NoVA DevSecOps / Continuous Testing Assessments

March 22, 2022 (Virtual)

Overview

I browsed through www.meetup.com for technology meetups related to System Administration and the NoVA (Northern Virginia) DevSecOps group peaked my interest through their weekly discussions on DevSecOps. This week, the presentation was of the Continuous Testing Assessment by a group of folks from Titania Solutions Group – a consulting company dedicated to servicing federal government entities. This group was contracted to CMS (Centers for Medicare & Medicaid Services) to ensure their applications are meeting continuous testing and integration standards to meet production deployment deadlines.

The first task the team accomplished is migrating all CMS application development from the Waterfall methodology to SAFe/Agile which helped them focus on DevSecOps and delivery scalability across the applications. The main component of DevSecOps maturity is to reduce cycle time of development, increase (safe) deployment frequencies, and reduced number of defects in production. DevSecOps ensure the safe deployment of an application lifecycle through application and security testing.

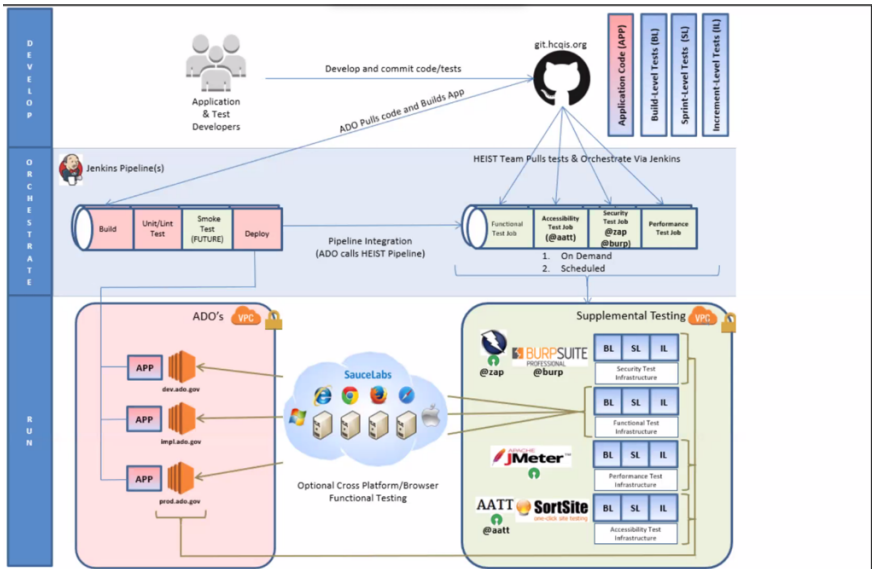
The Product

The Titania team had done several rounds of research and development to determine a way to quantify the maturity level of an application's lifecycle. A major source that they had gotten inspiration from is www.devops.com which offers a preliminary maturity assessment quiz.

They had constructed a two-pronged approach to determining an application's maturity level:

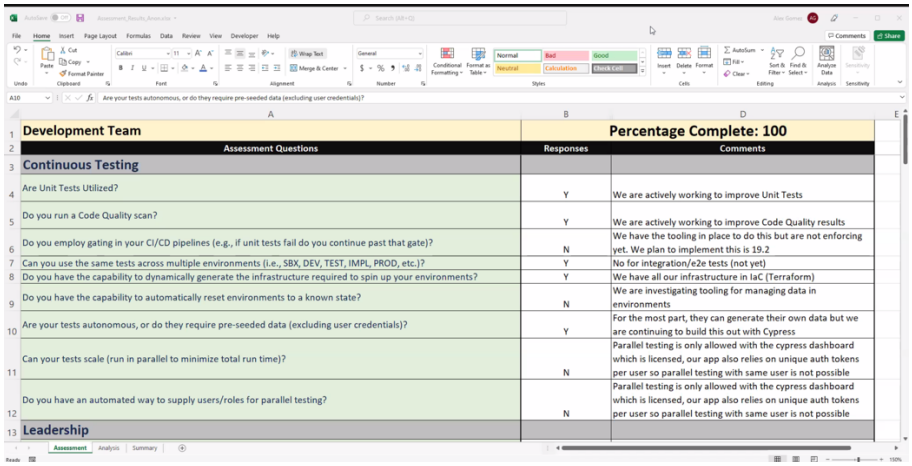
1. Functional components (code) – Are unit tests being used properly? Are testing scripts tagged properly (smoke, regression, integration)?
2. Non-functional components (culture, leadership) – Is the team consistently monitoring the testing results and improving their score? Are team members fully aware of the importance of application testing?

The functional assessment was done through the automated process in a Jenkins Pipeline. Jenkins is an open source automation tool that supports the CI/CD (Continuous integration/Continuous Delivery) flow by running automated scripts against all committed code in a repository. These scripts are often used for testing automation as well as code packaging. Some of the tools they used for application code testing are aatt for Accessibility, zap and burp for Security, and JMeter for performance. An overview of their architecture is shown on the following page. The output of each of these tools allowed for a quantifiable score of each application's testing maturity and security levels.



Jenkins Functional Testing Architecture

Unsurprisingly, the non-functional component of their maturity assessment involves a human component. The broad application team, including Developers, Architects, Information Security, Managers, System Administrators, was brought together to fill an automated questionnaire relating to the application lifecycle and security. The questionnaire produced results that the DevSecOps team would then have to review in order to provide the final score and recommendations. The first release of this questionnaire was a simple Excel sheet but they are in the midst of migrating this to a web application.

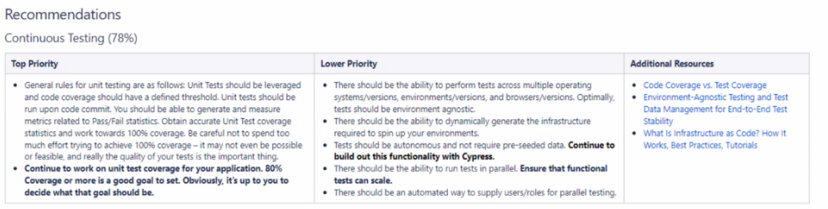


	A	B	D	E
1	**Development Team**		**Percentage Complete: 100**	
2	**Assessment Questions**	**Responses**	**Comments**	
3	**Continuous Testing**			
4	Are Unit Tests Utilized?	Y	We are actively working to improve Unit Tests	
5	Do you run a Code Quality scan?	Y	We are actively working to improve Code Quality results	
6	Do you employ gating in your CI/CD pipelines (e.g., if unit tests fail do you continue past that gate)?	N	We have the tooling in place to do this but are not enforcing yet. We plan to implement this is 19.2	
7	Can you use the same tests across multiple environments (i.e., SBX, DEV, TEST, IMPL, PROD, etc.)?	Y	No for integration/e2e tests (not yet)	
8	Do you have the capability to dynamically generate the infrastructure required to spin up your environments?	Y	We have all our infrastructure in IaC (Terraform)	
9	Do you have the capability to automatically reset environments to a known state?	N	We are investigating tooling for managing data in environments	
10	Are your tests autonomous, or do they require pre-seeded data (excluding user credentials)?	Y	For the most part, they can generate their own data but we are continuing to build this out with Cypress	
11	Can your tests scale (run in parallel to minimize total run time)?	N	Parallel testing is only allowed with the cypress dashboard which is licensed, our app also relies on unique auth tokens per user so parallel testing with same user is not possible	
12	Do you have an automated way to supply users/roles for parallel testing?	N	Parallel testing is only allowed with the cypress dashboard which is licensed, our app also relies on unique auth tokens per user so parallel testing with same user is not possible	
13	**Leadership**			

Non-Functional Questionnaire Example

After both assessments are completed, the DevSecOps team would then review the final scores and the Maturity Assessment product creates a report for the application team to digest. The final report consists of maturity score across various categories which is seen in the screenshot. In the event that an application had a low score in any category, the Maturity Assessment product provided recommendations on how to improve this area which sometimes included code snippets and documentation. They also had the ability to upload this report to a

Confluence (Internal Wiki) page for the application teams to digest in a more user-friendly fashion. Application teams enjoyed this view and were eager to get re-evaluated after making code changes in each Sprint. I think this was a great idea that appeals to Game Theory, who doesn't love to see a higher score after you put in tons of work?



Recommendations

Continuous Testing (78%)

Top Priority	Lower Priority	Additional Resources
• General rules for unit testing are as follows: Unit Tests should be leveraged and code coverage should have a defined threshold. Unit tests should be run upon code commit. You should be able to generate and measure metrics related to Pass/Fail statistics. Obtain accurate Unit Test coverage statistics and work towards 100% coverage. Be careful not to spend too much effort trying to achieve 100% coverage – it may not even be possible or feasible, and really the quality of your tests is the important thing. • **Continue to work on unit test coverage for your application. 80% Coverage or more is a good goal to set. Obviously, it's up to you to decide what that goal should be.**	• There should be the ability to perform tests across multiple operating systems/versions, environments/versions, and browsers/versions. Optimally, tests should be environment agnostic. • There should be the ability to dynamically generate the infrastructure required to spin up your environments. • Tests should be autonomous and not require pre-seeded data. **Continue to build out this functionality with Cypress.** • There should be the ability to run tests in parallel. **Ensure that functional tests can scale.** • There should be an automated way to supply users/roles for parallel testing.	• Code Coverage vs. Test Coverage • Environment-Agnostic Testing and Test Data Management for End-to-End Test Stability • What is Infrastructure as Code? How it Works, Best Practices, Tutorials

Confluence Result Showing Recommendations and Resources

Maturity Level Legend

Maturity Level	Range
Requires Focus	0% - 65%
Good	66% - 85%
Mature	86% - 100%

Category	9/1/2021 Score	3/1/2022 Score
Continuous Testing	58%	78%
Leadership	75%	75%
Culture	90%	100%
Test Strategy and Plans	73%	81%
Test Management	88%	88%
Test Automation	50%	60%
Test Tools	76%	81%
Test Infrastructure Management	49%	99%
Test Scripts	90%	100%
Test Results	70%	74%
Security Tests	93%	93%
Integration Testing	80%	80%
Pre-Production Testing	95%	97%
Test in Production	80%	80%

Maturity by Category

* This chart shows both previous scores (9/1/2021) in blue and latest scores (3/1/2022) in orange.

Maturity Assessment Over Time

Learnings & Conclusion

The DevSecOps went through many hurdles during the development of their Maturity Assessment framework. The one that was least surprising is to handle many different codebases, the enterprise application code included .NET, ReactJS, and Java so the framework had to be versatile. Another major difficulty was to gather the broad application team to fill out the non-functional maturity questionnaire. When you have many people that have different roles, you are bound to have conflicting thoughts which led to some pain points in determining the final score.

A major takeaway from this product demonstration is teams are not as focused on security testing as they should be! There are many tools out there that have automated security testing such as ZAP and BURP that should be integrated in every CI/CD pipeline. In this day and age, this is vital for any application whether it is internal or external facing.

Overall, I highly enjoyed the discussion from Titania Solutions Group showcasing their framework. The community asked thought-provoking questions concerning the future of the product and intentions for other Federal applications migrating into this framework as well. The

correct path is to continuously test your application to ensure there are no disasters when an application goes live in production.

The speaker also made a plug of his latest obsession of laughter yoga which I thought was fun, check it out here: www.laughteryoga.org