Form Scriber DevSecOps

User Guide

Version 2.0

Ivy Pham

Vincent Leung

SWEN 670 Software Engineering Project

University of Maryland Global Campus

March 30, 2021

**REVISION HISTORY**

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 03/01/2021 | 1.0 | Initial draft |
| 03/02/2021 | 1.0 | Added Culture |
| 03/03/2021 | 1.1 | Added Git, Azure |
| 03/03/2021 | 1.2 | Added Language Style Guides |
| 03/04/2021 | 1.3 | Added Code Quality |
| 03/05/2021 | 1.3 | Edited |
| 03/15/2021 | 1.4 | Edited |
| 03/15/2021 | 2.0 | Final Review |

Contents

[1. DevSecOps Culture 4](#_Toc66728650)

[2. Git Tips and Best Practices 7](#_Toc66728651)

[3. Code Review Process 9](#_Toc66728652)

[4. Principles and Style Guides 10](#_Toc66728653)

[5. Code Quality 11](#_Toc66728654)

[6. Developer Access to Azure Tools 17](#_Toc66728655)

[7. Code Review Checklist Example 18](#_Toc66728656)

[8. References 20](#_Toc66728657)

# 1. DevSecOps Culture

DevSecOps starts with culture. It involves the entire team to adapt Agile, shifting development “to the left”, and engaging in automation. In the context for development teams, this means business analysis and quality assurance practices are considered by the entire team instead of siloed and only expected from certain team member(s) with specific titles. The whole team ensures testable requirements and testable code throughout the lifecycle for smooth integration with the CI/CD pipeline services downstream. Figure 1 demonstrates some of the important elements required for efficient collaboration in an organization with a DevOps mindset.

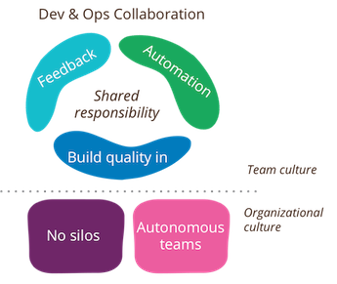


Figure 1. DevOps Culture

*Note*. DevOps elements visual from martinfowler.com

Implementing test automation is a crucial element for the development teams to align with DevSecOps culture. Test automation allows for testing and verification of software to be done earlier, faster, more frequently, with more accuracy than with manual efforts, and enables the shift to the left by identifying issues earlier in the development cycle. Manual testing therefore should be reserved for instances where automation does not yet have the context to run well, such as usability, user acceptance, and ad-hoc testing. Consider the test automation pyramid below.

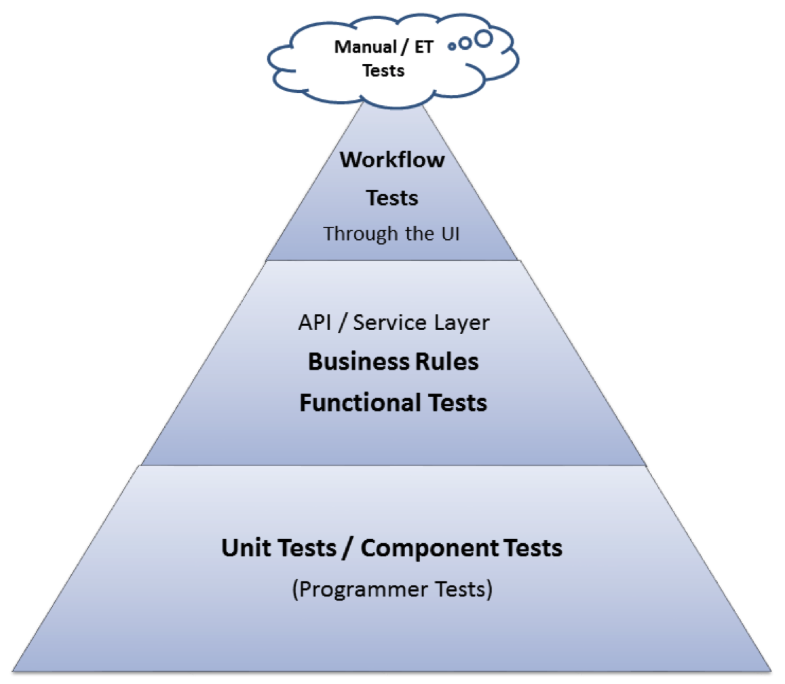


Figure 2. Testing Automation Pyramid

*Note*. Test automation pyramid for test effort recommendations, from dzone.com

From Figure 2, we can adopt the guidelines for test automation coverage:

* Most of the tests should be unit tests and API or integration/service layer functional testing
* A small percentage of tests are the automated UI testing and manual testing

Many industry tools and frameworks are available for every type of test automation effort, many of which are open-source.

To ensure a complete implementation of DevSecOps, security best practices must also be considered. As developers are responsible for security of their code, project management needs to enable security practices by promoting a culture of openness and communication about the code produced. Project management needs to ensure that development teams are up to date on cybersecurity best practices via training and learning resources. This mitigates risks and legality possibilities, especially when working on products within certain sectors.

By doing this, security is embedded into development from the start rather than a final step before deployment, when it often becomes extremely costly to remediate vulnerabilities. Another intra-team effort to ensure security practices is through code reviews by peers. Additional eyes on team members’ code provides invaluable insight, especially from experienced developers to junior ones. Continuous feedback allows development team members to fix their code and learn best practices when and where needed. Working collectively with this mindset is a way towards a more secure codebase and represents a proactive approach versus relying strictly on automated code analysis tools.

# 2. Git Tips and Best Practices

Version control system has been selected for this project utilizing the free tier of GitHub. Changes are managed through GitHub throughout the development cycle. The code repository design is documented in the DevSecOps Technical Design Document. To keep the design simple and effective, DevSecOps team has deployed 3 tiers of Git branches (Figure 3). The first branch is the master branch which is controlled by DevSecOps team to check for the success of build, unit test and code quality before each component of application is being released to a specified environment for the full integration testing. The second branch is the development branch which allows each development team to collaborate. The development branch is controlled by each team lead which is set with policies to promote code review processes before merging. The third branch is the feature branch which allows developers to create the new branch for each feature being worked on.

Diagram

Description automatically generated

Figure 3. Branch Overview

Below is the tips and best practices recommended by the industry.

1. **Agree on a branch naming convention for your team.** Keeping names of feature/integration branches consistent helps with code history tracking, organization, and readability.
2. **Use a .gitignore file.** Some files should not or do not need to be tracked, such as dependencies, log files, auto-generated files, or files with sensitive information.
3. **Use meaningful commit messages.** Write a short, descriptive summary of the change(s) that the commit contains. Aim to provide context for the change.

**git commit –m “added feature”**

versus

**git commit –m “Added automatic logout after user idles for 15 minutes”**

1. **Make small commits.** This keeps the feedback cycle short and easier to manage changes shall the need arise.
2. **Commit frequently.** This minimizes the occurrences of merge conflicts.
3. **Push working code only.** Test your code to ensure you are not committing broken code, which will affect other team members.
4. **Agree on a schedule to merge to master.** This sets expectations for development team members and creates a rhythm. Smaller merges are more manageable for code reviewers.
5. **Use pull requests.** This standardizes merges to the main branch and allows for code reviews, helping with codebase stability.
6. **Address pull requests in a timely manner.** This lessens chances of conflicts that may occur if they are unattended while other development continues.
7. **Delete branches from the remote repo after merging.** This keeps the repository clean and easy to navigate between active branches.

# 3. Code Review Process

Each development team is responsible for code reviews following the recommended industry guidelines. Please see Principles and Style Guides section. The DevSecOps team will do the overall codebase review in additional to each team’s internal code review to catch issues that are not caught from the development team review. However, DevSecOps team will not try to identify every issue as resources are limited. Each development team shall follow the code review process below.

* Step 1: A team member pulls the code from GitHub to check for code standards.
* Step 2. The reviewer reviews the code and recommends changes if there any.
* Step 3. The changes are proposed to the team and documented for implementation.

Peer review is also recommended from each development team to catch errors and check for code standards earlier in the process. Each development team will follow the peer review process below.

* Step 1: A pull request is initiated from a team member to merge from a feature branch to the development branch.
* Step 2: The lead developer reviews the pull to check for issues including code standards.
* Step 3: The lead developer either rejects or approves the merge.

DevSecOps will follow a similar process to check for code standards and quality.

* Step 1: A pull request is initiated from a team member to merge from development branch to the master branch as agreed upon.
* Step 2: A DevSecOps team member reviews the pull request for issues including unit test, code standards and quality either manually or automatically utilizing the Azure and GitHub pipeline.

# 4. Principles and Style Guides

It is expected that developers shall follow conventions particular to coding languages, and to also employ object-oriented programming fundamentals, design patterns, and software engineering best practices. Remember some acronyms:

* DRY (Don’t Repeat Yourself)
* KISS (Keep It Simple, Stupid)
* YAGNI (Ya Ain’t Gonna Need It)

The general standard is created for the development team to follow regardless of the programming language being used.

* Proper and consistent indentation (minimum of three spaces)
* Inline comments to explain the function
* Classes, subroutines, functions, and methods shall be kept in a reasonable size and not too large
* Source files and variable names shall represent its functions
* Keep line length short and utilize the wrapping line break expression
* Meaningful error messages

Below are the coding languages used by the development teams for the project. Rather than repeat or recreate what has already been elucidated fully by leaders in the industry, the following resources provided offer style guides for each language and represent the standards recommended to development teams.

**Dart:**

<https://dart.dev/guides/language/effective-dart/style>

**Go:**

<https://golang.org/doc/effective_go>

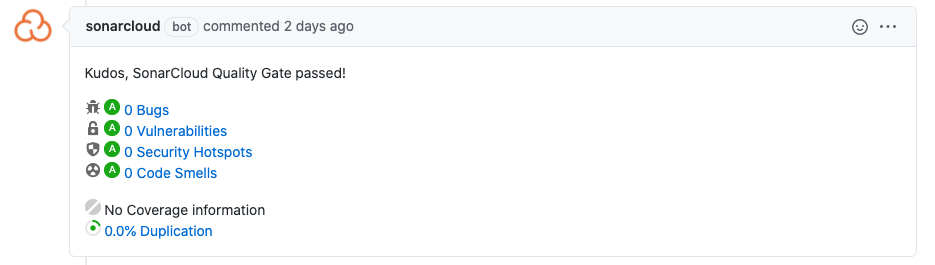
**HTML/CSS:**

<https://google.github.io/styleguide/htmlcssguide.html>  
  
  
**JavaScript:**

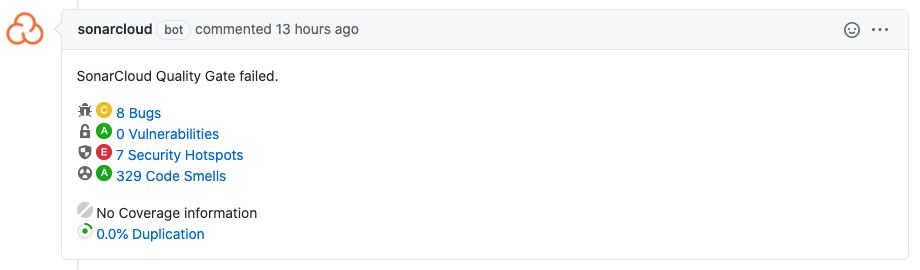
<https://google.github.io/styleguide/jsguide.html>

# 5. Code Quality

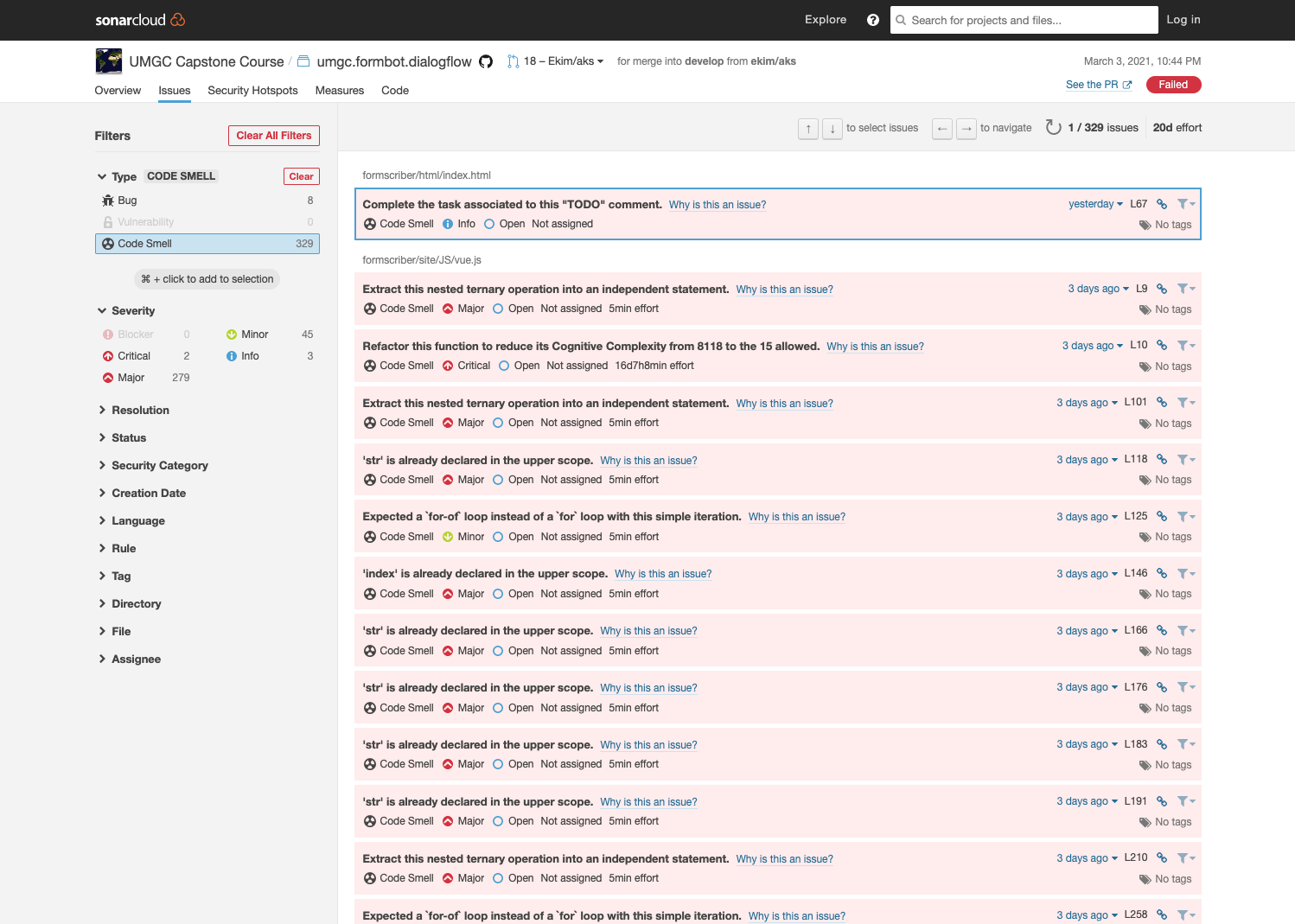
Code quality and security are enabled via SonarCloud integration with the GitHub repositories. When a pull request is made, code analysis and vulnerability scanning is executed and provide a report summary which is included in the history of the **Conversation** tab within the pull request. (Figure 4)

Figure 4. SonarCloud Code Quality Check (passed)

When issues are found, the appropriate change in statuses are shown with links to each type of issues detailed in the project within SonarCloud. (Figure 5)

Figure 5. SonarCloud Code Quality Check (failed)

Clicking on the issues brings up the Sonarcloud dashboards where each recommendation can be viewed by expanding the finding. (Figure 6 & 7)

Figure 6. SonarCloud Fix Recommendation

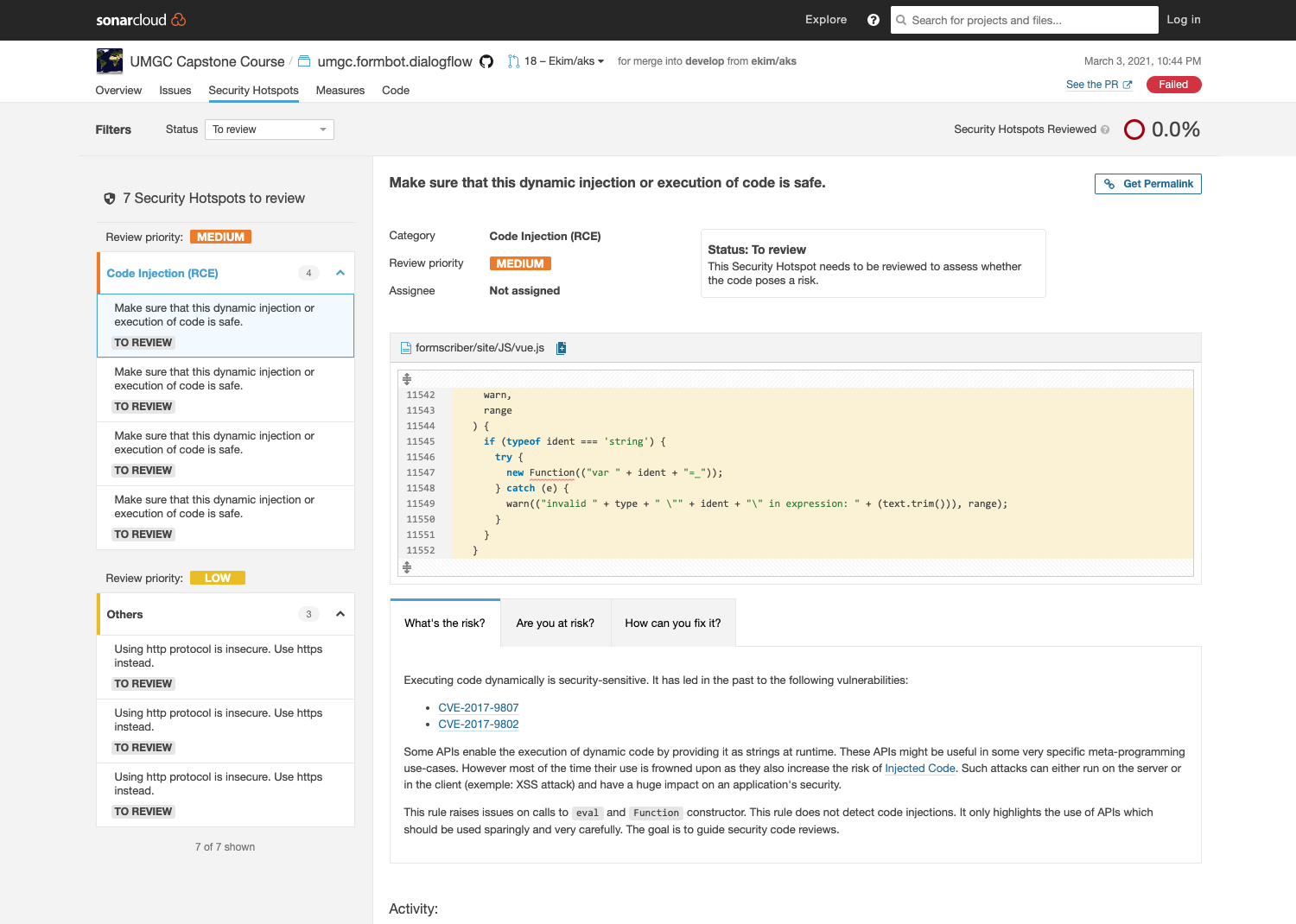
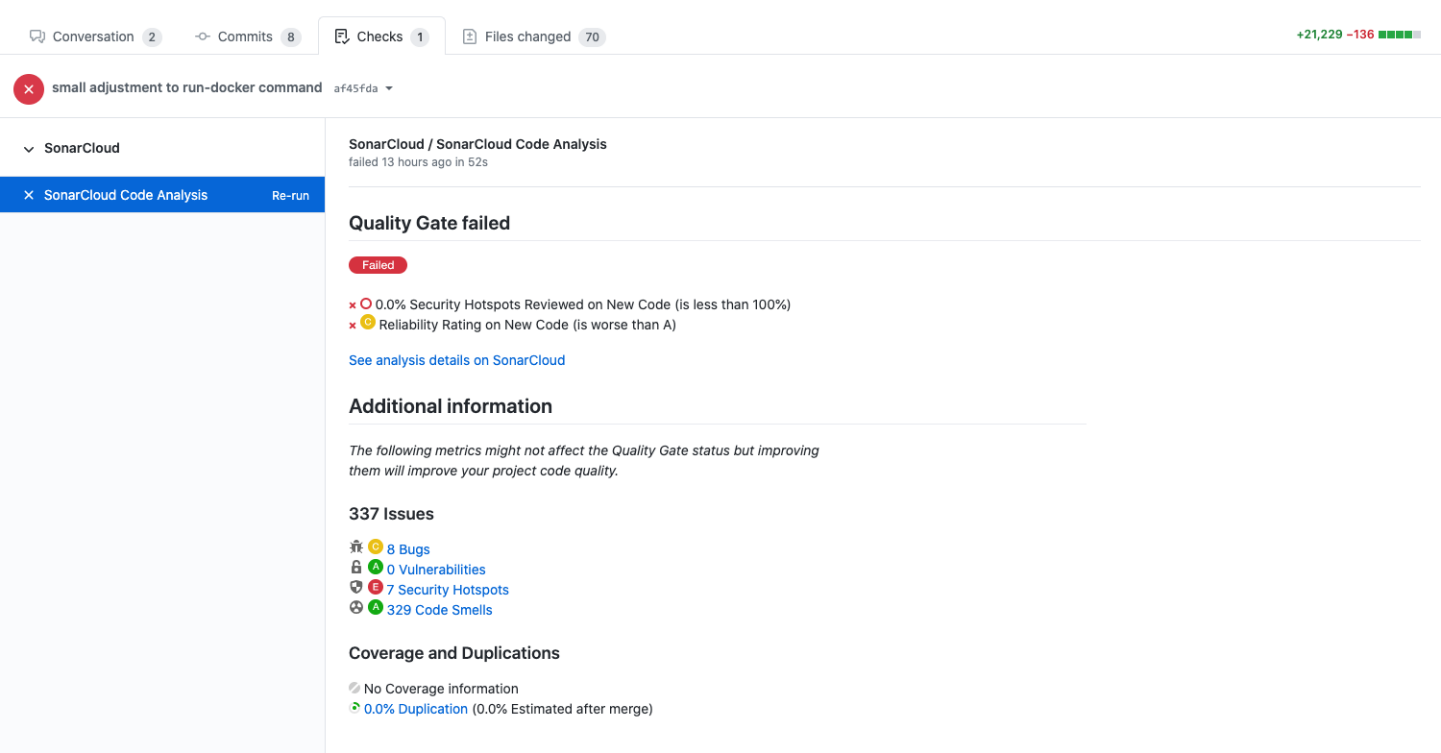


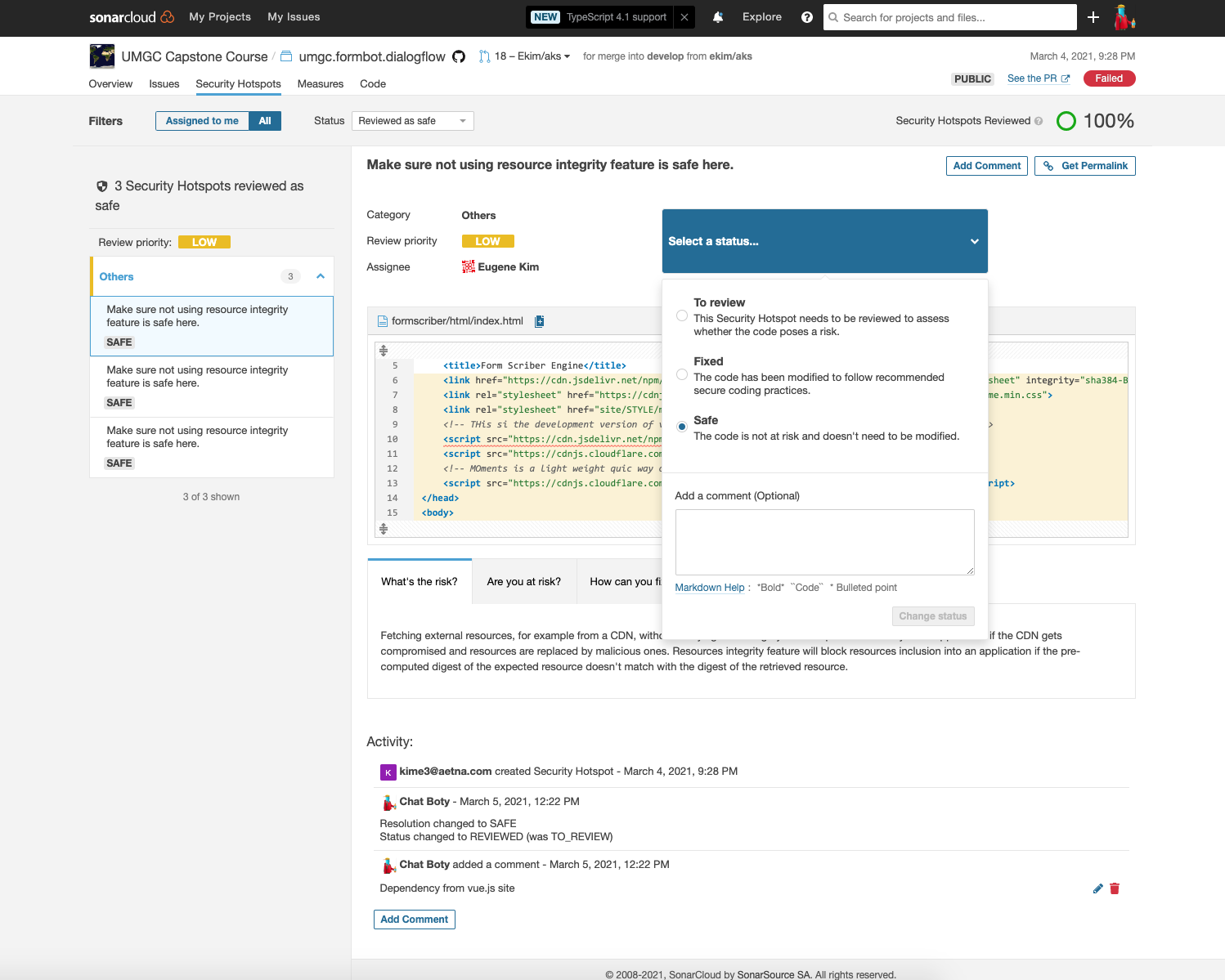
Figure 7. SonarCloud Fix Recommendation in details

When issues are resolved, a re-scan can be done by navigating to the **Checks** tab of the pull request and selecting **Re-run** in the left menu. (Figure 8)

Figure 8. SonarCloud Re-run

While ideally there should be no issues noted, when issues arise from components either within or external to Form Scriber code, an effort should be made by the development team as a whole to acknowledge major issues to either address within the development cycle or to prioritize to create a backlog.

When there are **Security Hotspots** noted, the merge would not be possible without DevSecOps team review. The DevSecOps member should review and communicate it to the development team and add a comment, then change the status to either **Fixed** or **Safe**. (Figure 9)

Figure 9. SonarCloud Review

After the status for the issue has been updated, the change in status will be updated automatically within the Pull Request in GitHub. After checks are all green, the merge can be done. (Figure 10)

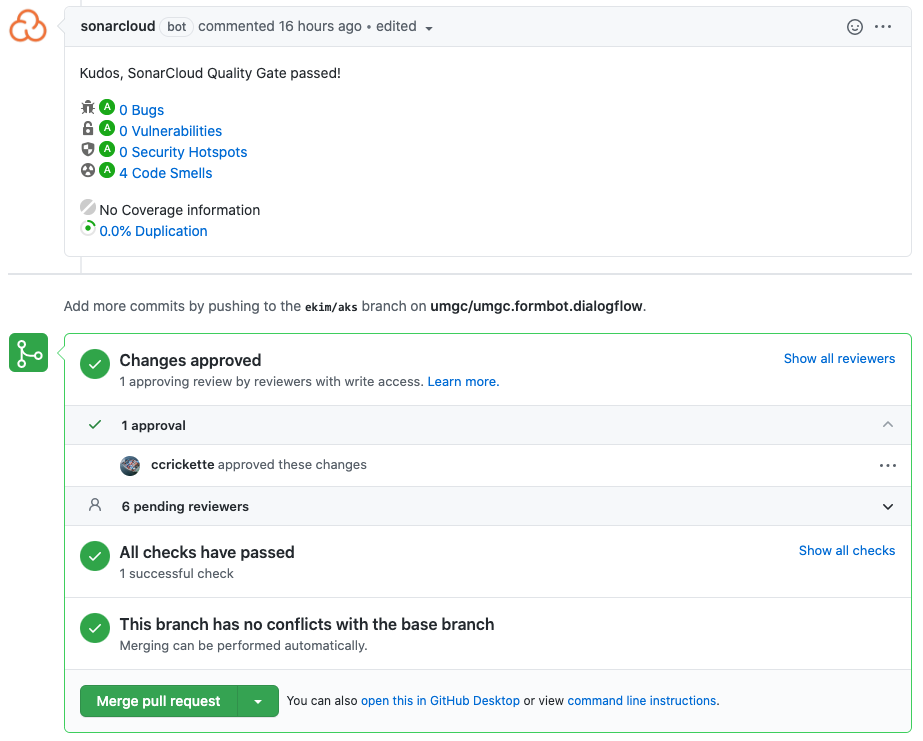


Figure 10. SonarCloud Status

# 6. Developer Access to Azure Tools

Microsoft offers a couple options to get started with free access to Azure tools that developer teams may utilize. As of Spring 2021, Microsoft offers a $100 credit for a Students account, redeemable for up to twelve months and renewable each calendar year so long as you are a student. This account can be set up at the link below.  

<https://azure.microsoft.com/en-us/free/students/>

Using your UMGC student account to register, you will gain access to dozens of tools that you can use for free or apply your credits towards. 

In addition to the Students account, Azure also provides a general free account that provides $200 credit to use within 30 days, and twelve months access to the free tools. This account can be set up at the link below. 

<https://azure.microsoft.com/en-us/free/>

Examples of developer resources and tools available for the accounts include virtual machines, storage, databases, compute services, app services, test environments, AI tools, and more. Review the list of tools that are free or free for a limited time/credit allowance. Some services, although listed as Free, may integrate with others that will require the use of credits, so apportioning out credits by scheduling usage towards production deployment is recommended.

# 7. Code Review Checklist Example

Check List

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **ITEM** | **Y, N, N/A** |
| Comment Block | Code respects the project coding conventions |  |
| The variable delectations are properly commented |  |
| All functions, methods and classes documented |  |
| Complex algorithms and code optimization adequately are commented |  |
| The comment blocks are consistent and of the same size |  |
| Code Checks | Constants are literal and not hardcoded |  |
| The structure is clean and indentations correct |  |
| All variables used have distinct or descriptive names and correct scope. |  |
| Code performs intended functions and operates correctly |  |
| Classes, subroutines, and functions are written in small size |  |
| Meaningful error messages |  |
| Unit Test Code Checks | Tests are written for each unit |  |
| Tests matches requirements |  |
| Tests are written for security aspects |  |
| Tests are reliable |  |

Code Review Record Forms

|  |  |
| --- | --- |
| Change | Description of implementation |
|  |  |
|  |  |
|  |  |

Other code review comments

|  |
| --- |
|  |
|  |

# 8. References

Crispin, L. (2018, October 21). Your automated testing strategy: Pyramids, triangles, and beyond. Retrieved from <https://dzone.com/articles/your-automation-testing-strategy-pyramids-triangle>

dart.dev. (n.d.). Effective Dart: Style. Retrieved from <https://dart.dev/guides/language/effective-dart/style>

golang.org. (n.d). Effective Go. Retrieved from <https://golang.org/doc/effective_go>

Google. (n.d.). Google HTML/CSS style guide. Retrieved from <https://google.github.io/styleguide/htmlcssguide.html>

Google. (n.d.). Google Java style guide. Retrieved from <https://google.github.io/styleguide/javaguide.html>

Google. (n.d.). Google JavaScript style guide. Retrieved from <https://google.github.io/styleguide/jsguide.html>

Wilsenach, R. (2015, July 9). DevOpsCulture. Retrieved from <https://martinfowler.com/bliki/DevOpsCulture.html>