# EMBEDDING PERFORMANCE TESTING IN AGILE SOFTWARE MODEL

Suresh Kannan Duraisamy, Bryce Bass and Sai Mukkavilli

Department of Computer Science,
Georgia Southwestern State University, Americus, GA

## *ABSTRACT*

*In the last couple of decades, the software development process has evolved drastically, starting from Big Bang to Waterfall to Agile. The primary driver for the evolution of the software was the "Speed of Delivery" of the Software Product which has significantly accelerated from months to less than weeks and days. For IT (Information Technology) Organizations to be successful, they inevitably need a strong technology presence to roll out new software and features as quickly as possible to their customer base. The current user generation tends to use technology to maximum potential and is always striving to keep up with the new trends. The main subject is for the organizations to be ready with their Speed of Delivery strategy adapting to all technology modernization initiatives like CICD (Continuous Integration and Continuous Deployment), Agile, DevOps, and Cloud so that there are negligible customer friction and no risks to their Market shares,. The aim of this paper is to compare the performance testing in every stage of the agile model to the traditional end testing. The results of the corresponding testing phases are presented in this paper.*

## *KEYWORDS*

*Agile, CICD, Waterfall, Performance Testing.*

## 1. INTRODUCTION

### 1.1. Problem Statement

Many a times while using a software product or websites during peak holiday seasons and during ongoing high demand promotions, customers come across digital disruptions while using the software products which impacts the end user experience. This issues are caused due to slowness or stability concerns of the applications. That are pointed towards the volume of transactions that the application Logic is not designed to handle or some time the capacity of the computation that was required was not planned ahead of time.

Performance Issues usually have a high stake of all the tangent of an organization starting from the Customer trust on using the product again, Total cost of Operations and Financial impacts and the Overall brand reputation of the organization.

Waterfall model for software development is a well-known and was very widely used for more than a decade for a software development process. The waterfall model has software development phases accomplished in a sequential linear way which comprises of Requirement Phases, Design, Implementation, Verification and Maintenance. Requirements are documented and moving to next phase is dependent on the signoff of the prior phase. It usually takes time for

the end users before they can start using the software or partial features. Often time there are risks that the requirements are deviated and considerable amount of rework has to be accomplished to meet the end user's needs.

Agile software development model is an iterative deliverable approach where software Product are broken down into smaller features and are delivered incrementally in sprints [1]. Each sprint has all the Phases incorporated starting from Design to Deployment and generally are delivered within 2 to 4 weeks. In Agile the application is released to end users in continuously and the feedbacks are incorporated in the upcoming sprints with the additional development of the Product, this entire cycle of Sprint is repeated until the desired software product development is completed.

Agile is more popular these days as the customer are more confident on the outcome as they have a continuous feel of the product and can share feedbacks continuously to the software development teams. One of the challenges of Agile Software development model is about Addressing of Quality control Processes including Performance testing in the short sprint of time. This work will be covering the problems and potential solutions that can be adapted to deliver Performance readiness and ensure high stability of software applications in agile software development model.

The primary Performance readiness requirement in agile is to ensure there is no disruptions to the new features that are to be delivered in the sprint and the existing product features that the customers are already using in the production.

## 1.2. 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.

## 1.3. Finding Performance Issues late in the release

The Branching and Merging Strategy plays a vital role in the agile model that drives the development velocity and the overall Release Cadence. Figure 1 has the List of typical activities carried out in an agile development branch. Performance testing usually is pushed as much as to the end of the sprint leaving less time to execute good test and resolve any Performance issues that are introduced newly into that sprint. This Practice creates software application vulnerable to new issues and instability thus causing unhappy end users that impacts the overall revenue of the organization.

## 1.4. Lack of Automation in Performance Testing

Performance Testing [8] is manual driven and needs lot of human intervention in every step of Performance test execution that includes Deployment of application code base in the Perf environment, Smoke testing , test data preparations, Monitoring Instrumentation and running the required type of Performance tests, and Result analysis and Tuning the Performance bottlenecks. All these activities are time consuming and often struggle to fit within a short sprint release cadence and causes Performance issues to be compromised in production.

### 1.5. Executing Agile Performance testing with dedicated Centralized Performance Team

Performance Testing or any Testing practice in waterfall software model [3] used to be the final gate before the release of the product which was easy for Centralized Performance Team to execute. The way Performance testing was traditionally done with a dedicated Performance testing also referred to as Center of Excellence doesn't fits well in the rapid fast paced frequent feedback based agile software development model.

## 2. POTENTIAL APPROACHES TO MITIGATE THE PROBLEMS

### 2.1. Shift Left Performance Testing

Let's understand the problem deeper and for that we would need to deep dive in to The Branching and Merging Strategy that plays a vital role in the agile strategy and drives the development velocity and the overall Release Cadence. As seen in the below typical Agile Delivery, throughout the development cycle there is a continuous integration of some feature codes in to the application Master code base.

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

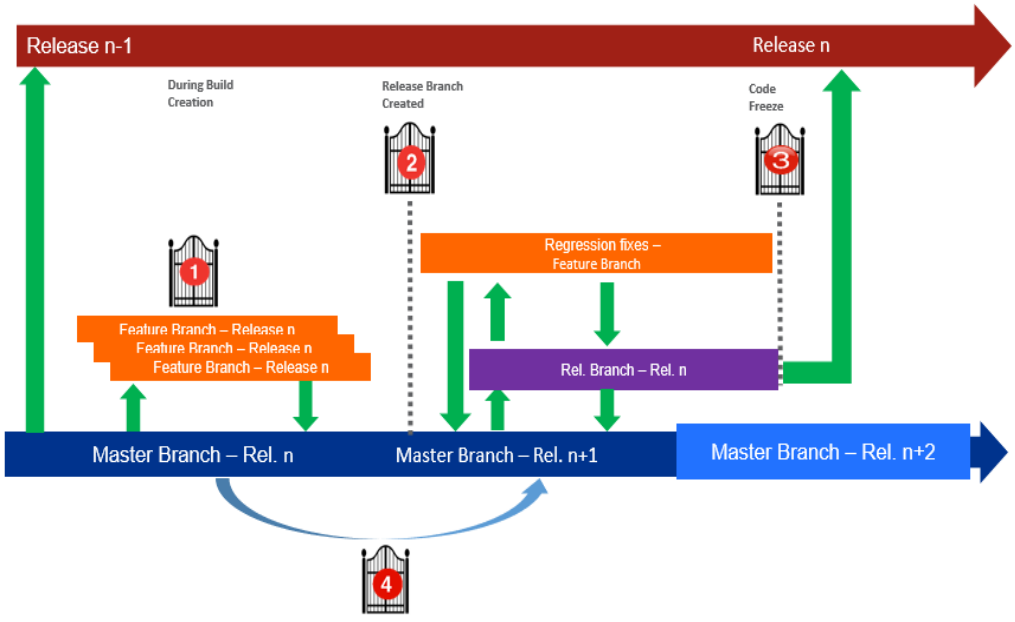


Figure 1. Agile Delivery Process

There are four possible gateways to enable the Performance testing in an agile sprint. The goal is to push the Performance testing practice as much as left in the development sprint so that Performance issues can be uncovered sooner to avoid pushing performance defects to backlog and often Performance issues gets leaked in to production which has high business impacts – due to stability concerns and end user experience dissatisfaction

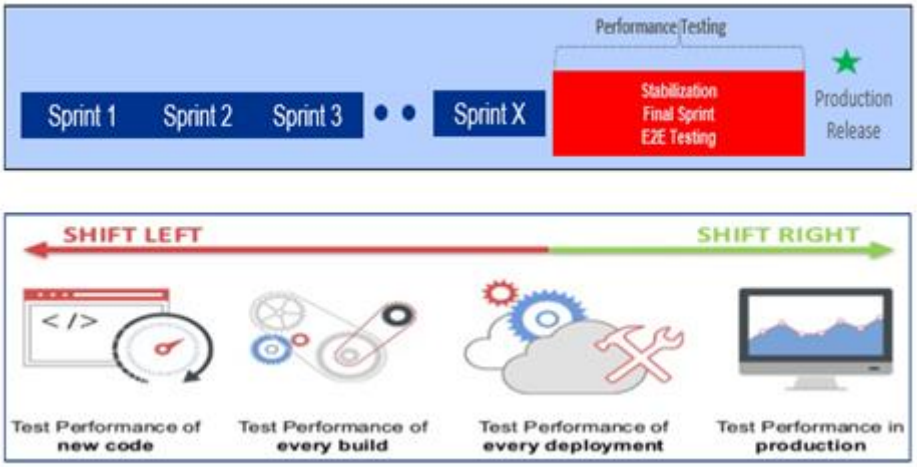


Figure 2. Shift Left and Right Strategy

As displayed in Figure 2, the performance testing has to be moved as much as left –"Shift Left strategy" that is having some level of testing at the feature branch level in the developer environment with minimum expected baseline volume. This will give an insight of the high-level performance of the code that is developed and will uncover any major open blockers that can show up later in the sprint. To enable Shift Left Performance strategy there should be a good self service capability where the developers can do effective performance testing at the feature branch level easily before the newly developed code or feature branch is merged into the Master Code base. Usually this kind of self-service testing tool are facilitated by the Performance teams or the Tools teams. This testing can be focused to an API Level or service level or a specific functionality corresponding to the newly developed feature to get the first insight of the performance and Later the Performance team will exercise the rigor performance testing cycle in the release branch with the full volume in the Production equivalent Performance test environment.

## 2.2. Automation of Performance Testing

### 2.2.1. Integration in the CICD Pipeline

With the Speed, Agile delivery runs it's not only hard to complete performance testing on time but impossible for Performance engineers to keep up with the 100s of features that is pushed continuously into pipeline. The Primary notion to handle the Performance testing in a CICD Pipeline is that enable the Developers to do the Performance testing at the CI part of the pipeline i.e. at the feature branch testing , And at the Continuous deployment stage which is more applicable later in the cycle at the release branch level have as much as automation.

Automation in the Continuous deployment pipeline can be implemented using the plugins the performance tool provides. It's important for Performance Testing tool to integrate in the Automation pipeline to have a successful automation in place. Performance test tools plugin has to be installed and configured in the Pipeline Automation solution there by calling the test suite after the deployment is completed in to the Performance environment's Performance test job that triggers the performance test suite is a point based SLA driven test suite that automatically sends signal to the pipeline about Success or failure of the Performance and enables the continuity of the pipeline.

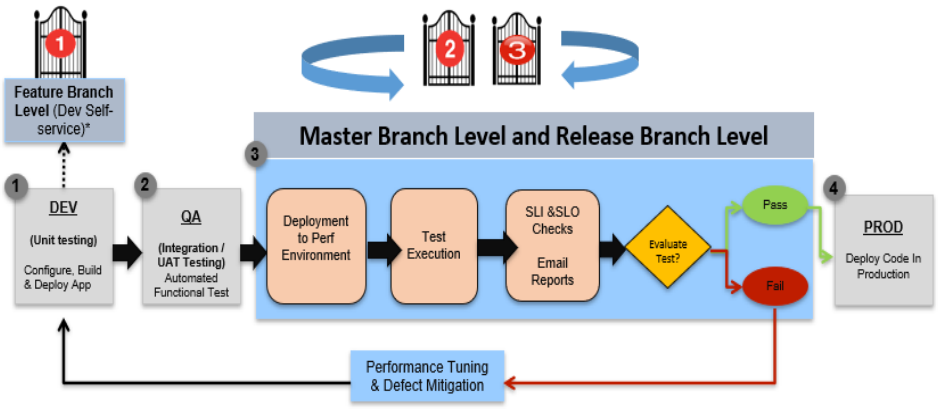


Figure. 3. Deployment Phases

### 2.2.2. Test Environment and Data Dependencies

For an effective test cycle wither manual or automation it's important to have a good test data management including the volume of the test data that is very important for the accuracy of the Performance testing results. Below are some of the important test data best practices that needs to be covered to have a successful automation of performance testing.

- 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

On the Performance Environment, it's important to settle on a common methodology across organization by standardizing Test tool, Test practice and solidifying on the environment needs. The feature branch testing that is primarily driven by the individual developers must leverage the local dev environment or can be facilitated by an on demand Virtual environment.

Later when the application is ready and deployable after the release branch is cut, the artefacts are deployed into the Performance environment. The Performance environment should be 100% equivalent to production capacity with as much as end to end component and logical flow connected. Performance monitoring should be well instrumented in the Performance environment to drill down any performance deviations noticed in the Performance environment.

### 2.2.3. Monitoring and Engineering Analysis

The Automation Pipeline should have all the monitoring capabilities, and this is possible only if the monitoring and the Application Performance monitoring tools (APM) is integrated with the Performance tools. The testing tool should be configured with the right defined Service level indicators and Service level objectives and can communicate back to the pipeline with the status of the performance testing.

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.

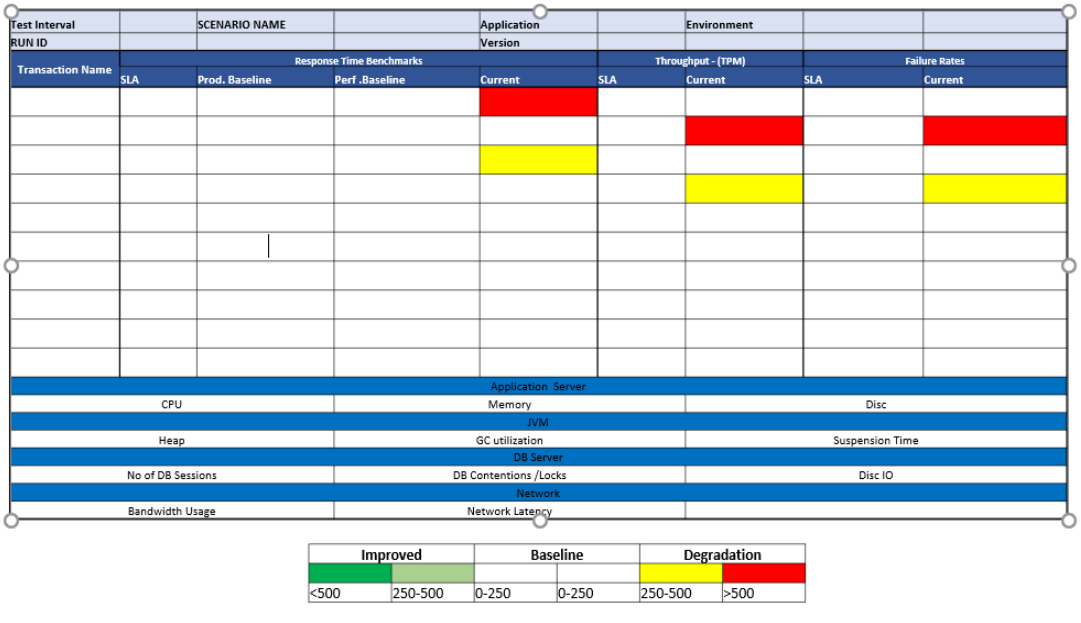


Figure 4. A template of the automated report.

## 2.3. Performance test Engagement model

Traditionally Performance testing was a centralized team that gets request for Performance testing across organization and the Performance team prioritizes there task and report out the execution results and any Performance issues that needs to be tuned back to the application engineering teams. This approach may turn out a bottleneck for the entire agile delivery timelines and Performance testing should be treated as a team sport with active collaboration between Performance team and Application development team.

Usually to address this challenge it's good to have Performance SME's identified for each of the scrum teams and they participate in the daily stand-ups and sprint planning to understand any Performance impacts with the new stories that are planned for the upcoming release. This enables the Performance team to plan of time and keep up the test suite when the release branch is cut. Also, this identified performance engineers enables performance testing for each of the Scrum team by facilitating the self-service performance test options to the developers.
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.

## 3. MERITS AND RISKS OF THE PROPOSED APPROACH

### 3.1. 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.

### 3.2. 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.

## 4. RESULTS

Following were the objectives that was 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.

| | Declarative: Tool Install | Git Checkout | Terraform Init | Terraform Apply | Execute Performance Test Job | HTML Performance Report | Terraform Destroy |
|---|---|---|---|---|---|---|---|
| Average stage times: (Average full run time: ~8min 42s) | 438ms | 2s | 2s | 2min 14s | 1min 21s | 521ms | 1min 9s |
| #179 Jul 01 14:05 No Changes | 283ms | 1s | 2s | 3min 0s | 4min 22s | 951ms | 1min 22s |
| #178 Jul 01 13:54 No Changes | 529ms | 1s | 2s | 3min 0s | 2min 19s | 1s | 1min 32s |
| #177 Jul 01 13:44 No Changes | 389ms | 2s | 2s | 3min 0s | 2min 25s | 1s | 4min 38s |

Figure 5. Tests being performed

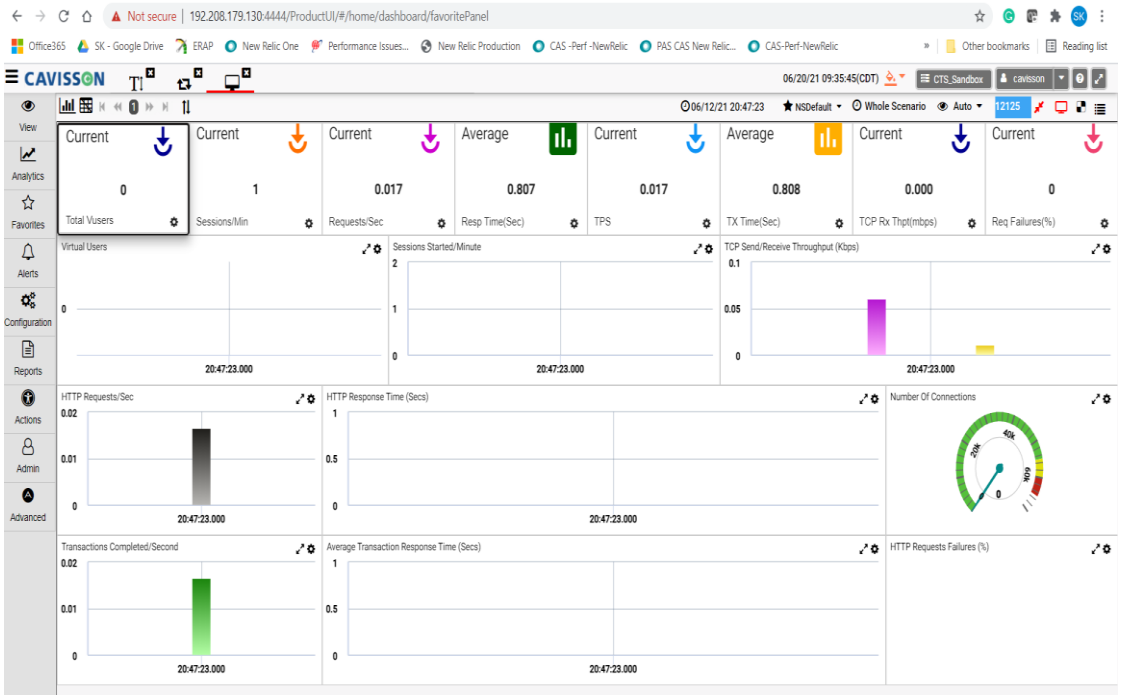


Figure 6. Performance testing using CAVISSON

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 /Open shift/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

With the proposed Automation solution proposed, this testing result can be shipped out to the stakeholders within an hour of deployment into performance environment. This includes spin up of an on-demand environment using Terraform scripts in the AWS EC2 Instance followed by running the Performance test suite and then shipping out the test report and finally tearing down the OnDemand server

## ALEXA APP JSON SCRIPT:

```
{
  "manifest": {
    "publishingInformation": {
      "locales": {
        "en-US": {
          "summary": "Quiz for GSW",
          "examplePhrases": [
            "Alexa, open georgia southwestern quiz"
          ],
          "keywords": [],
          "name": "GSW Quiz",
          "description": "This is a simple Quiz that tests your knowledge on Georgia
           Southwestern University and Americus. To start, open the skill by saying,
           \u0027Alexa, open georgia southwestern quiz\u0027, and then after the welcome
           message you will be prompted to say \u0027start quiz\u0027. Enjoy.",
          "smallIconUri": "file://assets/images/en-US_smallIconUri.png",
          "largeIconUri": "file://assets/images/en-US_largeIconUri.png"
        }
      },
      "automaticDistribution": {
        "isActive": false
      },
      "isAvailableWorldwide": true,
      "testingInstructions": "Say \u0027open georgia southwestern quiz\u0027 to start the
           skill and then \u0027start quiz\u0027 to start the quiz. It\u0027s just a
           simple quiz.",
      "category": "KNOWLEDGE_AND_TRIVIA",
      "distributionMode": "PUBLIC",
      "distributionCountries": []
    },
    "apis": {
      "custom": {
```

```
      "endpoint": {
        "uri": "arn:aws:lambda:us-east-1:289211917748:function:0903e819-07d9-49bd-9ea3-
         0b44e6f8dd5c:Release_1"
      },
      "interfaces": [],
      "regions": {
        "EU": {
          "endpoint": {
            "uri": "arn:aws:lambda:eu-west-1:289211917748:function:0903e819-07d9-49bd-
          9ea3-0b44e6f8dd5c:Release_1"
          }
        },
        "NA": {
          "endpoint": {
            "uri": "arn:aws:lambda:us-east-1:289211917748:function:0903e819-07d9-49bd-
          9ea3-0b44e6f8dd5c:Release_1"
          }
        },
        "FE": {
          "endpoint": {
            "uri": "arn:aws:lambda:us-west-2:289211917748:function:0903e819-07d9-49bd-
          9ea3-0b44e6f8dd5c:Release_1"
          }
        }
      }
    },
    "manifestVersion": "1.0",
    "privacyAndCompliance": {
      "allowsPurchases": false,
      "locales": {
        "en-US": {}
      },
      "containsAds": false,
      "isExportCompliant": true,
      "isChildDirected": false,
      "usesPersonalInfo": false
    }
  }
}
```


## 5. CONCLUSION AND FUTURE WORK

With this study the oncoming demand of Speed of delivery in software development lifecycle is important that performance testing practice must be automated in the pipeline with automated reports that can provide realtime feedback. I think the future of Automating performance testing will be around Shift Left Automation. There are multiple ways the industry itself is looking at this topic however the more automation in the left to enable the performance testing process as early as in the software development lifecycle will be true success of this study. Also, some limitations include lacking a proper testbed and software development process. Testing this feature on a small environment versus a true software development may cause some results to vary. Testing also takes up a significant amount of time in each and every phase which might lead to the delay in production for larger software. This study can be deployed in testing small projects and if successful, can be extrapolated on much bigger ones. This can be even used in app development from a small scale app to much bigger and most downloaded ones like the one built on Alexa by author 2 mentioned above.

## 6. ACKNOWLEDGEMENTS

We would like to thank the professors from the CS department at GSW for their continuous support on this project. We would also like to thank AWS for allowing us to rent their instance, the results of which were shown in this paper.

## REFERENCES


[1] André Janus (2018), "Towards a common Agile Software Development Model", *ACM SIGSOFT, July 2012 Volume 37 Number 4.*

[2] 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).*

[3] Marian Stoica, Marinela Mircea, Bogdan Ghilic-micu (2013), "Software Development: Agile vs. Traditional", *Informatica Economică vol. 17, no. 4/2013.*

[4] 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*

[5] https://f.hubspotusercontent30.net/hubfs/7652530/10-best-practices-app-performance-testing-071918.pdf

[6] https://softcrylic.com/blogs/performance-testing-for-devops/

[7] https://www.synopsys.com/blogs/software-security/continuous-testing-cicd/

[8] https://ieeexplore.ieee.org/abstract/document/4293621


## AUTHORS

**Suresh Kannan Duraisamy:** Suresh is a Master's student in the department of Computer Science at GSW. Along with pursuing Masters, he is a full time IT professional. SK also has more than 15 years of IT professional experience and is Specialized in transforming QA teams to adapt to the Technology modernization initiatives including Agile transformation, Cloud, Microservices, and Enabling Performance and Functional testing processes in DevOps -SRE culture by instrumenting Automation frameworks in the CI-CD pipeline.

**Bryce Bass:** Bryce is an undergraduate student in the department of Computer Science and IT Analyst/Programmer at GSW. Along with his studies, he works at the University IT office. He also developed the "GSW Trivia" Alexa app.

**Sai Mukkavilli:** Sai Mukkavilli is an Asst. Professor in the CS department at GSW. His area of interest and research are Cloud computing and security. Bryce and Suresh are Dr. Mukkavilli's former students.