MANTECH Publications
Recent Trends in Computer Science and Software Technology
Volume 7, Issue 2, May-August 2022

Integrating Performance Testing into the Agile Software Model

Dr. Jayachandran R[1], Sabu Joseph[2]
Professor[1], Student[2]
Department of CSE
KMP College of Engineering Cherukunnam
Corresponding Author's E-mail: *sabujoseph2@gmail.com[2]*

Abstract

The software development process has developed dramatically over the last few decades, from Big Bang through Waterfall to Agile. The "Speed of Delivery" of the Software Product, which has considerably sped from months to weeks and days, was the key motivation for the advancement of the software. IT (Information Technology) organisations must have a strong technological presence in order to push out new software and services to their consumer base as rapidly as feasible. The present user generation makes the best use of technology and is constantly seeking to stay up with new trends. The main topic is for organisations to be prepared with their Speed of Delivery strategy, adapting to all technology modernization initiatives such as CICD (Continuous Integration and Continuous Deployment), Agile, DevOps, and Cloud, so that there is minimal customer friction and no risk to their market share. The purpose of this article is to compare performance testing throughout the agile paradigm to traditional end testing. This document presents the findings of the appropriate testing phases.

Keywords: *Performance Testing, CICD, Agile, Waterfall, Embedding Performance*

INTRODUCTION

Customers frequently encounter digital disturbances when using software products or websites during peak holiday seasons and continuous high demand promotions, affecting the end user experience. This problem is caused by application slowness or stability problems. That are attributed to

the amount of transactions that the application Logic is not built to manage, or to the fact that the capacity of the calculation required was not foreseen ahead of time.

Performance issues often have a large stake in all tangents of a company, beginning with the customer's faith in utilising the product again, total cost of operations and financial repercussions, and the firm's overall brand reputation.

The waterfall model for software development is well-known and has been regularly utilised for over a decade in the software development process. The waterfall model includes software development phases that are completed in a sequential linear order, including requirement phases, design, implementation, verification, and maintenance. The requirements are recorded, and the following phase is contingent on the previous phase's approval.

It normally takes some time for end customers to be able to utilise the programme or partial functionalities. There is sometimes a possibility that the specifications will be deviated and a significant amount of rework will be required to suit the end user's expectations.

Agile software development is an iterative delivery strategy in which software products are divided into smaller parts and delivered progressively in sprints [1].

Each sprint includes all phases, from design through deployment, and is typically completed in 2 to 4 weeks. In Agile, the application is continually distributed to end users, and feedback is integrated in subsequent sprints with more product development; this entire Sprint cycle is continued until the intended software product development is accomplished.

Agile is becoming more popular because customers are more confidence in the conclusion since they have a constant feel for the product and can provide input to software development teams on a continual basis. One of the problems of the Agile Software development methodology is dealing with quality control processes such as performance testing in a short period of time. This study will address the issues and potential solutions that may be used to achieve performance readiness and high stability of software applications

under an agile software development approach.

The key performance readiness requirement in agile is to guarantee that there are no interruptions to the new features that will be delivered in the sprint as well as the current product features that customers are already utilising in production.

Related Work

Andre et.al [1] talks about developing a common agile software development model. Also, Jun Lin et.al [2] talk about modelling user stories in agile software development model. Marian et.al [3] uses the agile vs traditional comparison to show the importance of the former. [6] talks about the performance testing for developers which relates to the performance testing in agile mentioned in this paper.

Finding Performance Issues late in the release

The Branching and Merging Strategy is critical in the agile approach since it drives development velocity and overall Release Cadence. Figure 1 depicts a typical list of actions carried out in an agile development branch. Performance testing is typically pushed to the end of the sprint, giving less time to conduct good tests and handle any Performance issues encountered during that sprint. This practise makes software applications exposed to new difficulties and instability, resulting in dissatisfied end users and a negative influence on the organization's total income.

Lack of Automation in Performance Testing

Every step of the performance test execution process, including the deployment of the application code base into the performance environment, the preparation of test data, the use of smoke testing, the execution of the necessary types of performance tests, the analysis of the results, and the tuning of performance bottlenecks, requires a significant amount of human interaction.

All of these time-consuming tasks sometimes struggle to fit into a brief sprint release cycle, which compromises Performance concerns in production.

Executing Agile Performance testing with a specialised Centralized Performance Team or any other testing procedure in a waterfall software architecture [3] used to be the last gate before the product was released, and it was simple for the Centralized Performance Team to do so.

The conventional approach to performance testing, which included a specialised centre of excellence, doesn't work well with the agile software development model's rapid, frequent feedback cycle.

POTENTIAL APPROACHES TO MITIGATE THE PROBLEMS

Shift Left Performance Testing

Now that we have a better understanding of the issue, let's take a closer look at the Branching and Merging Approach, which is crucial to the agile strategy and determines the development pace and overall release cadence. As can be seen in the example of a typical Agile Delivery below, some feature codes are continuously integrated into the application Master code base during the development cycle.

Positioning the Performance testing in Agile thus is very complicated considering

- The Application and Scope is continuously changing and
- There is a very short runway to complete the Performance testing on time

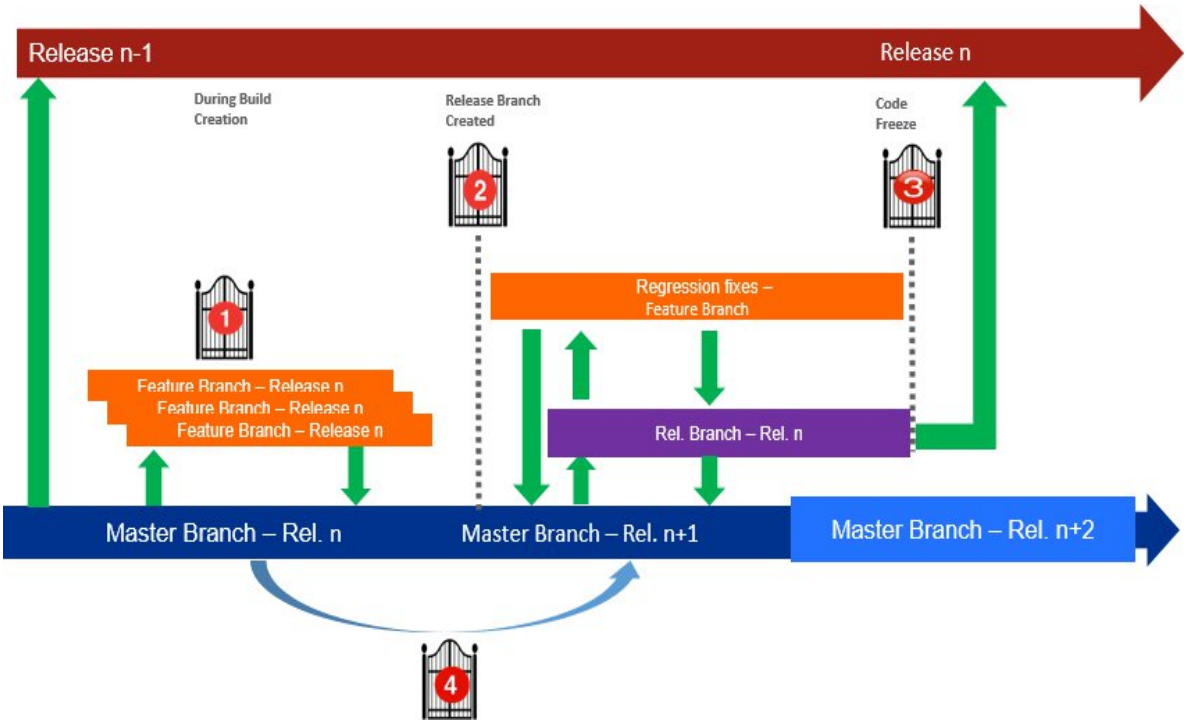

Figure 1 Agile Delivery Process

In an agile sprint, there are four alternative gateways that enable performance testing. In order to uncover performance issues sooner and prevent them from being pushed to the backlog and frequently leaking into production, which has high business impacts due to stability concerns and end user experience issues, the goal is to push the performance testing practise as much as possible during the development sprint.

Figure 2 illustrates how performance testing must be shifted as far to the left as possible. This "Shift Left technique" involves performing some testing at the feature branch level in the developer environment with the very minimum predicted baseline volume. This will reveal any significant open blockages that may emerge later in the sprint and provide insight into the high-level performance of the produced code. Before freshly produced code or a feature branch is merged into the Master Code base, there should be a solid self-service capability that enables developers to do efficient performance testing at the feature branch level. These self-service testing tools are often supported by the Performance or Tools teams. To get a first impression of the performance, this testing can be narrowed down to an API level, service level, or a specific functionality that corresponds to a recently developed feature. Later, the performance team will execute the rigorous performance testing cycle in the release branch with the full volume in the production-equivalent performance test environment.

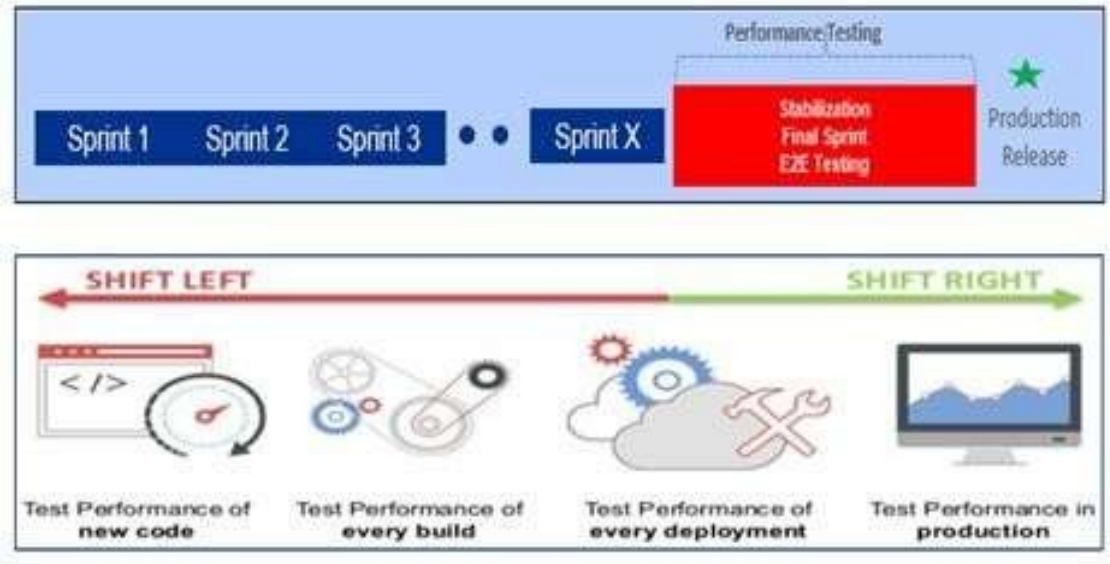

Figure 2: Shift Left and Right Strategy

Automation of Performance Testing Integration in the CICD Pipeline

Performance testing is difficult to complete on time with the speed at which Agile delivery occurs, and performance engineers are unable to keep up with the hundreds of items that are being pushed into the pipeline at any given moment. The main idea behind a CICD pipeline is to provide developers the ability to perform performance testing at two different stages: feature branch testing and continuous deployment, which is more appropriate later on in the cycle at the release branch level and has greater automation.

The performance tool's provided plugins may be used to automate the Continuous Deployment workflow. For an automation to be effective, it's critical that the performance testing tool be integrated into the pipeline. The performance test job that initiates the performance test suite is a point-based SLA driven test suite that automatically sends signals to the pipeline about Success or Failure of the Performance and enables the continuity of the pipeline. Performance test tools plugin must be installed and configured in the Pipeline Automation solution in order to call the test suite after the deployment is complete in the performance environment.

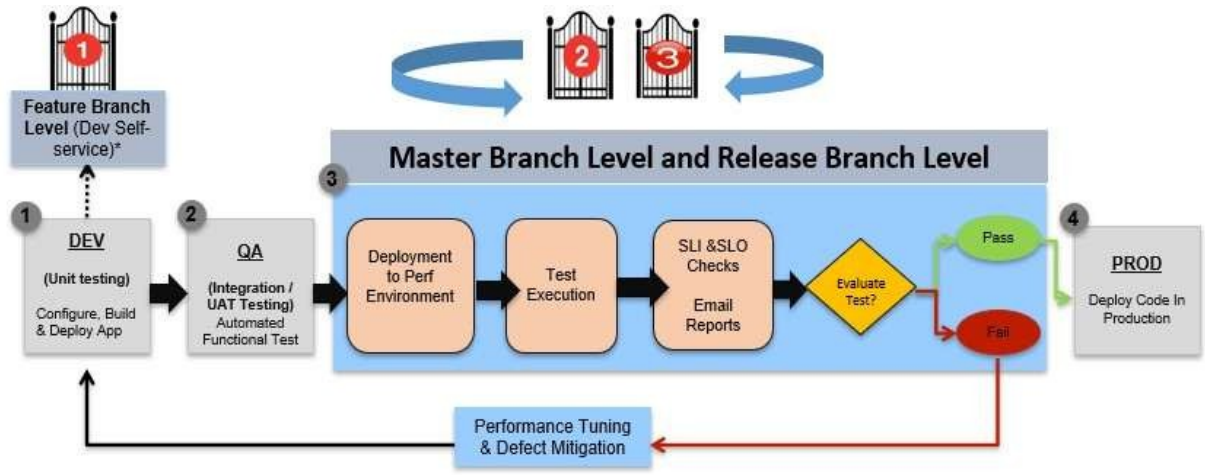

Figure 3: Deployment Phases

Test Environment and Data Dependencies

The volume of test data, which is crucial for the accuracy of the Performance testing findings, must be managed well for an efficient test cycle to occur, whether it is human or automated. The critical test data best practises that must be addressed for a successful automation of performance testing are listed below.

- Volume of the test data in the Performance environment should match the production volume - A better mechanism is to have a regular refresh of production database and have the process of export and import into perf environment automated.
- There are huge number of scenarios that are data dependent and cannot be reused iteratively with in scripts .These scenarios has to be identified and should have a good plan in place to retrieve test data real time before the test is executed in pipeline and should be able to feed in to scripts to have the test suite exercised without any disruptions

By standardising Test tool, Test practise, and establishing the environment requirements, it's crucial to decide on a consistent approach for the Performance Environment throughout the whole business. The local development environment or an on-demand virtual environment must be used for the feature branch testing that is mostly driven by the individual developers. The artefacts are later delivered into the Performance environment after the application is prepared and ready for deployment following the cutting of the release branch. The performance environment should have every end-to-end component and logical flow linked, and it should be 100% comparable to production capacity. To investigate any performance aberrations found in the performance environment, performance monitoring has to be well-instrumented.

Monitoring and Engineering Analysis

All monitoring functions should be available in the Automation Pipeline, but this is only achievable if monitoring and Application Performance Monitoring (APM) tools are connected with Performance tools. The testing tool should be set up with the appropriate Service level objectives and indicators, and it should be able to transmit the status of the performance testing back to the pipeline.

Automated report that can be shipped as an email will expedite the Performance analysis process that highlights any

performance Service level agreement (SLA) deviations and this helps performance engineers to easily pinpoint and start the deep dive analysis quickly and engage with the partner team and start the triage at the earliest.

Performance test Engagement model

The application engineering teams received requests for performance testing from throughout the business, and the performance team prioritised its tasks and reported the outcomes of its execution as well as any performance concerns that needed to be addressed. Performance testing should be regarded as a team sport with active collaboration between the Performance team and the Application development team because this method might become a bottleneck for the overall agile delivery deadlines.

It is typically a good idea to identify Performance SME's for each of the scrum teams in order to address this challenge. These individuals should take part in the daily stand-ups and sprint planning in order to understand any Performance impacts with the new stories that are planned for the upcoming release. As a result, when the release branch is cut, the Performance team is able to plan ahead and maintain the test suite. Additionally, by providing the developers with self-service performance test choices, the identified performance engineers allow performance testing for each Scrum team.

Figure 4: A template of the automated report

In general, One to One Performance engineers mapped for each developer is not a cost efficient model and transition of Performance team from an enterprise center of Excellence to center of Enablement team can make the Performance testing in Agile scrum team more productive as depicted in Figure 5. There will be a representative identified for each scrum team that will be driving all the end to end perf needs including enabling shift left and preparation of Automating Performance test execution in the pipeline. Now that some of the general underlying problems are discussed about agile software development model with incorporating Performance testing is discussed and potential solution has been laid out. There are many pros and cons that comes up with the discussed solutions, however the defined approach must be tailored based on the development ecosystem and the system architecture considering the details that are discussed below.

MERITS AND RISKS OF THE PROPOSED APPROACH

Time to Market

Having a shift left Performance testing enables Performance issues to be uncovered at the featured branch level and resolve earlier in the life cycle, this save lot of time and operational expenses to the overall software development program. With the automated Performance testing in the Continuous deployment pipeline enables the Performance findings to be shared quickly and wrap up the Performance readiness signoff process sooner in the release sprints. Overall, these two practices enable the speed of delivery in the agile software development model thus increasing the time to market that the business can take benefit. Another important aspect where the software can be delivered to the end users more.

End to End and Integration Gaps

Often Shift Left Performance testing are executed in a low scale environment with low transaction volume and virtualized integration end points of transactions though this uncovers high level performance issues and gives opportunity to developers to resolve performance issues earlier before the code is committed to master. The Performance delays and issues that are tied to the integration points or asynchronous services are not in scope of this testing. One of the practices to mitigate to some extend is have a realistic delays with the virtualized endpoints and to know about the latency patterns it needs some level of Production monitoring trend

analysis about the latency of these end points.

As discussed on the different merits and drawbacks about bringing in continuous performance testing in the DevOPS practices, we also wanted to try out if this concept can be proven in a lab setup where on demand environments can be spun and Performance testing process be automated in the pipeline.

RESULTS

Following were the objectives that were defined for the proof of concept.

- Terraform scripts that will be used to spin up a instance will be retrieved from GIT HUB
- Jenkins Terraform job will spin a mock App instance on Amazon cloud (AWS) on demand.
- Same pipeline job will initiate another freestyle Jenkins job, which will run a test (test suite) and hit the sample app instance created by the first JOB.
- As soon as the test job is complete, there will be a performance test report that will be shipped out as an email
- Created instance will be destroyed by the Jenkins job after test suite is complete.

Figure 5 Tests being performed

Figure 6: Performance testing using CAVISSON

Below is the summary of the data that was captured about the time duration usually they spent to complete a full end to end performance testing cycle. This data clearly depicts that it easily takes about 2 to 4 days to get the performance findings out and these are matured technology organizations with experts driving the performance testing with decades of experience.

Domain	Insurance		Retail		Banking	
Application -Tech stacks	**Policy Apps** Java Oracle Tibco EIS Tomcat	**Claims Apps** Java /Oracle/GW/Some AWS App components and .net apps	**Supply chain -Order Management** Java Weblogic Mainframe DB2 F5	**Store Operations Datapower** Json Rest services Oracle Java	**Internal Contact Center App** .Net Thick Client Apps , SQL Server	**Consumer Banking -** React /Openshift/AWS API Gateway Java OCP ,Oracle , MDM -IBM ,
Release Deployment *Coordination and Ticket creation	4 to 5 Hrs.	2 to 4 Hrs.	1 day	1 day	2 to 4 Hrs.	2 to 3 Hrs.
Smoke Testing and issue resolution *Environment , functional and Data issues	2 days	1 Hr.	4 Hrs.	3 Hrs.	3 Hrs.	1 day
Performance Testing *including Dat prep	2.5 Hrs.	3 Hrs.	2 Hrs.	8 Hrs.	2 Hrs.	2 Hrs.
Result Preparation	2 Hrs.	2 Hrs.	3 Hrs.	3 Hrs.	1 Hr.	2 Hrs.
Analysis	4 Hrs.	2 Hrs.	1 day	3 Hrs.	1 Hr.	2 Hrs.
Number of issues	6 issues	4 issues	1 to 2 issue	2 to 4 issue	1 Issue	2 to 4 issues
% of new issues are related to new development	~ 50%	~25%	~50%	~50%	~50%	~50%
Feature branch testing	No	No	No	No	No	No
Total Time for 1 Performance Testing Cycle	***~4 Days***	***~2 Days***	***~2 Days***	***~3 days***	***` 2 Days***	***` 3 Days***

Figure 7: Performance testing cycle results

CONCLUSION AND FUTURE WORK

The performance testing process must be automated in the pipeline with automated reporting that can offer real-time feedback in order to meet the upcoming need for speed of delivery in the software development lifecycle. Shift Left Automation, in my opinion, will be the centre of performance testing automation in the future. The industry as a whole is approaching this issue in a variety of ways, but the ultimate success of this research will depend on how much automation is added to the left to enable the performance

testing process as early as possible in the software development lifecycle. Lack of a suitable testbed and a software development process are two further drawbacks. Some findings may differ while testing this functionality in a tiny context as opposed to a real software development environment. Additionally, testing takes up a lot of time throughout each phase, which might cause a production delay for larger software. This research may be used to evaluate modest initiatives, and if it is effective, it can be expanded to much larger ones. This may

even be applied to the creation of apps, ranging in size from the smallest to the largest and most popular ones, as the one made on Alexa by author 2 who was described above.

REFERENCES

1. https://softcrylic.com/blogs/performance-testing-for-devops/
2. Srdjana Dragicevic, Stipe Celar, MiliTuric (2017), "Bayesian network model for task effort estimation in agile software development", Journal of Systems and Software Volume 127, May 2017, Pages 109-119
3. https://ieeexplore.ieee.org/abstract/document/4293621
4. Jun Lin, Han Yu, Zhiqi Shen, Chunyan Miao (2014), "Using goal net to model user stories in agile software development", ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD).
5. https://www.synopsys.com/blogs/software-security/continuous-testing-cicd/
6. Marian Stoica, Marinela Mircea, Bogdan Ghilic-micu (2013), "Software Development: Agile vs. Traditional", Informatica Economică vol. 17, no. 4/2013.
7. [André Janus (2018), "Towards a common Agile Software Development Model", ACM SIGSOFT, July 2012 Volume 37 Number 4.