



Azure DevOps

Tooling and Code Analysis

.NET CORE

*DevOps technologies,
combined with people and
processes, enable teams
to implement CI/CD and
continually provide value
to customers.*

[HTTPS://AZURE.MICROSOFT.COM/EN-US/SOLUTIONS/DEVOPS/](https://azure.microsoft.com/en-us/solutions/devops/)

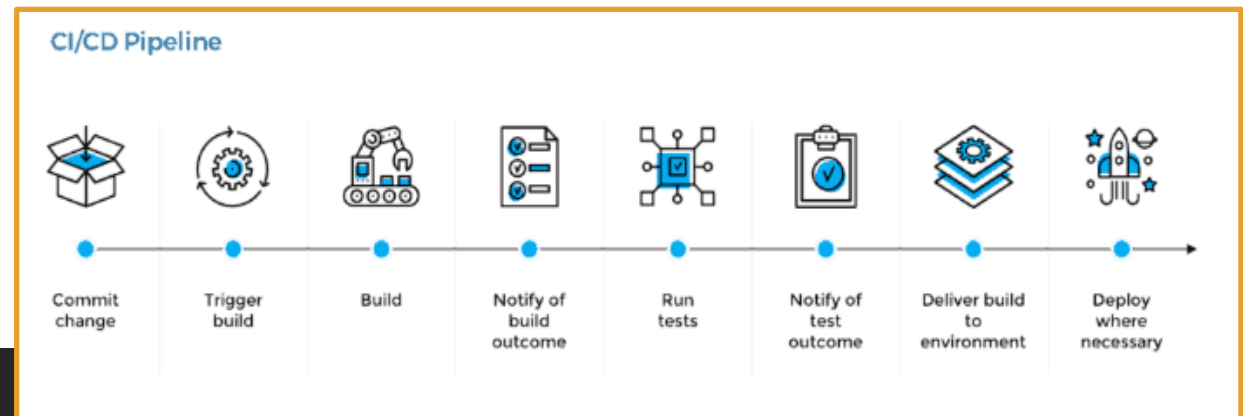
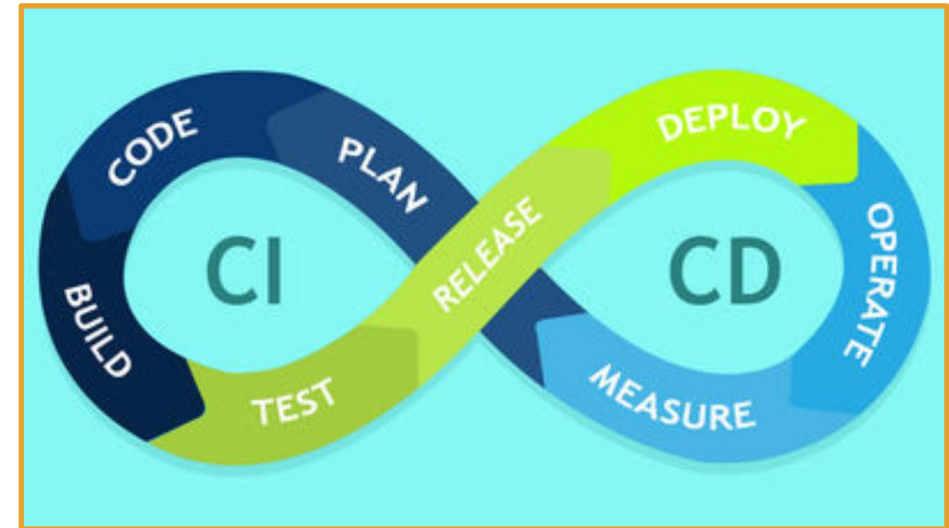
CI/CD and Continuous Testing (CT)

<https://docs.microsoft.com/en-us/azure/devops/pipelines/overview?view=azure-devops-2019>

Continuous Integration (CI) is the practice of automating the merging and testing of code. Implementing **CI** helps catch bugs early, which makes them less expensive to fix. Automated tests execute as part of the CI process.

Continuous Delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production environments to help improve product quality.

Continuous Testing (CT) is the use of automated build-deploy-test workflows that test your changes continuously in a fast, scalable manner.



Azure DevOps - Introduction

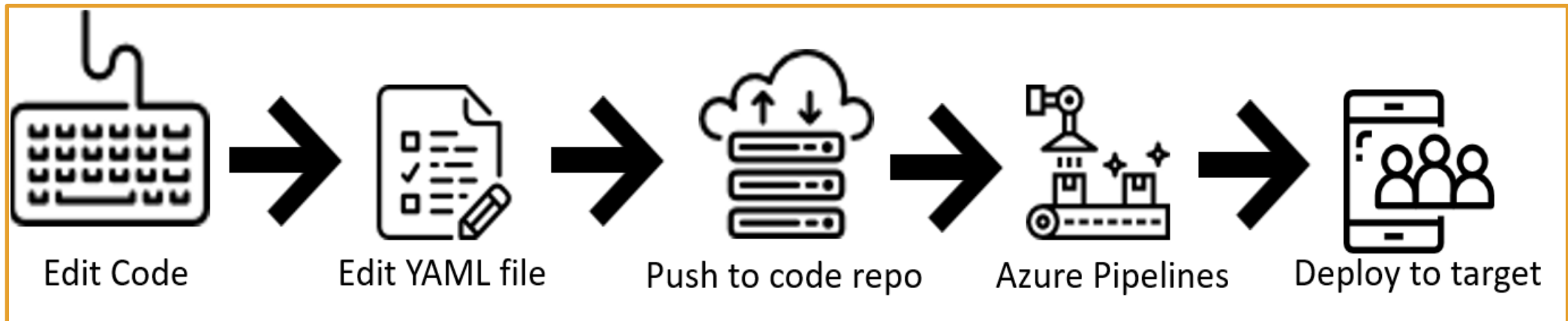
<https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/dotnet-core?view=azure-devops>

<https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops>

<https://docs.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops>

Azure Pipelines is a cloud service that you can use to automatically build and test your code and make it available to other users. **Azure Pipelines** works with many language or project types.

Azure Pipelines combines *Continuous Integration (CI)* and *Continuous Delivery (CD)* to constantly test and build your code to be shipped to any target.

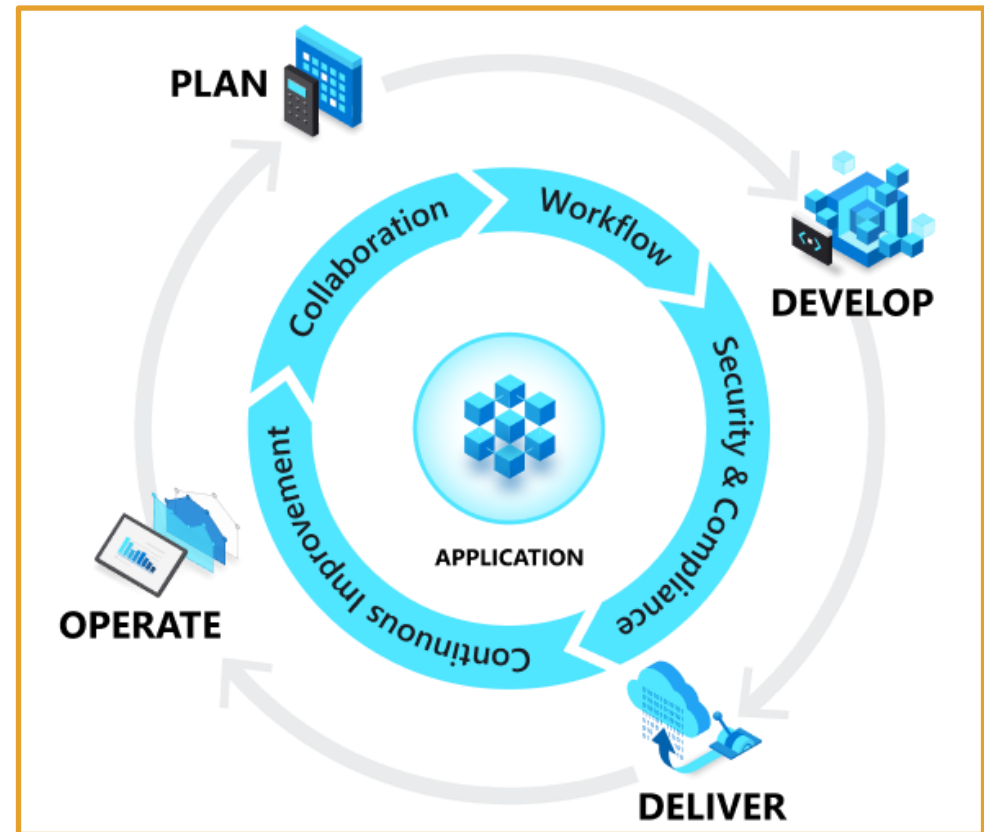


Build Definition

<https://docs.microsoft.com/en-us/aspnet/web-forms/overview/deployment/configuring-team-foundation-server-for-web-deployment/creating-a-build-definition-that-supports-deployment#task-overview>

A **build definition** is the mechanism that controls how and when builds occur. **Azure DevOps** uses a **.yaml** file to define a build. Each build definition specifies:

- The things you want to build.
- The criteria that determine when a build should take place
- The location to which the Build should send build outputs.
- The amount of time that each build should be retained.
- Various other parameters of the build process.

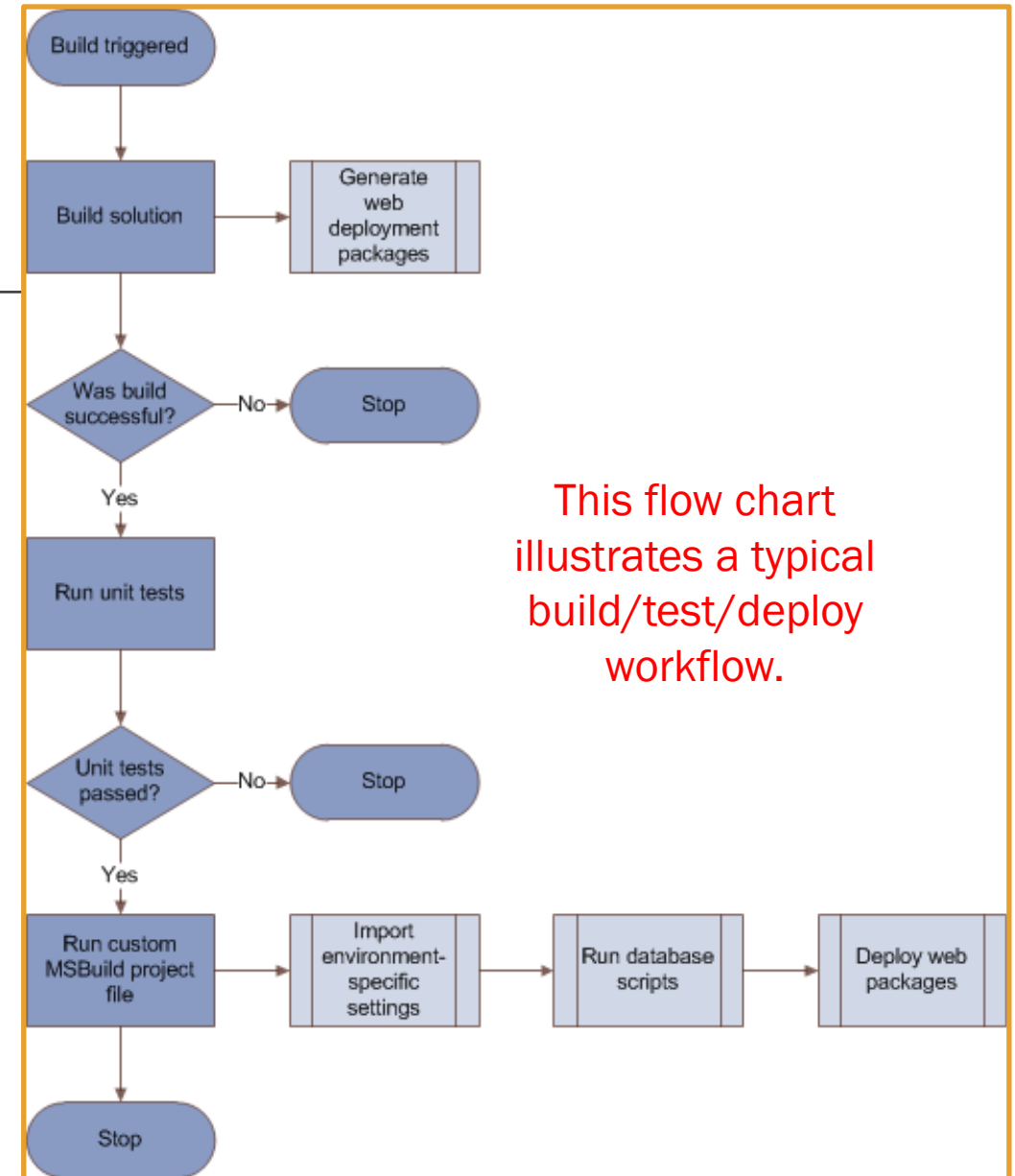
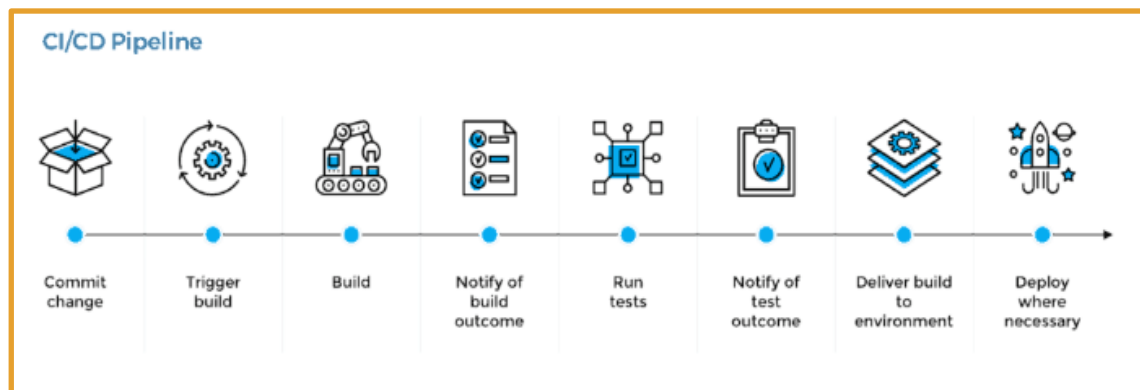


Release Pipeline

<https://docs.microsoft.com/en-us/azure/devops/pipelines/release/?view=azure-devops>

Release pipelines in **Azure Pipelines** help your team implement CI/CD and deliver software to your clients faster and with lower risk.

You can fully automate the testing, delivery, and analysis of your software all the way to production or set up semi-automated processes with required approvals and on-demand deployments.



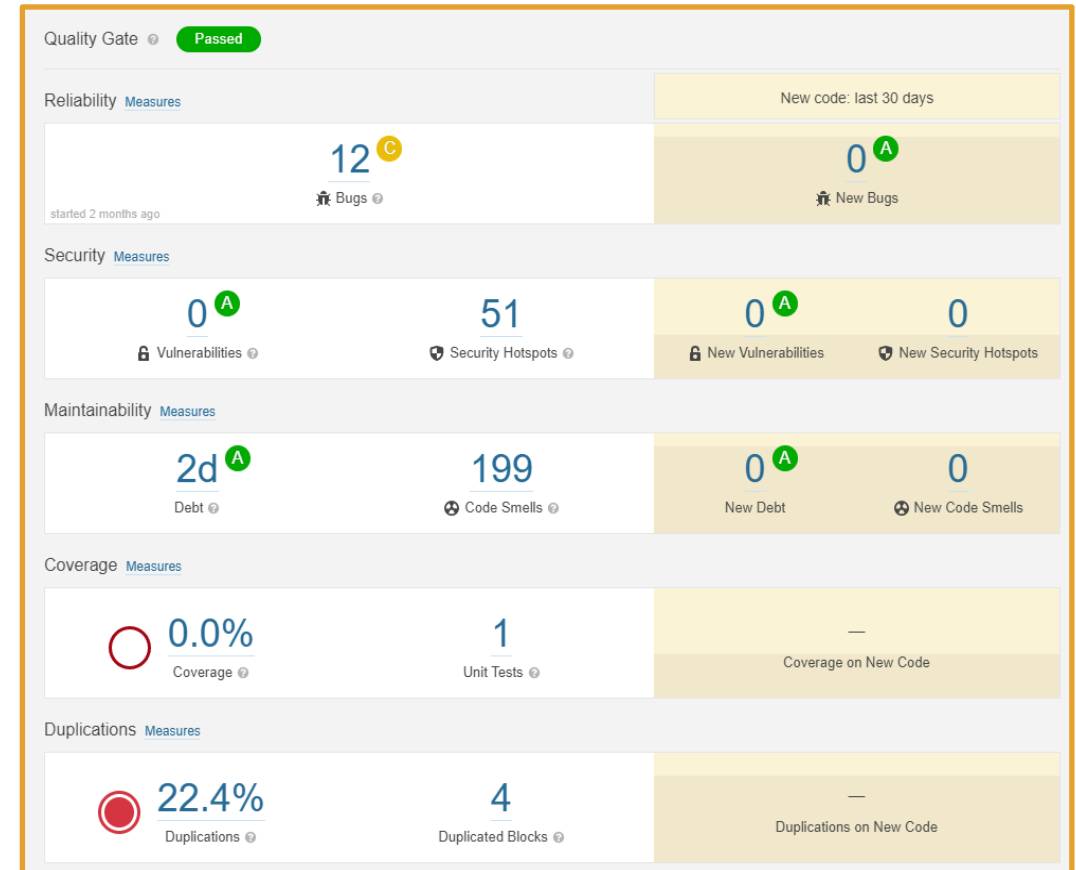
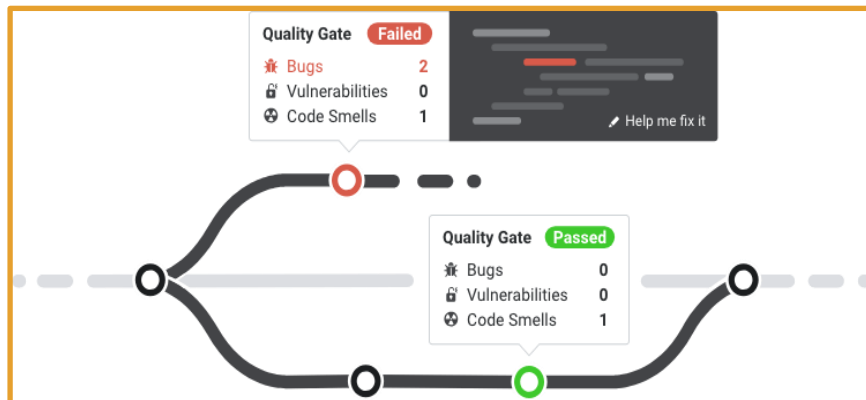
What is Static Code Analysis?

https://en.wikipedia.org/wiki/Static_program_analysis

Static code analysis is the analysis of computer software performed without executing the program. **Static code analysis** is usually performed on the source code.

The term is usually applied to the analysis performed by an automated tool. SonarCloud and SonarQube are popular **Static Code Analysis** tools.

Human analysis is called **Code Review**.



What is a Coverage Review?

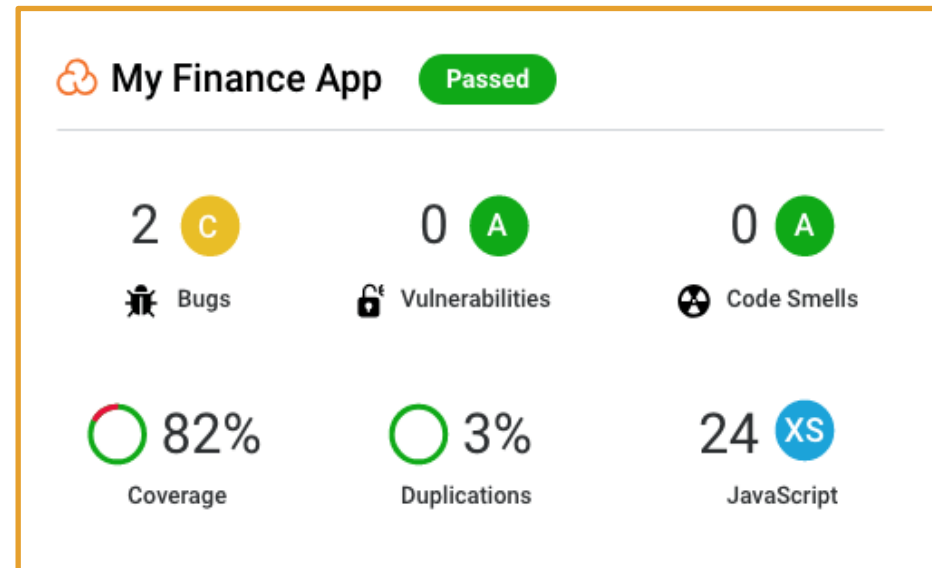
<https://sonarcloud.io/documentation/user-guide/metric-definitions/>
<https://sonarcloud.io/documentation/user-guide/concepts/>

How much of the source code has been covered by the unit tests?

Code Coverage is determined by evaluating what percentage of the total lines of code are covered by unit testing. It is a mix of Line coverage and Condition coverage.

$$\text{Coverage} = (CT + CF + LC) / (2*B + EL)$$

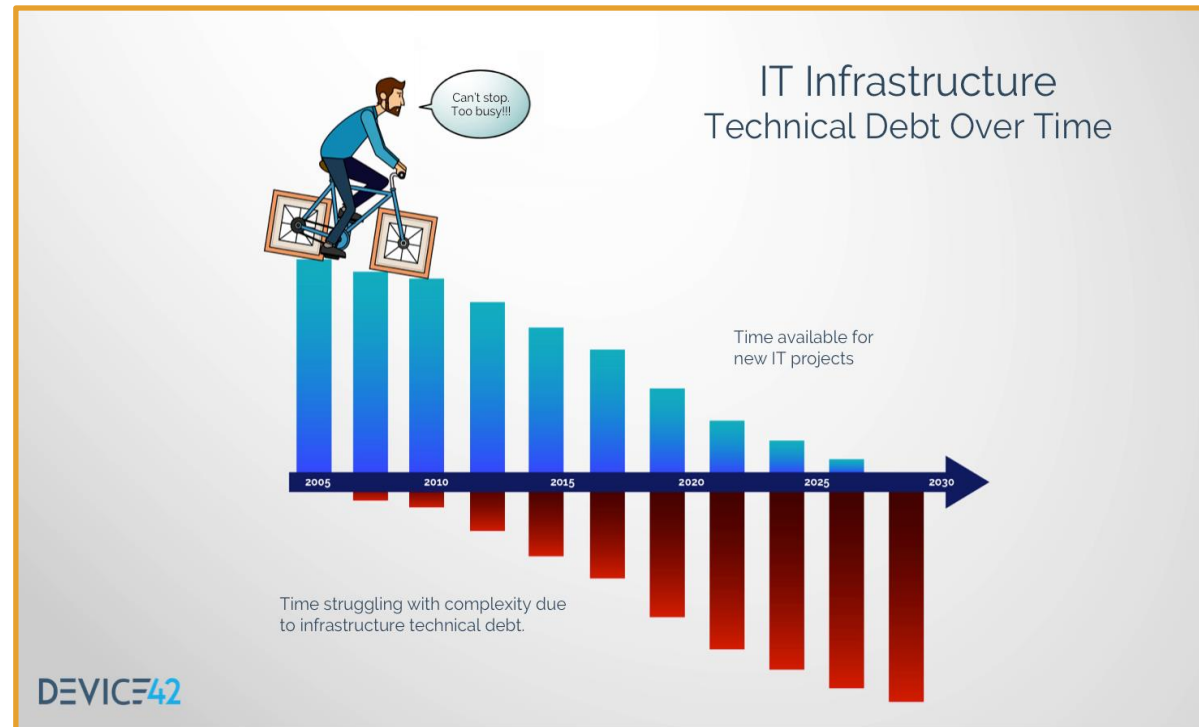
- CT = conditions that have been evaluated to 'true' at least once
- CF = conditions that have been evaluated to 'false' at least once
- LC = covered lines = lines_to_cover - uncovered_lines
- B = total number of conditions
- EL = total number of executable lines (lines_to_cover)



Technical Debt

<https://sonarcloud.io/documentation/user-guide/concepts/>

Technical Debt is the estimated time required to fix all Maintainability Issues/code smells.

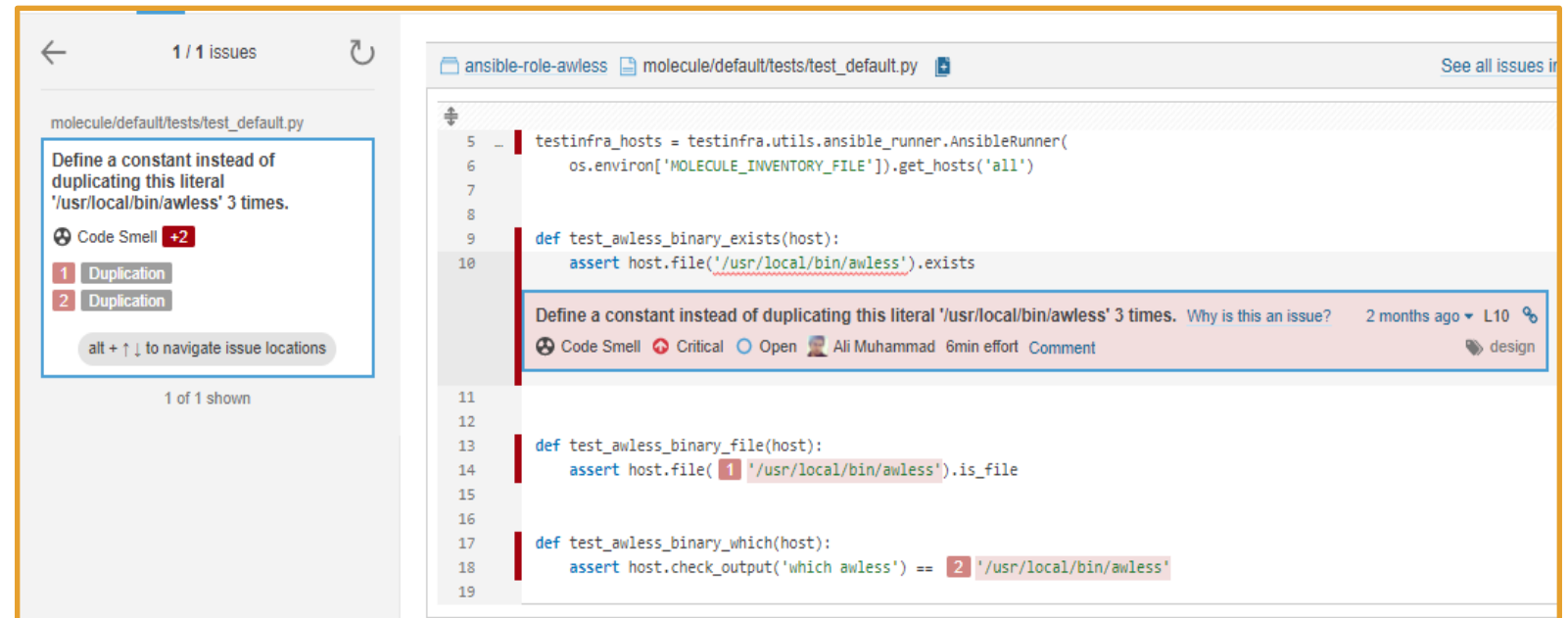


What is a Code Smell?

<https://sonarcloud.io/documentation/user-guide/concepts/>
https://sonarcloud.io/project/issues?id=ansible-role-awless&open=AXEseUF1IRsecPgXK050&resolved=false&types=CODE_SMELL

A **Code Smell** is any characteristic in the source code of a program that possibly indicates a deeper problem. Determining what is and is not a **Code Smell** is subjective, and varies by language, developer, and development methodology.

A **Code Smell** is an issue with long-term maintainability in the code. Leaving it as-is means that it will be more difficult for maintainers to make changes to the code. They'll risk introducing new errors as they make changes.



The screenshot displays the SonarCloud interface for a project named 'ansible-role-awless'. On the left, a sidebar shows '1 / 1 issues' and a list of two 'Duplication' issues. The main panel shows the source code of 'molecule/default/tests/test_default.py'. A red box highlights a 'Code Smell' issue with the message: 'Define a constant instead of duplicating this literal '/usr/local/bin/awless' 3 times.' The issue is marked as 'Critical' and 'Open'. The code snippet shows three function definitions, each using the literal path '/usr/local/bin/awless' in an assertion.

```
5 - testinfra_hosts = testinfra.utils.ansible_runner.AnsibleRunner(
6     os.environ['MOLECULE_INVENTORY_FILE']).get_hosts('all')
7
8
9 def test_awless_binary_exists(host):
10     assert host.file('/usr/local/bin/awless').exists
11
12
13 def test_awless_binary_file(host):
14     assert host.file(1 '/usr/local/bin/awless').is_file
15
16
17 def test_awless_binary_which(host):
18     assert host.check_output('which awless') == 2 '/usr/local/bin/awless'
19
```

Duplication

https://sonarcloud.io/component_measures?id=microsoft_vscode-python&metric=Duplications

Duplication in code analysis indicates lines of code that are identical and could theoretically be separated into a method to be called or resolved using SOLID or DRY principles.

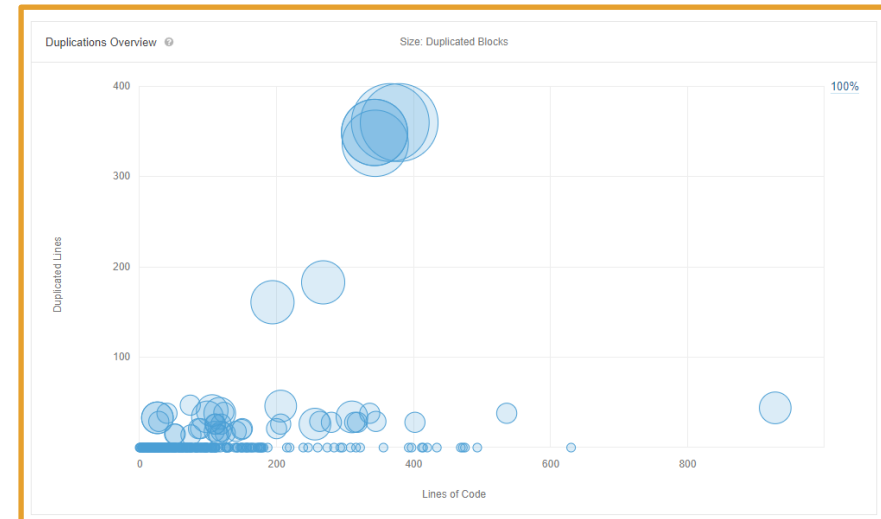
Duplications OVERVIEW	
Overview	
On new code	
Density	0.7%
Duplicated Lines	85
Duplicated Blocks	8
Overall	
Density	2.3%
Duplicated Lines	3,178
Duplicated Blocks	81
Duplicated Files	46

Inheritance Principle: Example

Duplicated Code

```
public class BusinessCustomer
{
    public int CustomerId { get; set; }
    public string CompanyName { get; set; }
    public string Address { get; set; }
    public string PostalCode { get; set; }
    public string Country { get; set; }
    public string Telephone { get; set; }
    public List<Order> Orders { get; set; }
    public DateTime Created { get; set; }
    public DateTime Modified { get; set; }
}

public class PrivateCustomer
{
    public int CustomerId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Title { get; set; }
    public string Address { get; set; }
    public string PostalCode { get; set; }
    public string Country { get; set; }
    public string Telephone { get; set; }
    public List<Order> Orders { get; set; }
    public DateTime Created { get; set; }
    public DateTime Modified { get; set; }
}
```



Quality Gate

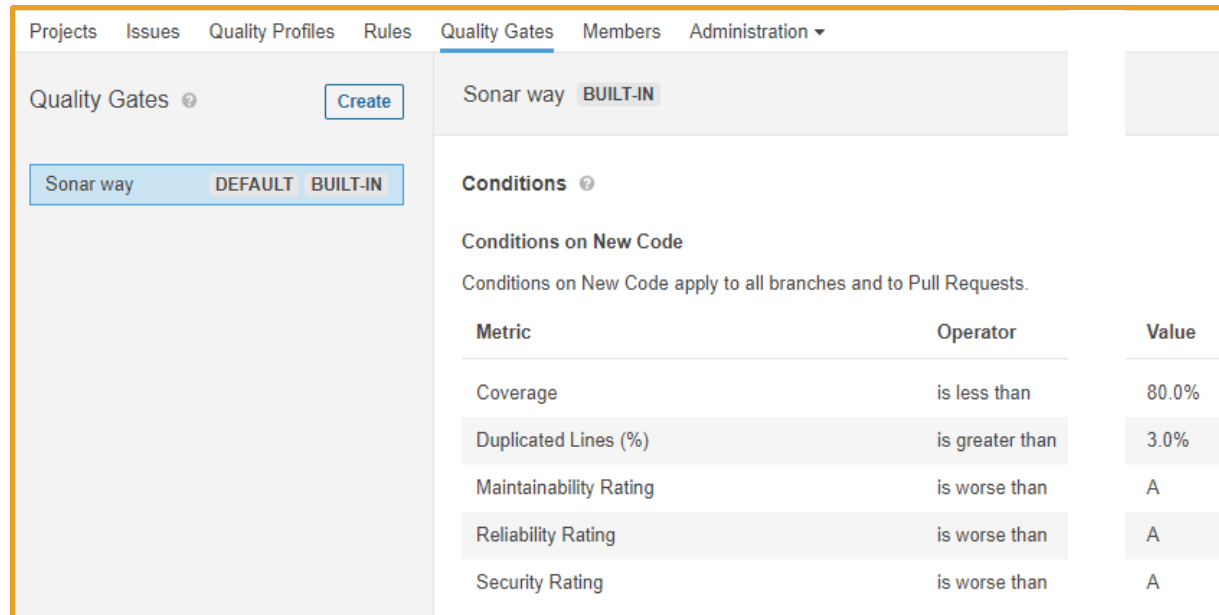
<https://sonarcloud.io/documentation/user-guide/quality-gates/>

A **Quality Gate** is the best way to [Fix the Water Leak](#) and enforce policies ensuring high quality code in your organization.

You can define as many quality gates as you wish. SonarCloud, by default, provides a built-in **Quality Gate** that is recommended for most projects. You can receive a notification when the **Quality Gate** fails.

To create a **Quality Gate**, define a set of Boolean conditions based on measure thresholds. Projects are then measured against them. For example:

- No new blocker issues
- Code coverage on new code greater than 80%



The screenshot displays the SonarCloud interface for managing Quality Gates. The top navigation bar includes links for Projects, Issues, Quality Profiles, Rules, Quality Gates (active), Members, and Administration. On the left, the 'Quality Gates' section shows a 'Create' button and tabs for 'Sonar way', 'DEFAULT', and 'BUILT-IN'. The main content area shows the 'Sonar way' gate, which is 'BUILT-IN'. Below this, the 'Conditions' section lists 'Conditions on New Code' that apply to all branches and Pull Requests. A table details the specific conditions:

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A

Clean as you Code Methodology

<https://www.sonarqube.org/features/clean-as-you-code/#:~:text=With%20the%20Clean%20as%20You,clean%20up%20after%20someone%20else.>

The act of sorting out new code smells, etc, each time you build the pipeline.

Monitoring Security and Vulnerability

<https://sonarcloud.io/documentation/user-guide/concepts/>

<https://sonarcloud.io/documentation/user-guide/metric-definitions/#security>

Security-related issues represent a place in your code that attackers could exploit.

Security hotspots are areas of the code that may cause security issues and therefore need to be reviewed.

The SonarCloud Quality Model has three different types of rules: Reliability (bug), Vulnerability (security), and Maintainability (code smell).

One of these rules will usually find and flag anything suspicious.

Then a human security auditor can manually review the report, delete the false positives, and send the appropriate issues for remediation.

