# Azure DevOps

## Tooling and Code Analysis

---

.NET

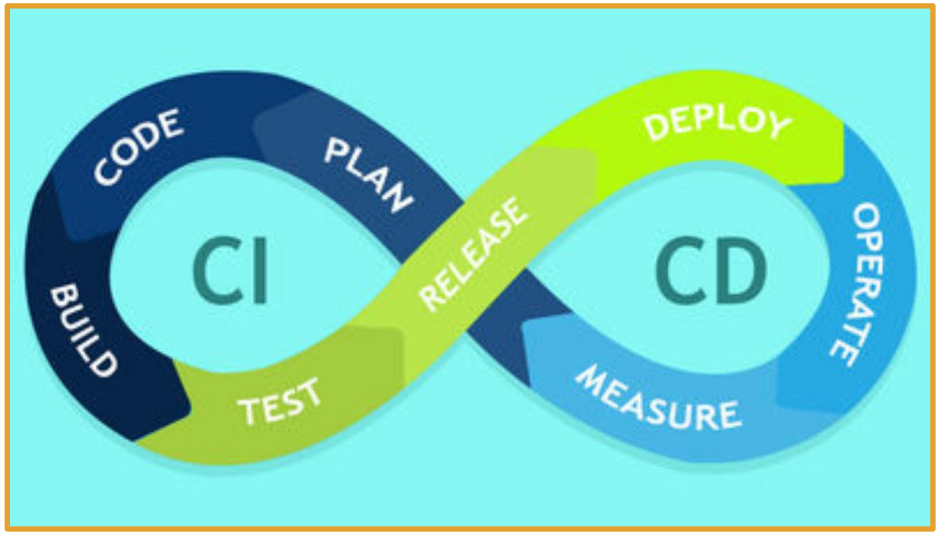
*DevOps technologies, combined with people and processes, enable teams to implement CI/CD and continually provide value to customers.*

https://azure.microsoft.com/en-us/solutions/devops/

# CI/CD and Continuous Testing (CT)

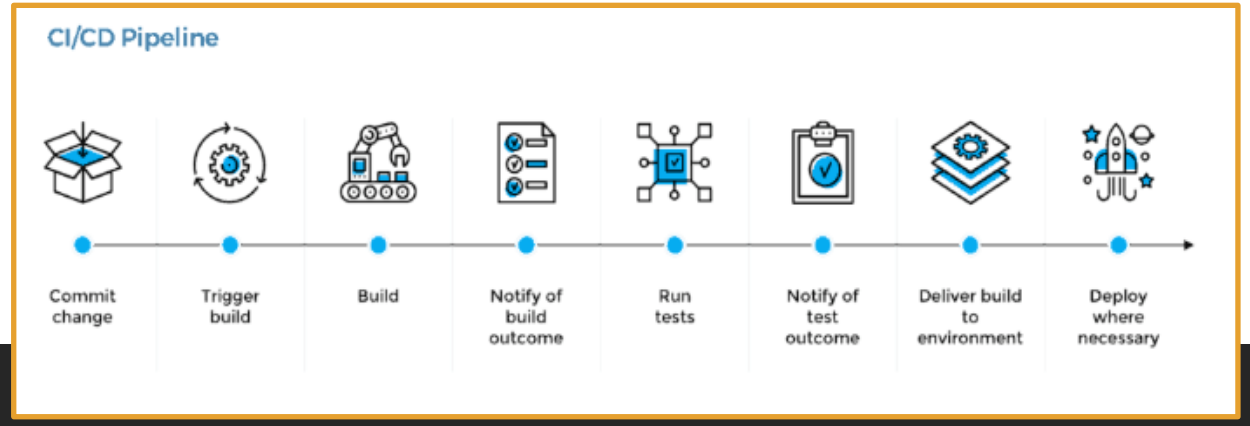
https://docs.microsoft.com/en-us/azure/devops/pipelines/overview?view=azure-devops-2019

***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 (Deployment) (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.



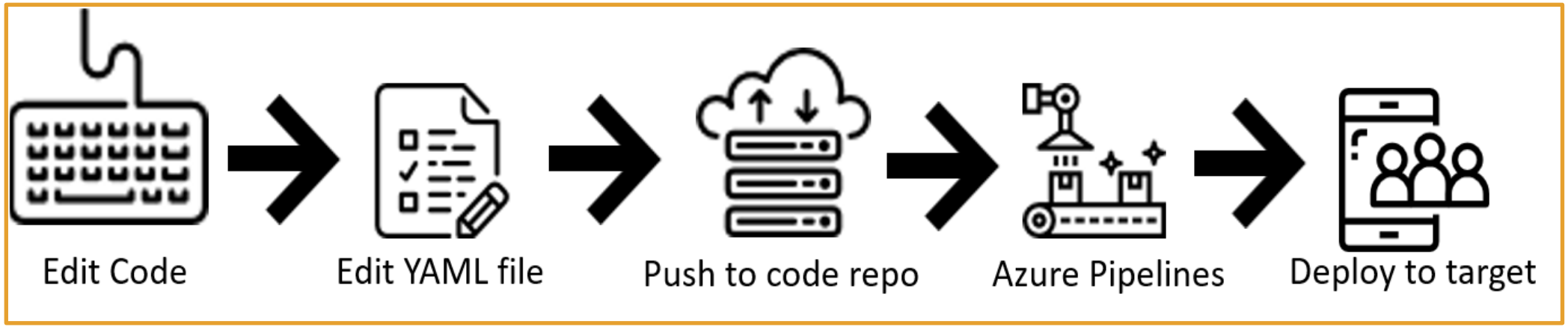
# Azure DevOps - Introduction

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.

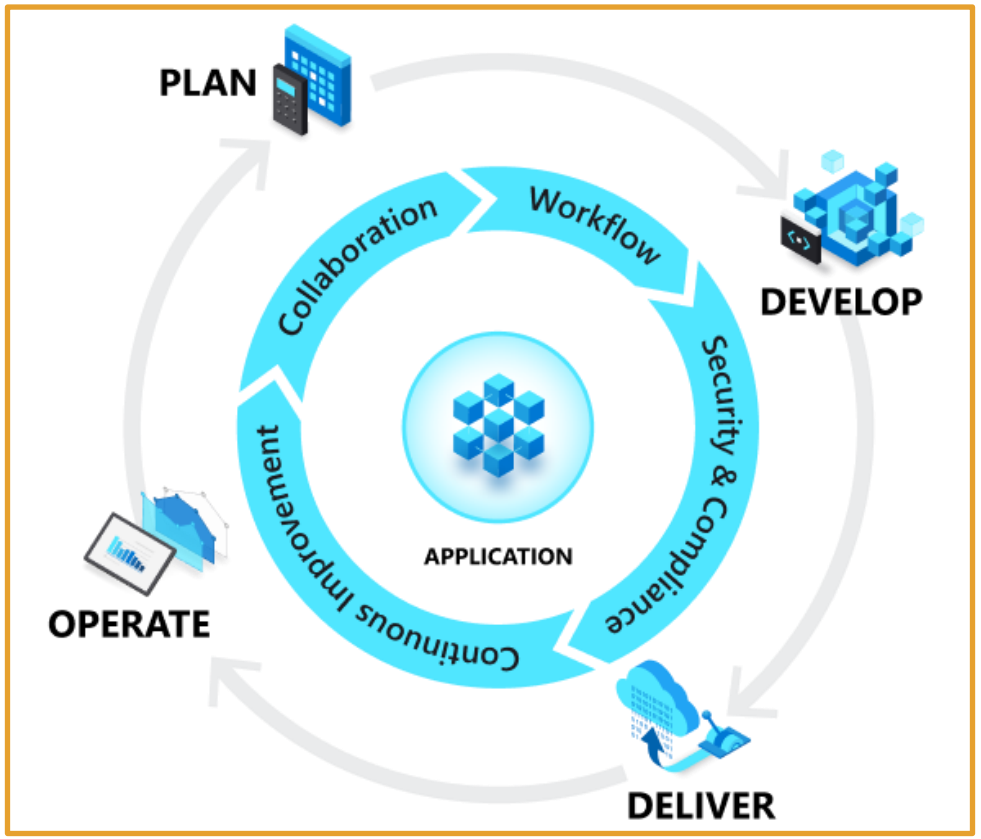


# Build Definition

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 .yml 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.

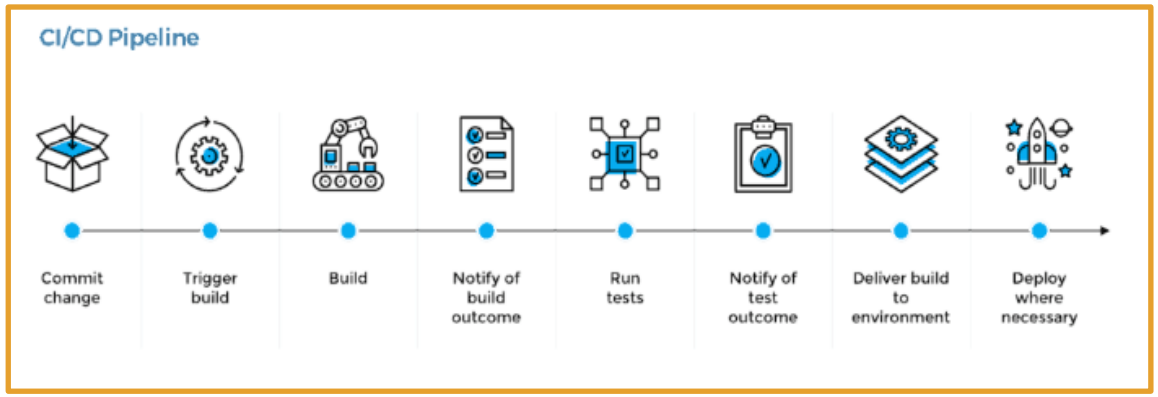


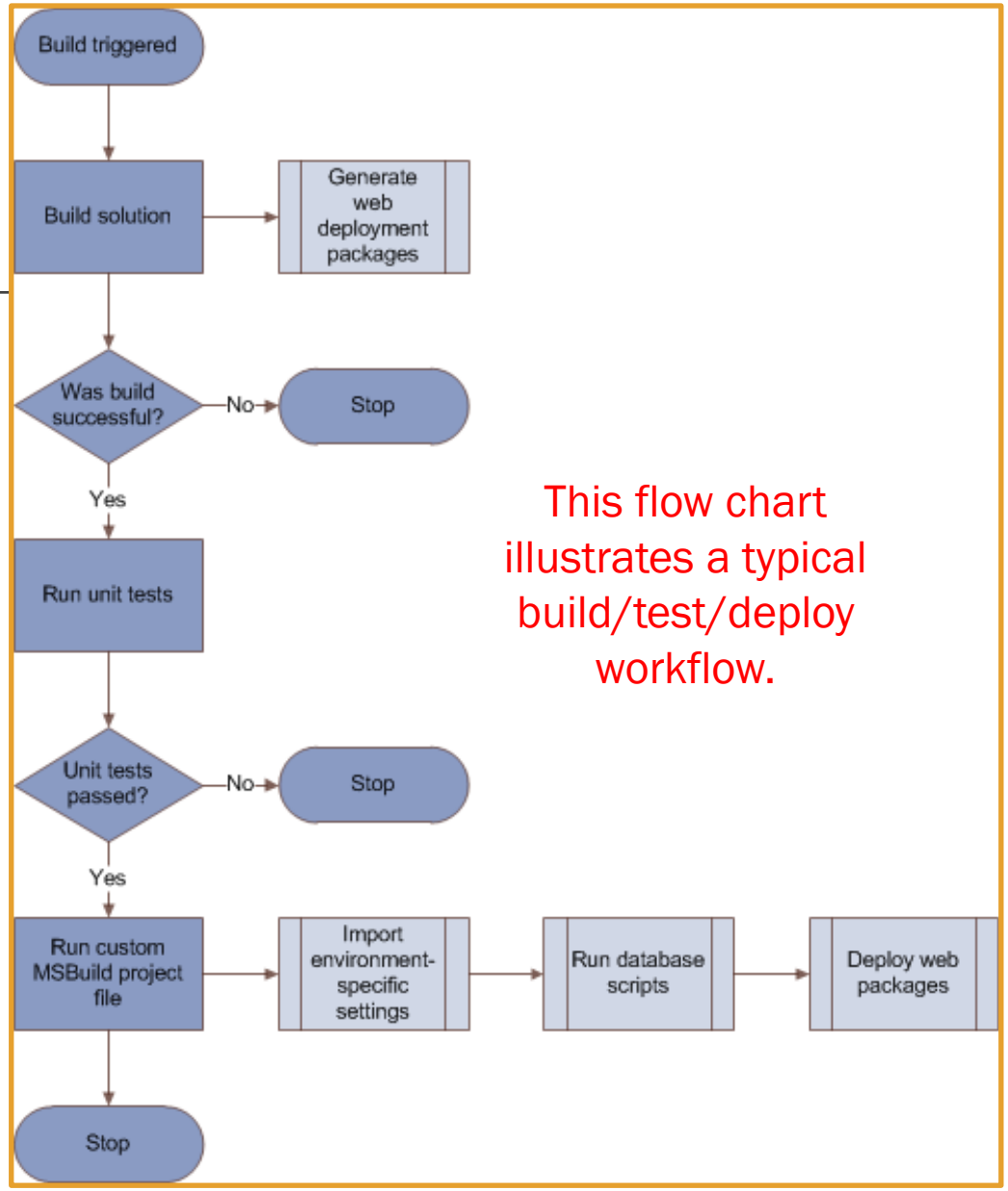
# Release Pipeline

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.



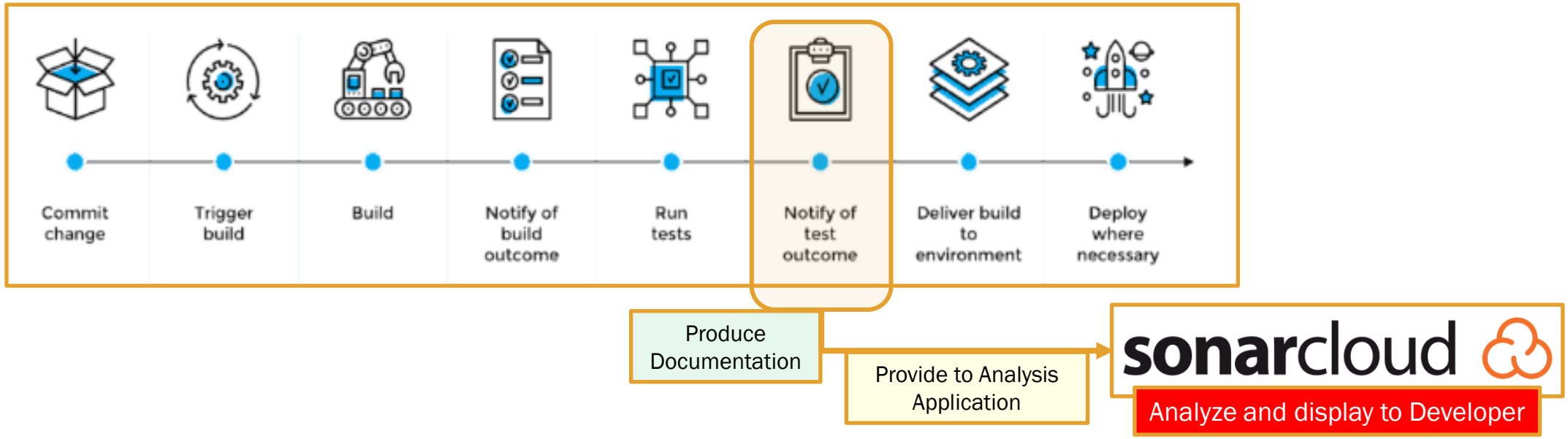




This flow chart illustrates a typical build/test/deploy workflow.

# What is Code Analysis?

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

Code analysis is performed on the code of an application as a tangent from the direct pipeline. For example, the pipeline will create documentation of how the unit and integration tests performed and make that documentation along with the base code available to another application. A 3rd party application or website like SonarCloud.io can then analyze the code and display useful information to the developer.



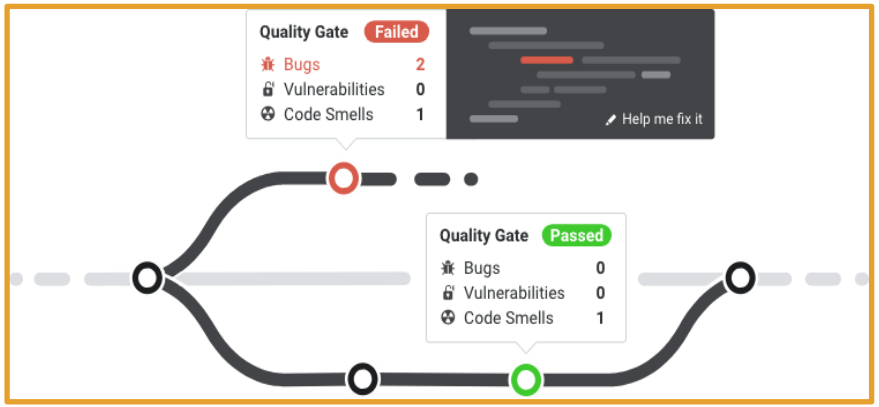
# What is Static Code Analysis?

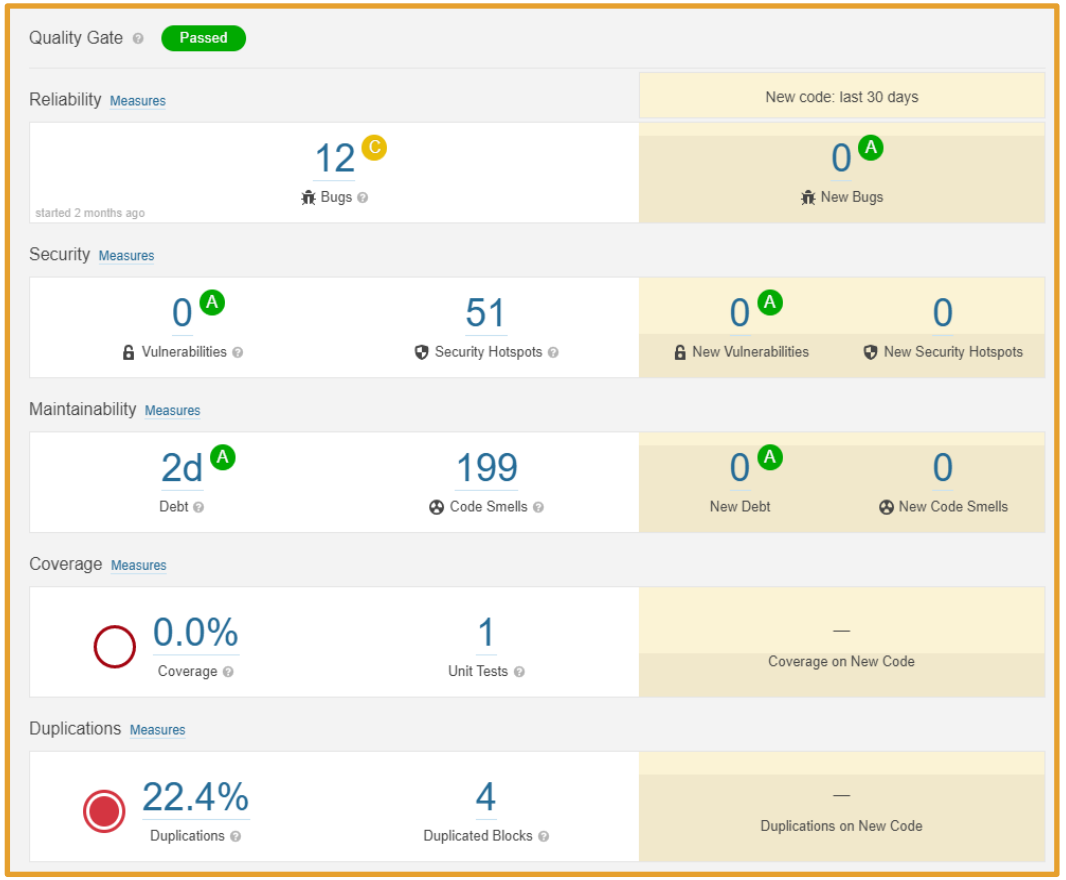
https://en.wikipedia.org/wiki/Static_program_analysis

***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***.



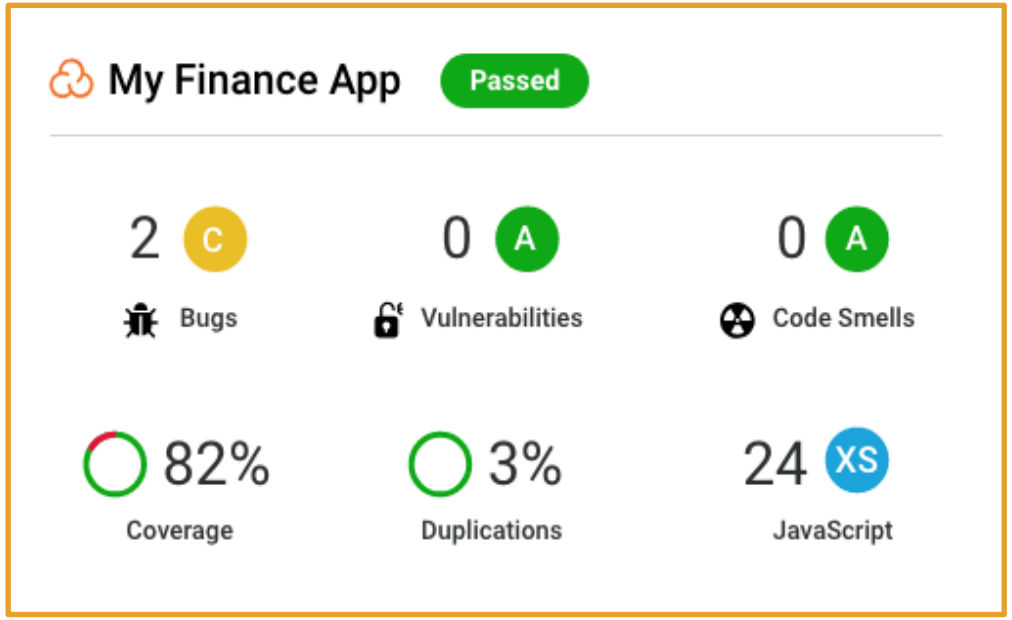
# What is a Coverage Review?

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.

Coverage = (CT + CF + LC) / (2*B + EL)

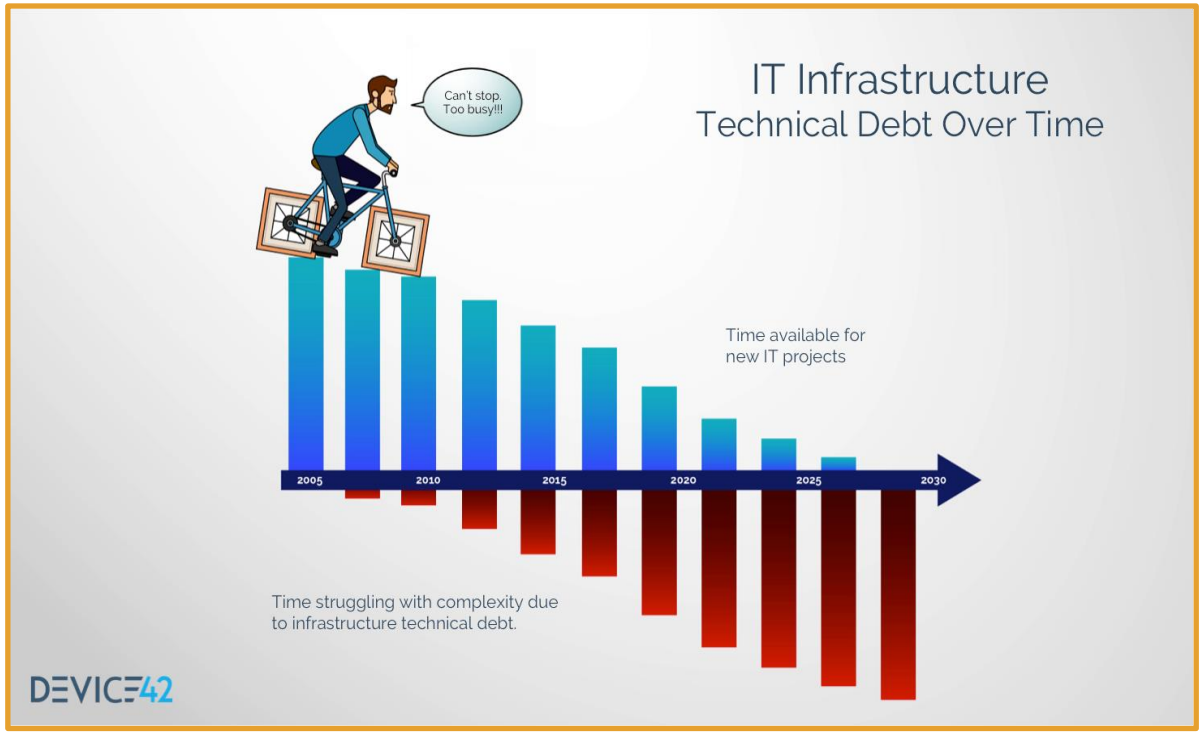
- CT = conditions that have been evaluated to 'true' at least once
- CF = conditions that have been evaluated to 'false' at least once
- LC = covered lines = lines_to_cover - uncovered_lines
- B = total number of conditions
- EL = total number of executable lines (lines_to_cover)



# 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

---

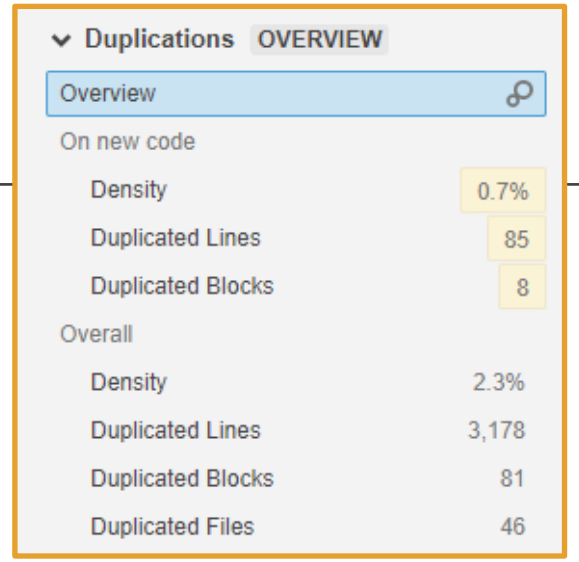
A ***Code Smell*** is any characteristic in the source code of a program that possibly indicates a deeper problem. Determining what is and is not a ***Code Smell*** is subjective, and varies by language, developer, and development methodology.

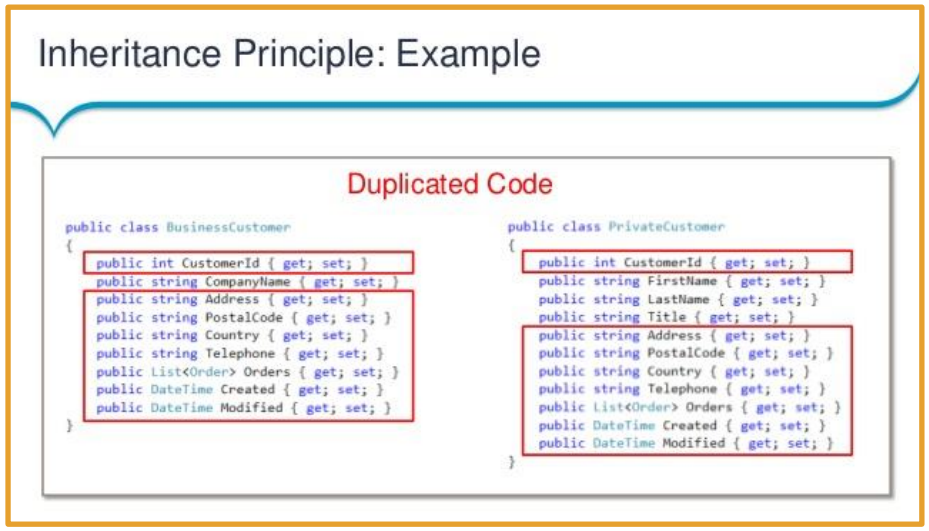
A ***Code Smell*** is an issue with long-term maintainability in the code. Leaving it as-is means that it will be more difficult for maintainers to make changes to the code. They'll risk introducing new errors as they make changes.

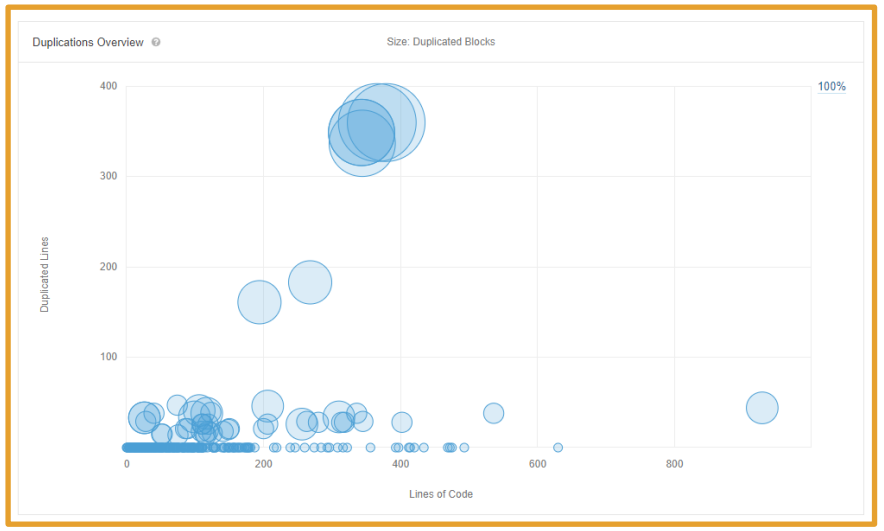
# Duplication

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.



# Quality Gate

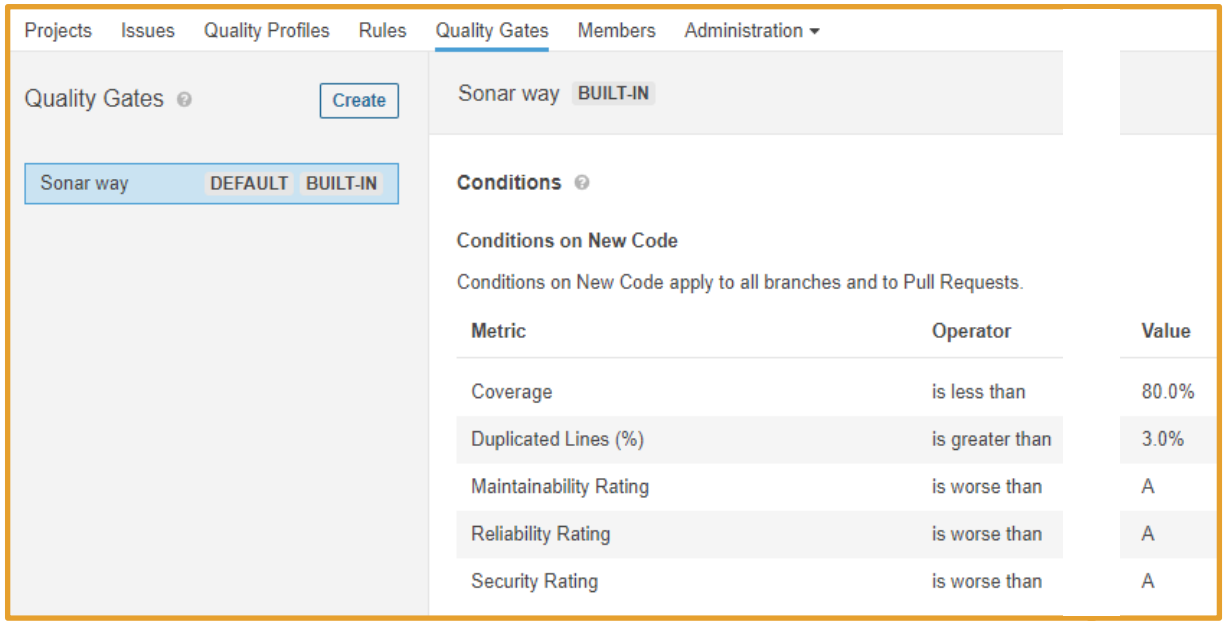
https://sonarcloud.io/documentation/user-guide/quality-gates/

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%



# Monitoring Security and Vulnerability

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

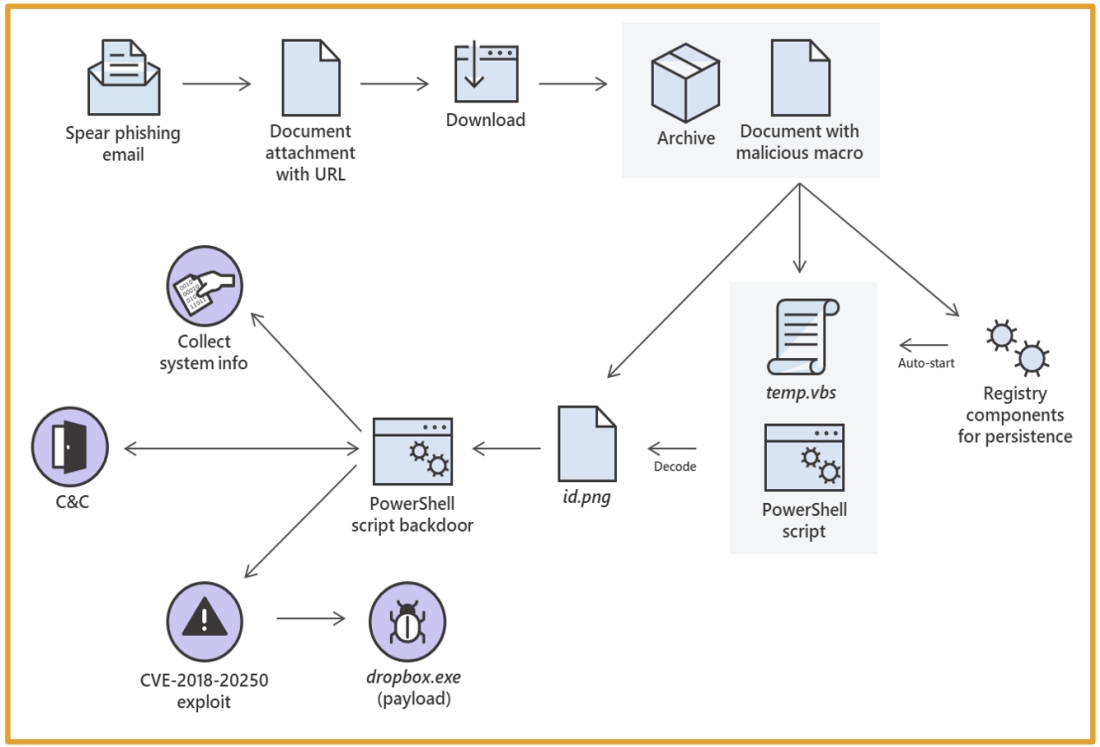
Security-related issues represent a place in your code that attackers could exploit.

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.



# 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.

# End of Presentation

Talks about servers and Kestrel which fills the gap between our code the the internet.

There is IIS (Internet Information Services) that is much more elaborate that windows uses. IIS is deactivated by default in windows. You can set it up correctly, but Nick will show it.

Go to Windows Features and switch on IIS. (Look at *.NET Core app to IIS*) . GO through the steps on the Docs page.

Nicks publishing with IIS in VS.  There is “publish” in VS but the meaning is extended.. More capable.

Ways to publish(after setting up IIS). Right click on startup project. .. Go under build-> publish.

There are a few options. Choose deploy to IIS. WebDeploy(means just copy the folder from A to B) He writes for server == localhost and gives a site name. He needed to launch VS in Admin mode.

SiteName Default WebSite

# The Easy VS way to publish

1. Open your mvc app of choice.
2. R click on starter proj and choose publish
3. Azure App Service →create new → create profile
4. (my VS is freezing….)
5. In App Service Screen give a name of your choice.
6. Fill in the blanks and hit enter
7. Azure creates the app service plan and app service.
8. When it’s finished the smaller window will disappear.
9. nick had a problem with publishing . You probably have to click publish to publish?

Now he’s using the Azure CLI to deploy to Azure App service. https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-dotnetcore-sqldb

https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest

We install Azure CLI to our machines.(this is for the VC Code people)

# Mark and Angie step by step deployment

1- upgrade or create an Azure Subscription account by putting in your phone number in.
2- Give the Subscription a name and choose the No Tech Support plan. Click upgrade. (wait a few minutes.)
3- Go back home click more services.
4- Click Resource Groups. Click Add
- - -choose your Subscription, resource group name, appropriate Region
- - - click Review + Create, Click Create (and wait)
4- Then find App service plan under more services. Click Add
5- Choose Subscription, Resource Group, Give it a name, select Windows. Select appropriate region
Change size of pricing tier, Click Dev/Test. F1 level Click Apply
Click review and create. Click create.
The above is a one-time thing,
Now go home and click App Service. Click Add.
Select correct Subscription, resource group
Give the website a name
Select Code.,
.NET Core 3.1(LTS)
Windows
Appropriate Region.
Verify the pricing tier.
Review and create. Create