DevOps Tooling and Code Analysis

.NET CORE

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

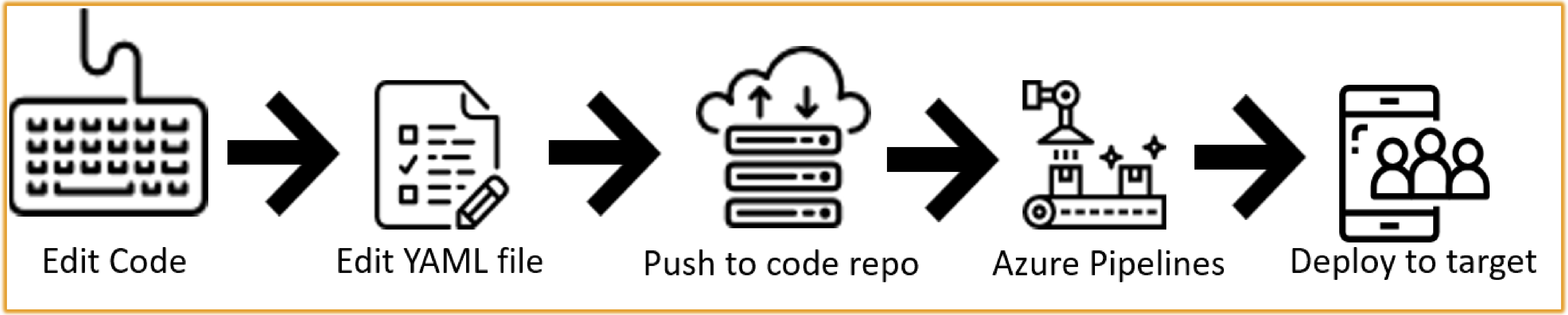
[HTTPS://AZURE.MICROSOFT.COM/EN-US/SOLUTIONS/DEVOPS/](https://azure.microsoft.com/en-us/solutions/devops/)

Tutorial –
Local to GitHub to Yaml to SonarCloud to Deloyment

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 project and make it available to other users. It works with just about any language or project type.

Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build your code and ship it to any target.



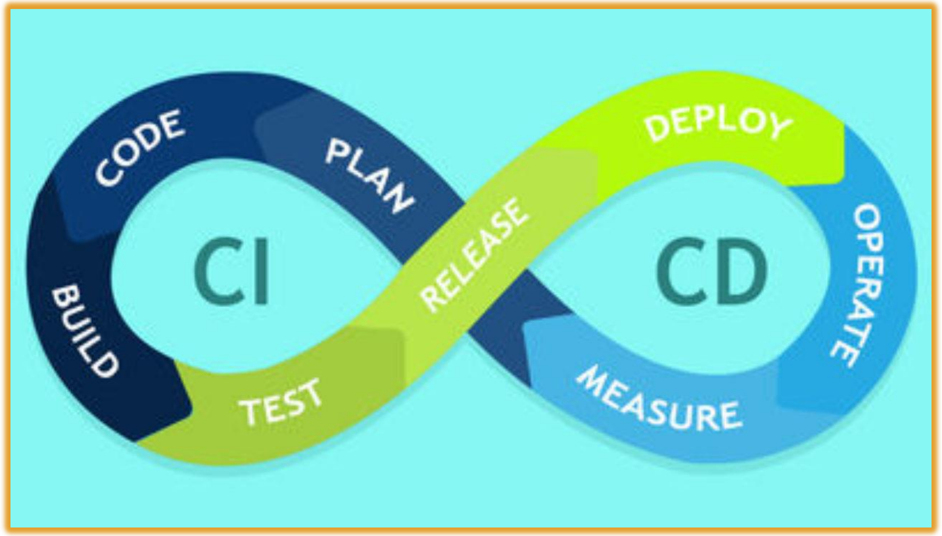
CI/CD and CT

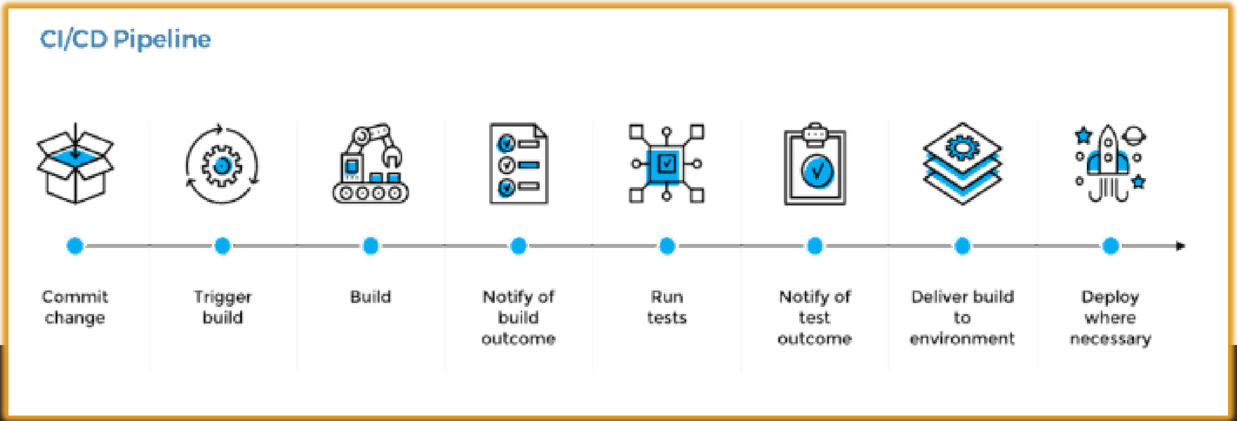
https://docs.microsoft.com/en-us/azure/devops/pipelines/overview?view=azure-devops-2019

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. Artifacts are produced from CI systems and fed to release processes to drive frequent deployments.

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. *CI* systems produce deployable artifacts including infrastructure and apps. Automated release processes consume these artifacts to release new versions and fixes.

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

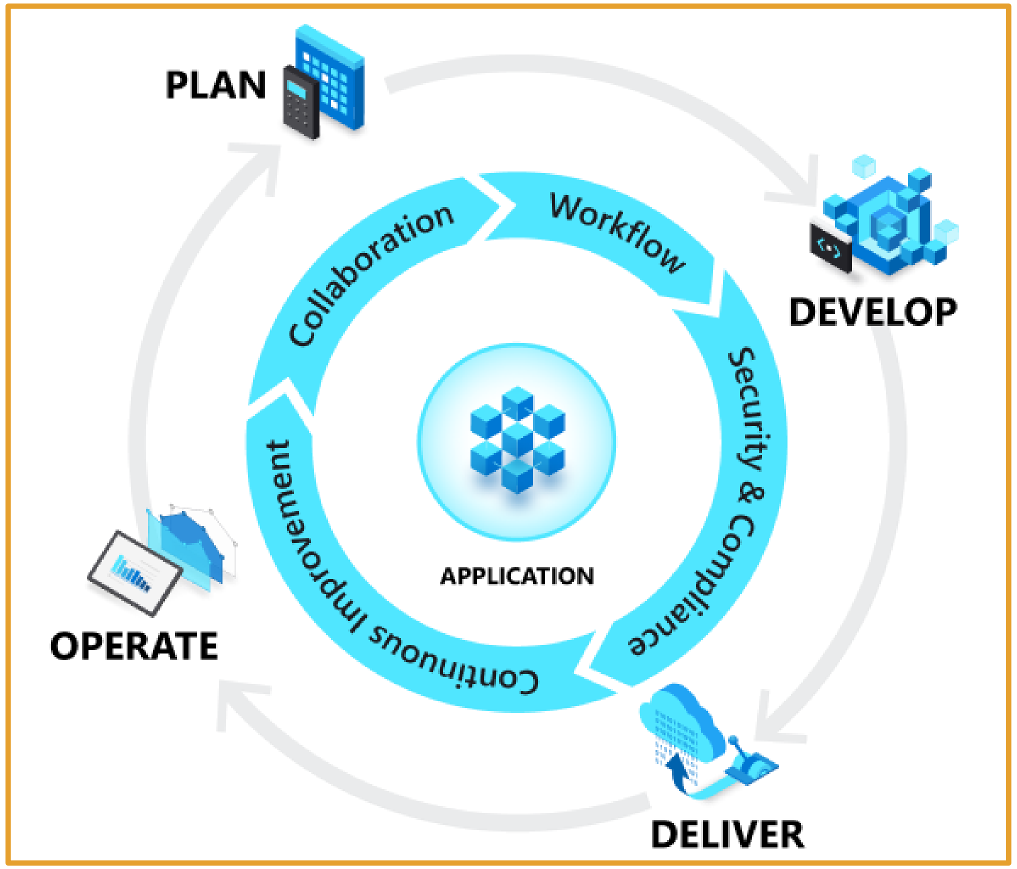




Build Definition

A ***build definition*** is the mechanism that controls how and when builds occur. Each build definition specifies:

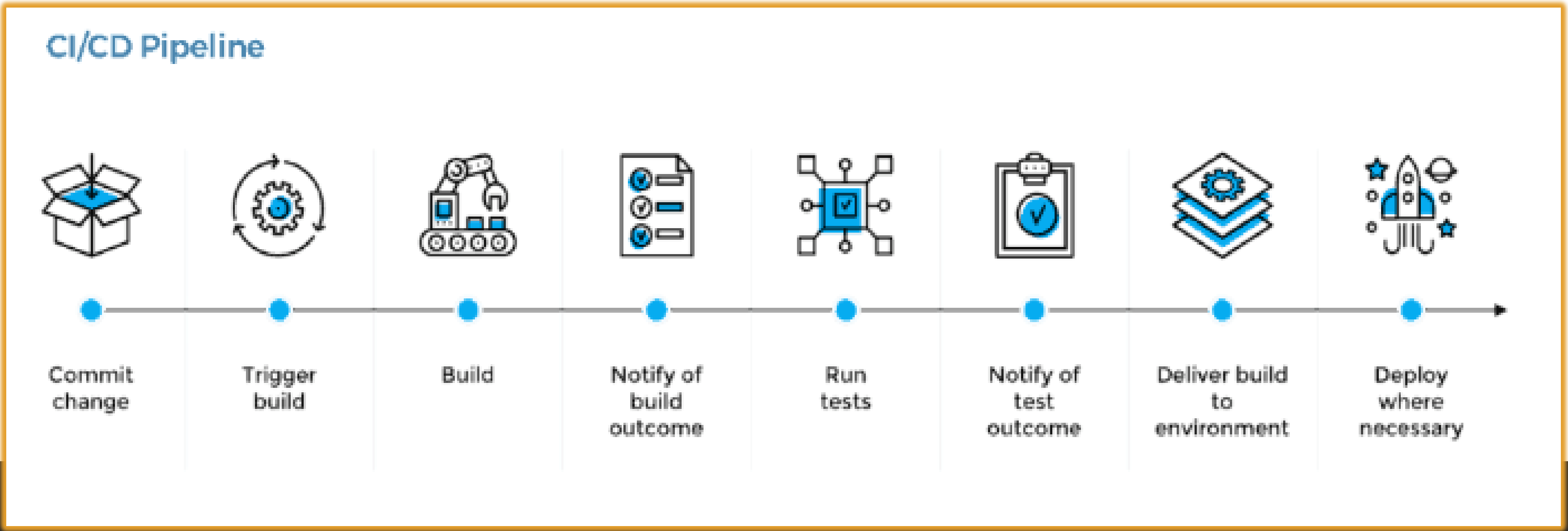
- The things you want to build.
- The criteria that determine when a build should take place, like manual triggers, continuous integration (CI), or gated check-ins.
- The location to which the Build should send build outputs, including deployment artifacts like web packages and database scripts.
- The 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 continuously deliver (CD) software to your customers at a faster pace and with lower risk. You can fully automate the testing and delivery of your software in multiple stages all the way to production or set up semi-automated processes with approvals and on-demand deployments.



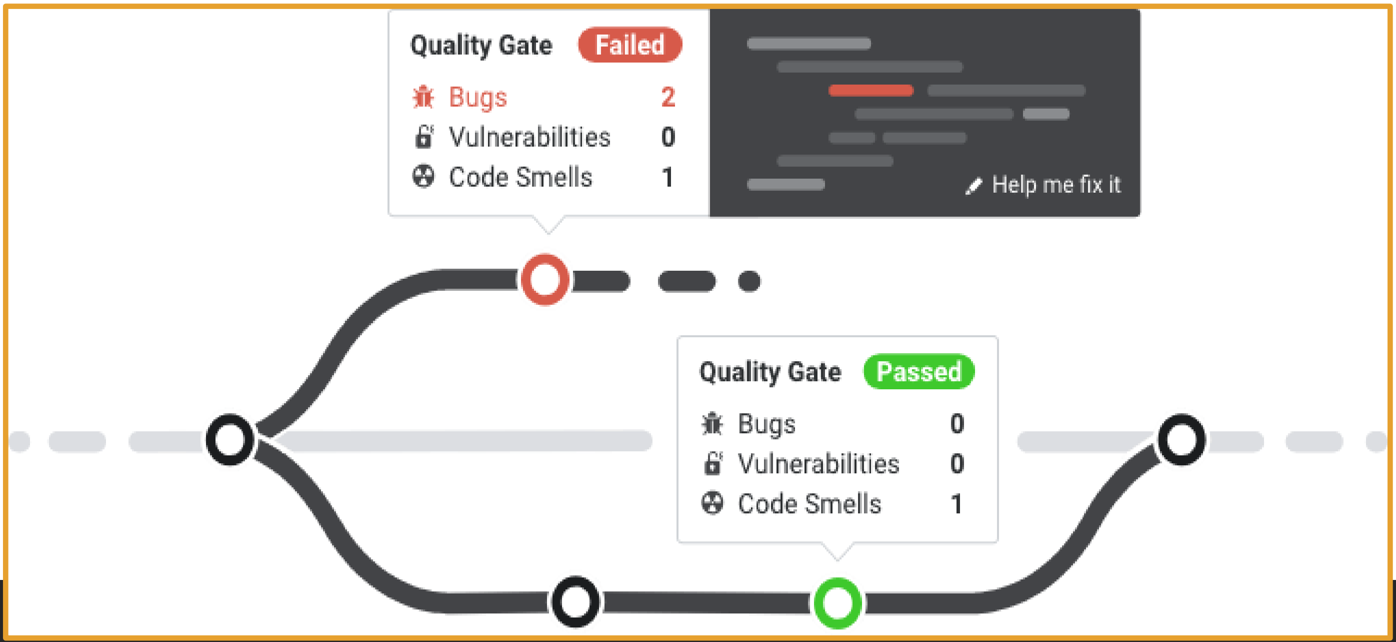
Static Code Analysis

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

Static code analysis is the analysis of computer software performed without actually executing programs. The analysis is usually performed on the source code.

The term is usually applied to the analysis performed by an automated tool (SonarCloud).

Human analysis is called code review.



Coverage Review

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

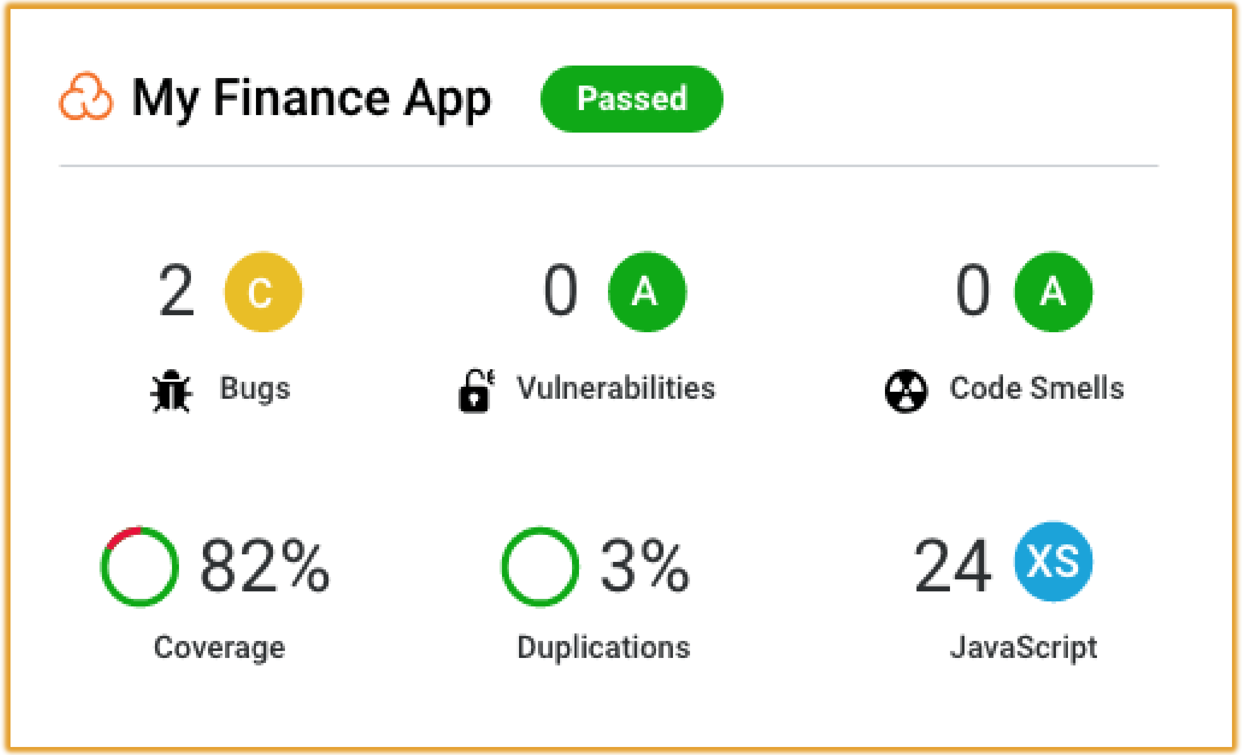
How much of the source code has been covered by the unit tests?

The answer to this question is:

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)

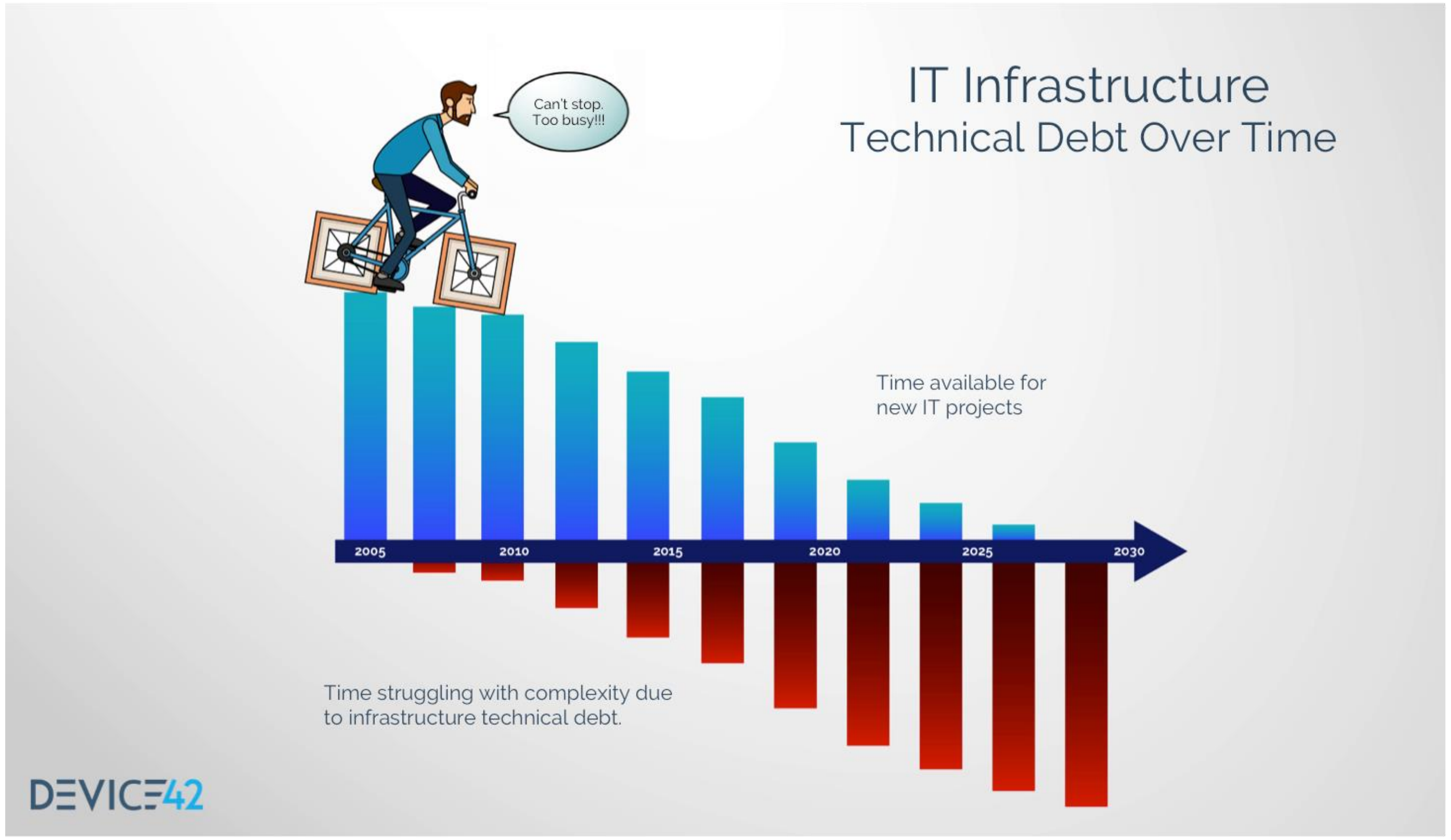
- CT = conditions that have been evaluated to 'true' at least once
- CF = conditions that have been evaluated to 'false' at least once
- LC = covered lines = lines_to_cover - uncovered_lines
- B = total number of conditions
- EL = total number of executable lines (lines_to_cover)



Technical Debt

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

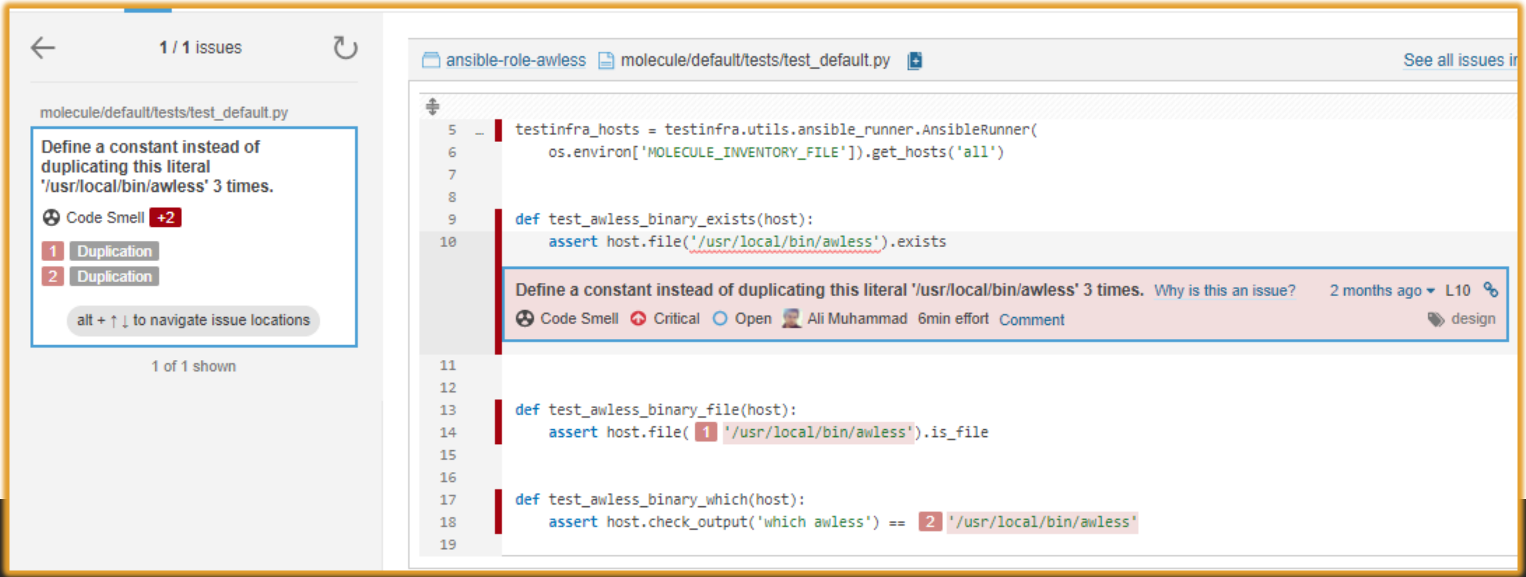
The estimated time required to fix all Maintainability Issues / code smells



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

In computer programming, 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 a maintainability-related issue in the code. Leaving it as-is means that at best maintainers will have a harder time than they should making changes to the code. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes.


1 / 1 issues
molecule/default/tests/test_default.py
ansible-role-awless
molecule/default/tests/test_default.py
See all issues i
1 of 1 shown
alt + ↑ ↓ to navigate issue locations

Define a constant instead of duplicating this literal '/usr/local/bin/awless' 3 times.

Code Smell +2

1 Duplication
2 Duplication

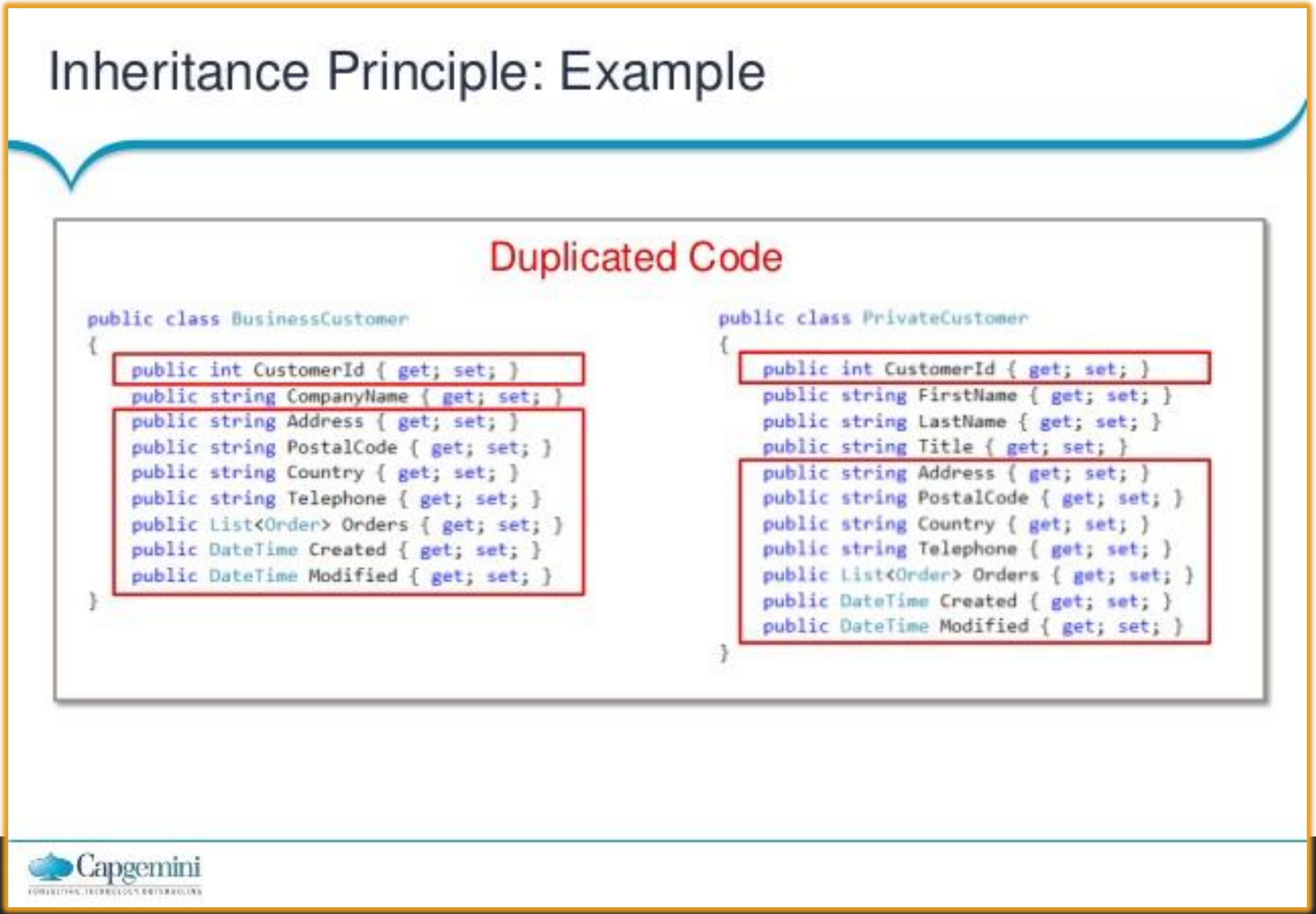
```
testinfra_hosts = testinfra.utils.ansible_runner.AnsibleRunner(
    os.environ['MOLECULE_INVENTORY_FILE']).get_hosts('all')


def test_awless_binary_exists(host):
    assert host.file('/usr/local/bin/awless').exists
```

Define a constant instead of duplicating this literal '/usr/local/bin/awless' 3 times. Why is this an issue? 2 months ago L10
Code Smell Critical Open Ali Muhammad 6min effort Comment design

```


def test_awless_binary_file(host):
    assert host.file( 1 '/usr/local/bin/awless').is_file


def test_awless_binary_which(host):
    assert host.check_output('which awless') == 2 '/usr/local/bin/awless'

```

Duplication

Lines of code that are identical and could theoretically be separated into a method to be called or resolved using SOLID or DRY principles.



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; }
}
```

Capgemini

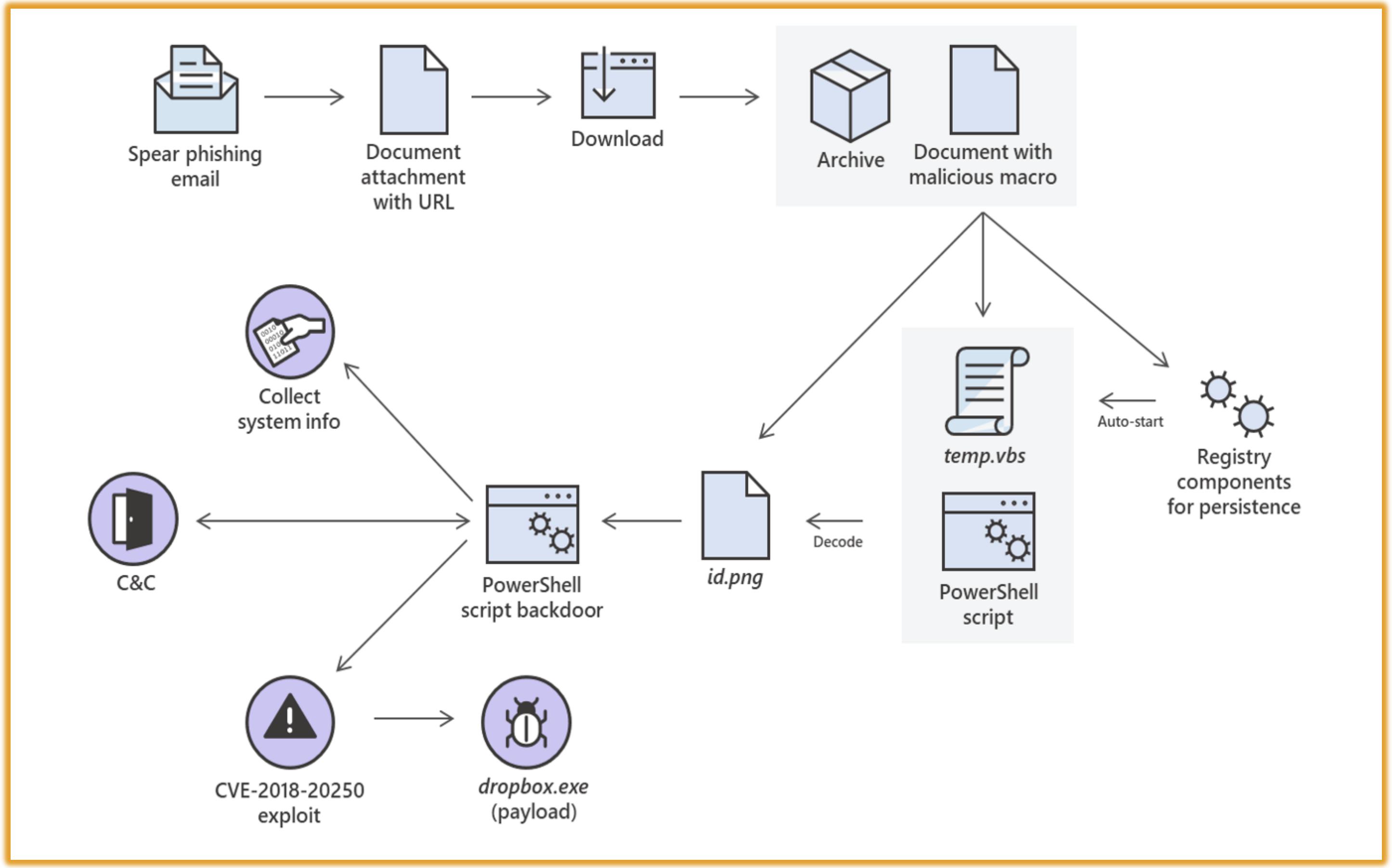
Security and Vulnerability

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

A security-related issue represents a backdoor for attackers.

No one wants bugs or vulnerabilities found on their source code to be unveiled to third-parties.

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



Spear phishing email
Document attachment with URL
Download
Archive
Document with malicious macro
Collect system info
temp.vbs
Auto-start
Registry components for persistence
C&C
PowerShell script backdoor
id.png
Decode
PowerShell script
CVE-2018-20250 exploit
dropbox.exe (payload)