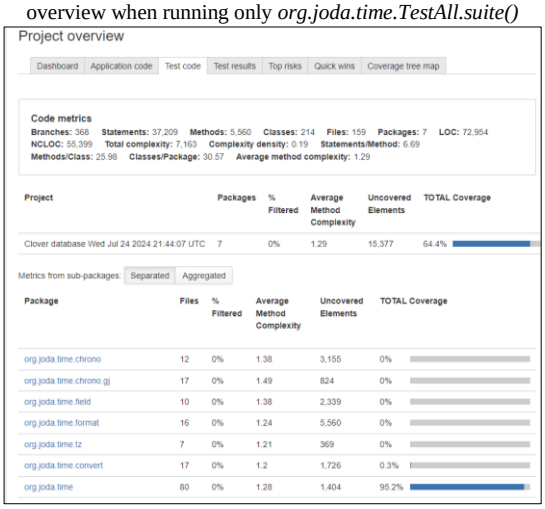


Figure 10. OpenClover report for test code coverage analysis overview when running only *org.joda.time.TestAll.suite()*

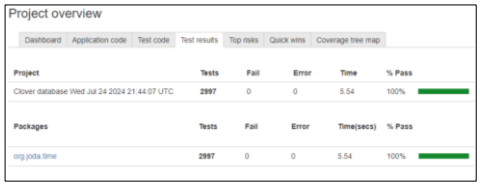


Figure 11. OpenClover report for test results when running only *org.joda.time.TestAll.suite()*

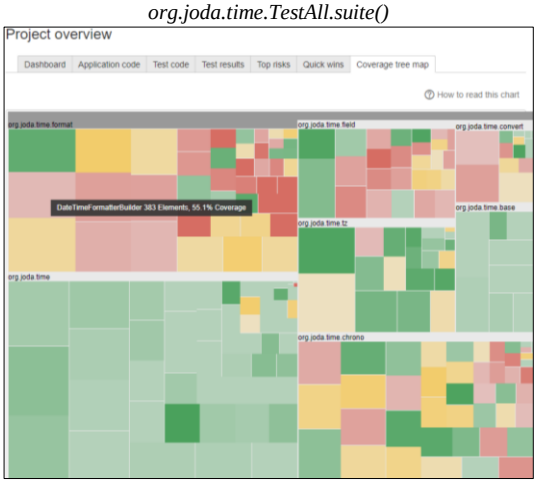


Figure 12. Coverage Tree map from OpenClover report when running only *org.joda.time.TestAll.suite()*

V. Discussion

The automated test environment provides a centralized solution to automate analysis of testing done as a part of product development. The environment encourages the developers to use key metrics with standardized definitions such as % Code Coverage, % Test passed, % Mutation Coverage and % Test Strength to evaluate their commits. This helps development team adopt principals of LEAN software into their Agile sprint cycles.

The generated reports provide quantitative (using summary metrics) and continuous feedback that is valuable to the development team as well as PMs when developing patches to existing software products. As observed from figures 10 and 6, the total test code coverage has improved from 64.4% to 93.9% once we included all test suites into our development code. Metrics like these provide crucial insights into quality of our code, providing continuous feedback. Furthermore, availability of such centralized and standardized metrics helps review code and enable face to face discussions within respective development teams to improve solutions providing customer value in terms of stability and minimized defect probability.

Careful choice of metrics and reporting formats can enable simple and scalable solutions to automated testing that can be adopted across the wider organization and development teams.

Detailed reporting views such as those represented in Figure 5-12 can provide insights into classes and packages that require attention enabling the team to practice continuous improvement overtime through summarized tracking across multiple builds. Using observations from tree map in figure 12, one can easily identify package (*org.joda.time.format*) to be prioritized for test cases to improve % test coverage. One can further identify class (DateTimeFormatterBuilder) that can have the biggest impact on the summary metric (% test coverage). The OpenClover report further provides details into subclasses as well as location within overall code to help development team get started, enabling members to be self-organized and independent.

PIT Mutation reports provide similar level of granularity and flexibility to deep dive, enabling teams to rally across summary metrics such as % mutation coverage and % test strength. This helps development teams to ensure not only test coverage but also strength of test suits to capture borderline scenarios. Like OpenClover reports the Pit Mutation Reports help users to deep dive into class level analysis to identify classes that require additional tests to ensure high test strength.

A combination of % test coverage and % test strength can help development teams track redundancy in test suits and minimize irrelevant test cases. This could potentially serve as a part of test suit optimization. This methodology coupled with other test suit optimization techniques such as test case prioritization, test suit minimization and test suit selection can play a crucial role into reducing resource requirements including both computational resources and time.

While the automated testing environment provides a scalable centralized solution that can be easily adopted by any development team and scale resources based on needs (owing to containerized solution), it may require additional levels of customization as practices and products evolve over time. Different teams may find other metrics more relevant given their stage in product development life stage. We also found both OpenClover and PIT Mutation Report to be very exhaustive and detail intensive, making it difficult to digest in a fast-moving product development cycle.

As a part of the minimum viable product, we excluded any limitations on summary metrics to mark build as a failure. This is representative of the difficulty one would face when orchestrating such conditions across a diverse set of products and teams. In practice, it may be rare to find all or many products at same/similar life stage and thereby making it difficult to impose thresholds on key metrics at scale and such thresholds needs to be customized for each project. To facilitate this process, as shown in Figure 2, we recommend isolating the visualization layer to provide higher level of flexibility in reporting and integrate reports directly into consumption layer of organization wide solutions.

During the exercise we faced many challenges identifying right combinations of plugin and package versions. This may remain a never-ending problem especially when using a diverse set of open-source solutions that are often released and developed at varied paces. To ease maintenance efforts, we recommend planning development of an antifactory store that stores relevant versions of the key components. This feature also helps maintain a closed system with high levels of security.

While the MVP uses a single docker container, different projects may need different levels of computation and storage resources. To efficiently manage the available resources a more distributed architecture is recommended. As shown in figure 2, we recommend splitting the main Jenkins control panel and project builds into separate containers, with build containers that spins up based on requirements.

While PIT mutation analysis provides a thorough analysis of the tests, it is very time consuming and has a major impact on both the computational resource requirements as well as the run time. The Pitest package allows customization of the analysis to our needs using many parameters such as withHistory, targetClasses, targetTests, excludedMethods, excludedClasses, excludedTests, excludedTestClasses and maxMutationsPerClass. The current setup ran out of resources and crashed when default and unrestricted configurations were used. Given the resourcing limitations of the system we used maxMutationsPerClass during our test runs to limit need for compute resources. Setting these values for all projects may be difficult and would still require participation from developers. Care must also be taken to ensure abuse of such levels of customizations since it may lead to biased top level summary metrics. There may be a need to establish standard recommendations to allow such customizations for individual projects and consequently adequate training within each product unit to familiarize developers with established ways of working. Similar customizations are also available within OpenClover and other packages.

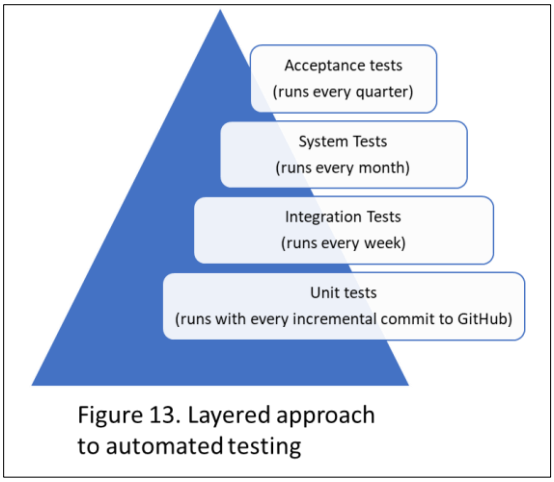


Figure 13. Layered approach to automated testing

Although not implemented and tested as a part of the process, based on the feedback from the development community, we recommend a layer approach to structing testing across the projects builds. Figure 13 represents a pyramid structure with multiple smaller unit tests that run with every incremental commit to the GitHub repository at the bottom and larger acceptance tests or system tests that run every month or quarter. Such levels of customizations can easily be achieved via splitting the project into multiple projects with each scheduled to run at required frequency. Each project can now be configured to produce relevant level of reporting, allowing us to track low level metrics such as % code coverage for unit tests while high level metrics such as average response time for system tests. A separate visualization layer, as shown in figure 2, would also allow us to extract relevant pieces across these projects to provide reporting suited at different levels organization and customer needs. This approach can also help us minimize repetitive test runs with every build and save on precious resources.

Segregating reports can help development teams structure regular discussions within different forums such as a squad level detailed discussion every sprint or a product level discussion among PMs and EMs (engineering managers) from different subproducts every quarter to share insights from integration and system wide tests.

Conclusion

Even with all issues one may face when scaling, owing to effort required to setup and maintain the environment for automated testing, it may provide high returns when released centrally and adopted across multiple development teams. Organisational structures can easily be integrated into the layered structure of project to ensure standardized reporting solutions well suited across the diverse forums. This enables teams to adopt Lean principals and influence sprint management based on quantitative metrics, reducing effort towards analysis of results and adding incremental customer value with each sprint cycle.

As explained previously, the use of docker containers can offer efficient management of resources through both vertical and horizontal scaling capabilities. This flexibility however, do not offset the gains one can achieve through focussed and continuous approach towards tests suit optimisations. The former can provide an initial breakthrough only to find the system outgrow its resourcing capacity as inefficient project builds and repetitive tests grow exponentially overtime. We must try to ensure tech stack efficiencies are coupled with those gained through adopting best practices and test suit optimisation within the development team.

Even with standardised solutions, high level of flexibility and customisation is needed when used across teams to capture edge use cases that often vary across individual units. Consequently effectiveness of such solutions equally depend on efficiency of an organisation and its people to adopt standardized practices for customizations and minimise abuse leading to biased metrics, unintended behaviours and regular discussion at multiple levels to utilise its complete potential in improving test analysis across the wider set of development teams.

Acknowledgment

This paper takes inspiration from Adaptive Lean Software Testing (PA2579), a course taught by David Fucci together with teaching assistants Nayla Nasir and Laiq Muhammad, Department of Software Engineering at Blekinge Institute of Technology, Sweden.

References

[1] Highsmith, J.A.: ‘Agile software development ecosystems’ (Addison-Wesley Professional, 2002. 2002)

[2] Hiranabe, K.: ‘Kanban applied to software development: From agile to lean’, InfoQ. URL: http://www. infoq. com/articles/hiranabe-lean-agile-kanban [accessed 7 March 2013], 2008

[3] Wang, X., Conboy, K., and Cawley, O.: ‘“Leagile” software development: An experience report analysis of the application of lean approaches in agile software development’, Journal of Systems and Software, 2012, 85, (6), pp. 1287-1299

[4] Staats, B.R., Brunner, D.J., and Upton, D.M.: ‘Lean principles, learning, and knowledge work: Evidence from a software services provider’, Journal of Operations Management, 2010, 29, (5), pp. 376-390

[5] Kristensen, T.B., Saabye, H., and Edmondson, A.: ‘Becoming a learning organization while enhancing performance: the case of LEGO’, International Journal of Operations & Production Management, 2022, 42, (13), pp. 438-481

[6] Fucci, D., Erdogmus, H., Turhan, B., Oivo, M., and Juristo, N.: ‘A dissection of the test-driven development process: Does it really matter to test-first or to test-last?’, IEEE Transactions on Software Engineering, 2016, 43, (7), pp. 597-614

[7] Rutherford, M.J., Carzaniga, A., and Wolf, A.L.: ‘Simulation-based testing of distributed systems’, Colorado University at Boulder Department of Computer Science, 2006

[8] Ma, Y.S., Offutt, J., and Kwon, Y.R.: ‘MuJava: an automated class mutation system’, Software Testing, Verification and Reliability, 2005, 15, (2), pp. 97-133

[9] Shukla, R., Strooper, P., and Carrington, D.: ‘A framework for statistical testing of software components’, International Journal of Software Engineering and Knowledge Engineering, 2007, 17, (03), pp. 379-405

[10] Dudekula Mohammad, R., Katam Reddy Kiran, M., Petersen, K., and Mantyla, M.V.: ‘Benefits and limitations of automated software testing: Systematic literature review and practitioner survey’. Proc. 2012 7th International Workshop on Automation of Software Test (AST)2012 pp. Pages

[11] Böhme, M., and Paul, S.: ‘On the efficiency of automated testing’. Proc. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering2014 pp. Pages

[12] Chen, T.Y., Kuo, F.-C., Liu, H., and Wong, W.E.: ‘Code Coverage of Adaptive Random Testing’, IEEE Transactions on Reliability, 2013, 62, (1), pp. 226-237

[13] Kaczanowski, T.: ‘Practical Unit Testing with TestNG and Mockito’ (Tomasz Kaczanowski, 2012. 2012)

[14] Stouky, A., Jaoujane, B., Daoudi, R., and Chaoui, H.: ‘Improving Software Automation Testing Using Jenkins, and Machine Learning Under Big Data’: ‘Big Data Technologies and Applications’ (2018), pp. 87-96

[15] Puri-Jobi, S.: ‘Test automation for NFC ICs using Jenkins and NUnit’. Proc. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)2015 pp. Pages

[16] Parsai, A., and Demeyer, S.: ‘Comparing mutation coverage against branch coverage in an industrial setting’, International Journal on Software Tools for Technology Transfer, 2020, 22, (4), pp. 365-388

[17] Poth, A., Werner, M., and Lei, X.: ‘How to Deliver Faster with CI/CD Integrated Testing Services?’: ‘Systems, Software and Services Process Improvement’ (2018), pp. 401-409

[18] Nuntapramote, T.: ‘Adapting Regression Test Optimization for Continuous Delivery’, PhD thesis. 2018 (cited on pages 1, 6, 60), 2018

[19] Chuchuen, Y., and Rattanaopas, K.: ‘Implementation of Container Based Parallel System for Automation Software Testing’. Proc. 2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering2021 pp. Pages

[20] Milojkovic, J., Ciric, V., and Rancic, D.: ‘Design and Implementation of Cluster Based Parallel System for Software Testing’. Proc. 2020 Zooming Innovation in Consumer Technologies Conference (ZINC)2020 pp. Pages

[21] Bitnami: ‘Popular applications, provided by Bitnami, containerized and ready to launch.’, in Editor (Ed.)^(Eds.): ‘Book Popular applications, provided by Bitnami, containerized and ready to launch.’ (GitHub, 2024, edn.), pp.

[22] Jenkins: ‘Discover the 1900+ community contributed Jenkins plugins to support building, deploying and automating any project.’, in Editor (Ed.)^(Eds.): ‘Book Discover the 1900+ community contributed Jenkins plugins to support building, deploying and automating any project.’ (2024, edn.), pp.

[23] Cai, X., and Lyu, M.R.: ‘The effect of code coverage on fault detection under different testing profiles’. Proc. Proceedings of the first international workshop on Advances in model-based testing - A-MOST '052005 pp. Pages

[24] Sánchez, A.B., Parejo, J.A., Segura, S., Durán, A., and Papadakis, M.: ‘Mutation Testing in Practice: Insights From Open-Source Software Developers’, IEEE Transactions on Software Engineering, 2024, 50, (5), pp. 1130-1143

[25] Suri, B., and Mangal, I.: ‘Analyzing test case selection using proposed hybrid technique based on BCO and genetic algorithm and a comparison with ACO’, International Journal of Advanced Research in Computer Science and Software Engineering, 2012, 2, (4)

[26] Khari, M., Kumar, P., Burgos, D., and Crespo, R.G.: ‘Optimized test suites for automated testing using different optimization techniques’, Soft Computing, 2017, 22, (24), pp. 8341-8352

[27] Agrawal, A.P., Choudhary, A., and Kaur, A.: ‘An Effective Regression Test Case Selection Using Hybrid Whale Optimization Algorithm’, International Journal of Distributed Systems and Technologies, 2020, 11, (1), pp. 53-67

[28] Poppendieck, M., and Cusumano, M.A.: ‘Lean Software Development: A Tutorial’, IEEE Software, 2012, 29, (5), pp. 26-32

[29] Nidagundi, P., and Novickis, L.: ‘Introduction to Lean Canvas Transformation Models and Metrics in Software Testing’, Applied Computer Systems, 2016, 19, (1), pp. 30-36

[30] Alwardt, A.L., Mikeska, N., Pandorf, R.J., and Tarpley, P.R.: ‘A lean approach to designing for software testability’. Proc. 2009 Ieee Autotestcon2009 pp. Pages

[31] Fucci, D., and Turhan, B.: ‘A replicated experiment on the effectiveness of test-first development’, in Editor (Ed.)^(Eds.): ‘Book A replicated experiment on the effectiveness of test-first development’ (IEEE, 2013, edn.), pp. 103-112

[32] Wolf, J., and Yoon, S.: ‘Automated testing for continuous delivery pipelines’, in Editor (Ed.)^(Eds.): ‘Book Automated testing for continuous delivery pipelines’ (2016, edn.), pp.

[33] Polo, M., Tendero, S., and Piattini, M.: ‘Integrating techniques and tools for testing automation’, Software Testing, Verification and Reliability, 2007, 17, (1), pp. 3-39

[34] Sneha, K., and Malle, G.M.: ‘Research on software testing techniques and software automation testing tools’. Proc. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)2017 pp. Pages

[35] Fleck, P.: ‘Utilizing Code Coverage Density to Enhance Software Quality Management Decisions’, Wien, 2022

[36] https://pitest.org/

[37] https://openclover.org/