Jet Propulsion Laboratory
California Institute of Technology

Raytheon

The Open Source Way to Standards Development: A NASA-JPL Approach to Software Excellence

Speakers: Rishi Verma[1], Kyongsik Yun[1], John Engelke[1,2]

[1]Jet Propulsion Laboratory, California Institute of Technology

[2]Raytheon Technologies

SCaLE 21x | Pasadena, CA MARCH 14-17, 2024

SLIM In a Nutshell.

Our Scope

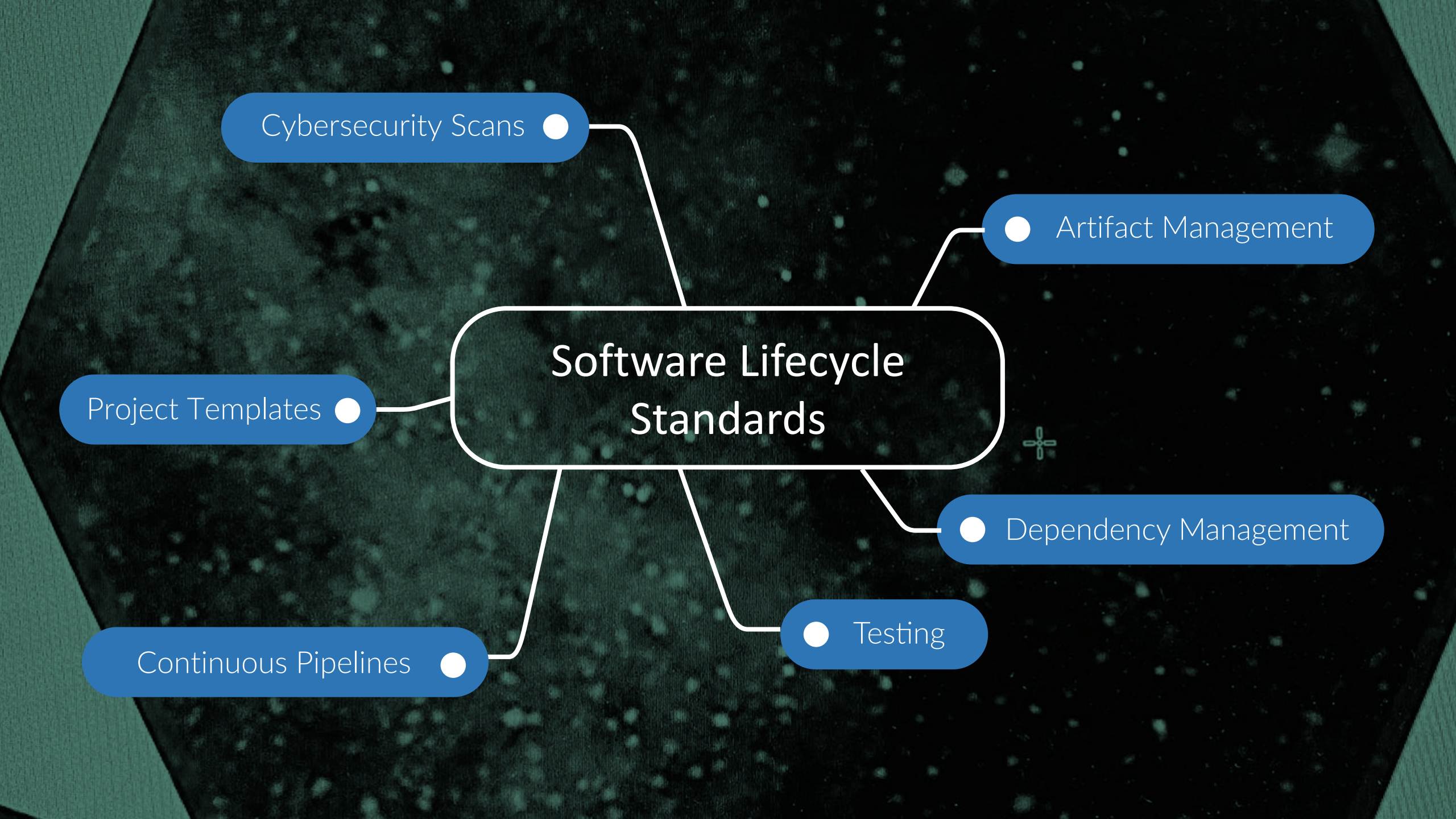
We focus on best practices related to **software project governance, documentation**, and **development life-cycles**.

Community Based

We solicit improvement ideas and solutions from **our community** deliver best practices back to our members.

Open Source

We treat best practices and **standards-as-code**. We iteratively improve our recommendations through the open source tickets and pull requests.


Software Lifecycle Standards
Cybersecurity Scans
Artifact Management
Project Templates
Dependency Management
Testing
Continuous Pipelines

Information Sharing Standards

- Training
- Documentation Standards
- Coding Standards
- User Feedback
- Software Publicity


Governance Standards
Decision Making
Triaging
Vetting Policies
Contribution Policies
Role Management
Task Management

Question

How can government software be infused with best practices that are:

- Consistently Implemented
- Have a low cost of adoption for projects
- Infused in a manner that easily scales to hundreds of projects

This is Why We Developed SLIM.

Consistent Implementation → *Strategy* →

Promote community-developed best practices using open reviews

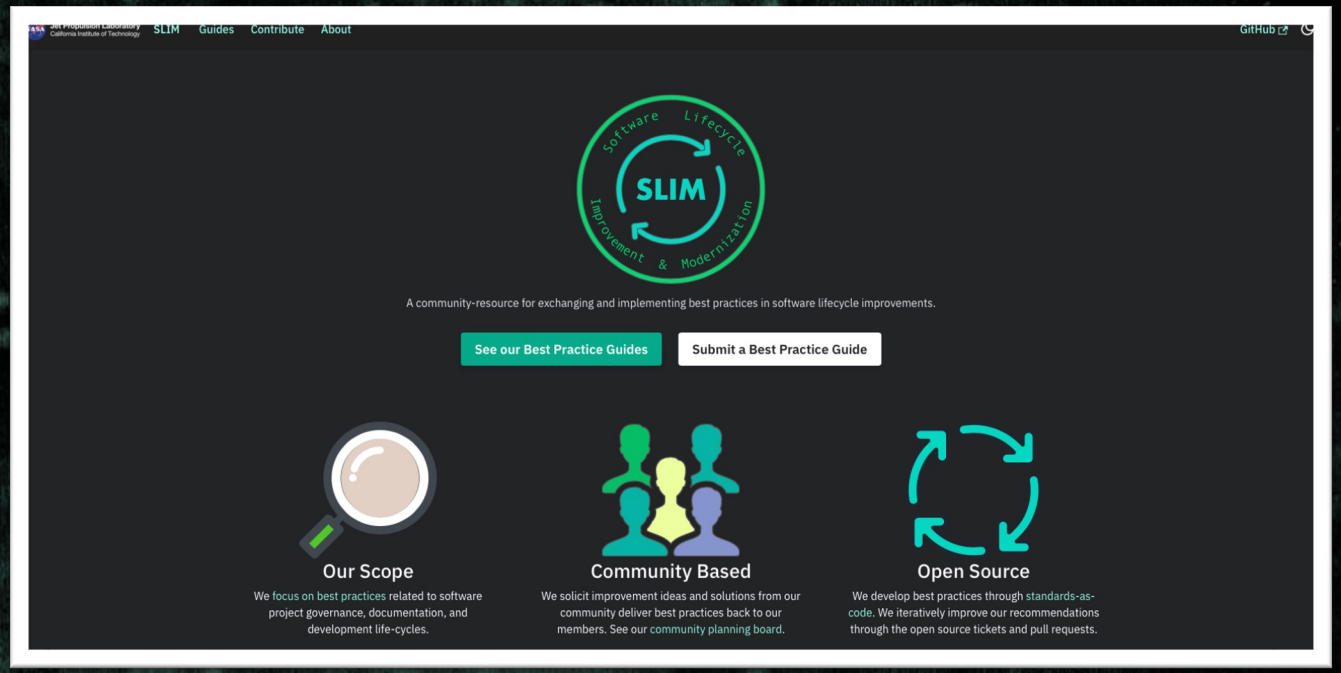
Low Cost of Adoption → *Strategy* →

Implement best practices as code that projects can patch at low-cost

Scale to hundreds of projects → *Strategy* →

Use automated pull-request and tickets to push out hundreds of best practices to projects at scale

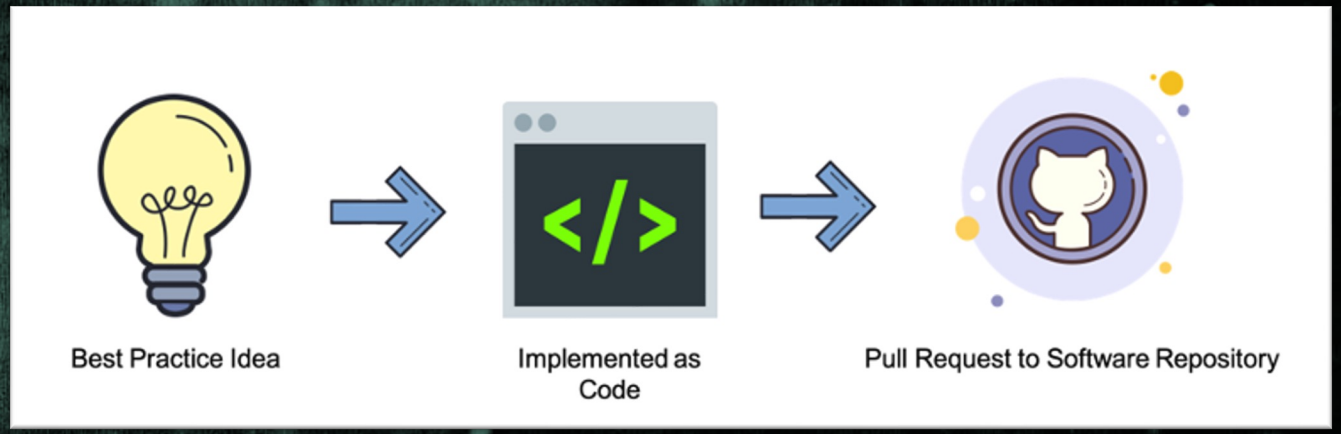
How We Deliver. How You Can Engage.



Through Our Website, We Provide:

- Downloadable best practice kits
- Submission system for best practice kits

https://nasa-ammos.github.io/slim



Through Automation, We:

- Disseminate best practice kits through pull requests, issue tickets directly on project repositories
- Scale to hundreds of repositories

Example: README Templates

Unity
SDS-as-a-Service

Unity Data Services

```
This repository contains source code that handles data ingest, data cata
```

Contributor Covenant 2.1

This repository contains

- Functions that parses metadata and is used within Cumulus workflows
- Functions that invoke Cumulus API and implement OGC DAPA and STAC
- Functions for data search and download through OGC DAPA and STAC
- Functions for data ingest through OGC DAPA, STAC, and CNM

Unity Documentation | Cumulus Documentation | Issue Tracker

Features

- Data ingest
- Data catalog (and metadata parsing)
- Data search
- Data access

Contents

- Quick Start
- Changelog
- FAQ

Example: Bug Ticket Templates

Issue: Bug Report Form

Report a bug to help us improve. If this doesn't look right, choose a different type.

Add a title

[Bug]:

Thanks for filing a bug ticket. We appreciate your time and effort. Please answer a few questions.

Checked for duplicates *

Have you checked for duplicate issue tickets?

Yes - I've already checked

Website or Best Practice Guide? *

Does the bug have to do with the SLIM website or a best practice guide?

Website

Describe the bug *

A clear and concise description of what the bug is. If the bug has to do with a best practice guide, be sure to mention which guide
snippets preferred but screenshots welcome.

When I did [...] action, I noticed [...]

What did you expect? *

A clear and concise description of what you expect to happen

I expected [...]

Reproducible steps

How would we reproduce this bug? Please walk us through it step by step. Plain-text snippets preferred but screenshots welcome

```
1.
2.
3.
...
```

Example: GitHub Security

General

Access
- Billing and plans
- Repository roles
- Member privileges
- Import/Export
- Moderation

Code, planning, and automation
- Repository
- Codespaces
- Copilot
- Actions
- Webhooks
- Discussions
- Packages
- Pages
- Projects

Security
- Authentication security
- Code security and analysis
- Verified and approved domains
- Secrets and variables

Third-party Access
- OAuth application policy
- GitHub Apps
- Personal access tokens (Beta)

Integrations
- Scheduled reminders

Archive
- Logs
- Deleted repositories

- Developer settings

Code security and analysis

Security and analysis features help keep your repositories secure and updated. By enabling these features, you're granting us permission to perform read-only analysis on your organization's repositories.

Private vulnerability reporting (Beta) — Disable all | Enable all
Allow your community to privately report potential security vulnerabilities to maintainers and repository owners. Learn more about private vulnerability reporting.
- [ ] Automatically enable for new public repositories

Dependency graph — Disable all | Enable all
Understand your dependencies.
- [x] Automatically enable for new private repositories

Dependabot
Keep your dependencies secure and up-to-date. Learn more about Dependabot.

Dependabot alerts — Disable all | Enable all
Receive alerts for vulnerabilities that affect your dependencies and manually generate Dependabot pull requests to resolve these vulnerabilities. Configure alert notifications.
- [x] Automatically enable for new repositories

Dependabot security updates — Disable all | Enable all
Enabling this option will result in Dependabot automatically opening pull requests to resolve every possible vulnerability. If you would like more specific configuration options leave this disabled and use Dependabot rules.
- [x] Automatically enable for new repositories

Code scanning — Disable all | Enable all
Identify vulnerabilities and errors with CodeQL. Default CodeQL analysis will be set up on eligible public repositories.
- [ ] Recommend the extended query suite for repositories enabling default setup
  The extended query includes the default suite, plus lower severity and precision queries.

Secret scanning — Disable all | Enable all
Receive alerts on GitHub for detected secrets, keys, or other tokens.
- [x] Automatically enable for new public repositories

Push protection — Disable all | Enable all
Block commits that contain supported secrets.
- [x] Automatically enable for repositories added to secret scanning
- [ ] Add a resource link in the CLI and web UI when a commit is blocked

Grant Dependabot access to private repositories

In order to update dependencies in your organization, Dependabot accesses dependencies in all public repositories and the selected private repositories. The contents of these repositories will be accessible to all users in your organization through Dependabot updates. Be sure that you want to share these repositories with everyone in your organization or consider using private registries instead.

Add private repositories

More examples: collection of software metrics, unit testing standards, governance approaches, and more.

How We Monitor Infusion.

Project	Repository	Issue Templates	PR Templates	Code of Conduct	Contributing Guide	LICENSE	README	Change Log	Link to Docs in README
nasa-ammos	VICAR	✅	✅	✅	✅	✅	❌	❌	✅
nasa-ammos	aerie	✅	🅿️	✅	🅿️	✅	🅿️	❌	❌
nasa-ammos	anms-docs	🅿️	🅿️	✅	✅	✅	❌	❌	❌
nasa-ammos	3DTilesRendererJS	✅	❌	✅	❌	✅	❌	❌	✅
nasa-ammos	MMGIS	✅	✅	✅	✅	✅	❌	❌	✅
nasa-ammos	aerie-mission-model-template	🅿️	🅿️	🅿️	🅿️	❌	❌	❌	✅
nasa-ammos	aerie-simple-model-telecom	❌	❌	❌	❌	✅	❌	❌	❌
nasa-ammos	aerie-gateway	🅿️	🅿️	✅	🅿️	✅	❌	❌	❌
nasa-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Leaderboard table rows have been randomized for effect

Impact.

No. of Best Practices Infused into Projects

556

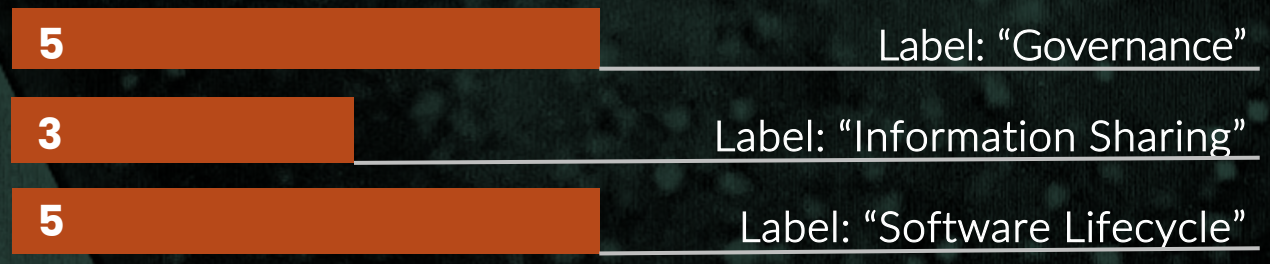
No. of Best Practices Proposed to Projects

768

Status of Standards

13	9	56
Specified	Developing	Backlogged

Types of Standards (Closed Tickets)



Number of Repositories Involved

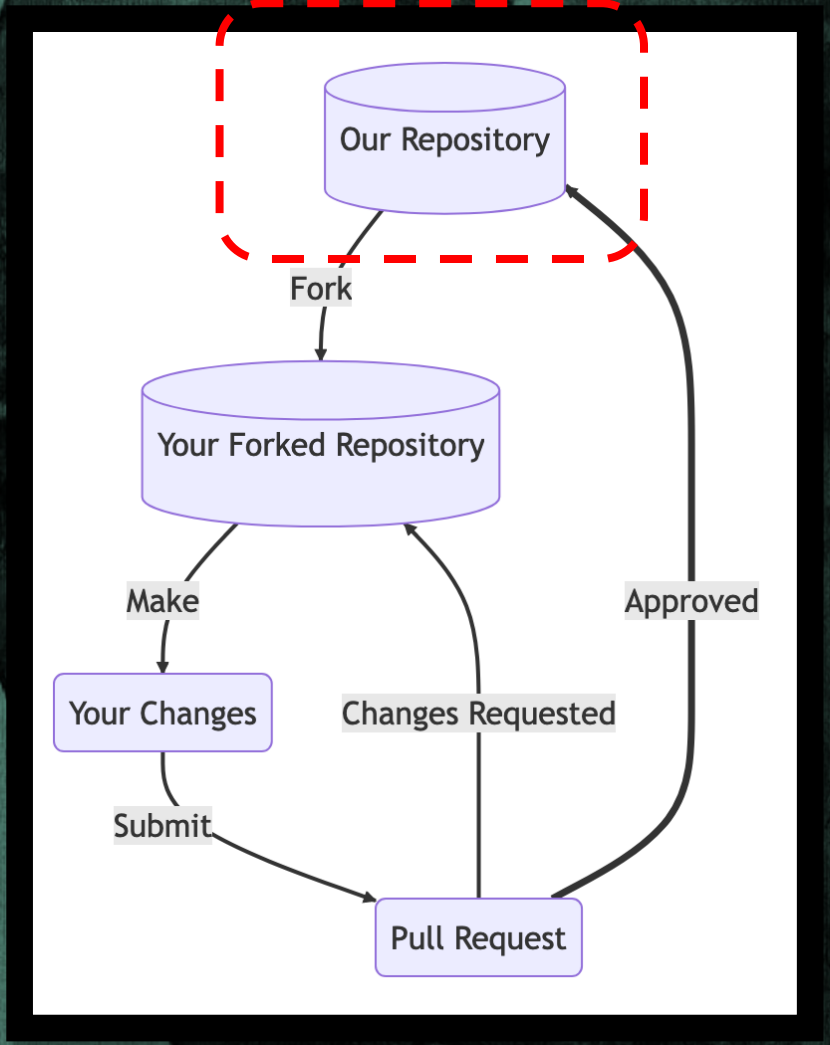
185

Contributing to SLIM: Continuous Testing Best Practice

Presenter: Kyongsik Yun

Contributing a Continuous Testing Best Practice.

Overview of Our Open Source Standards Development Process



1. Identify a Best Practice Need

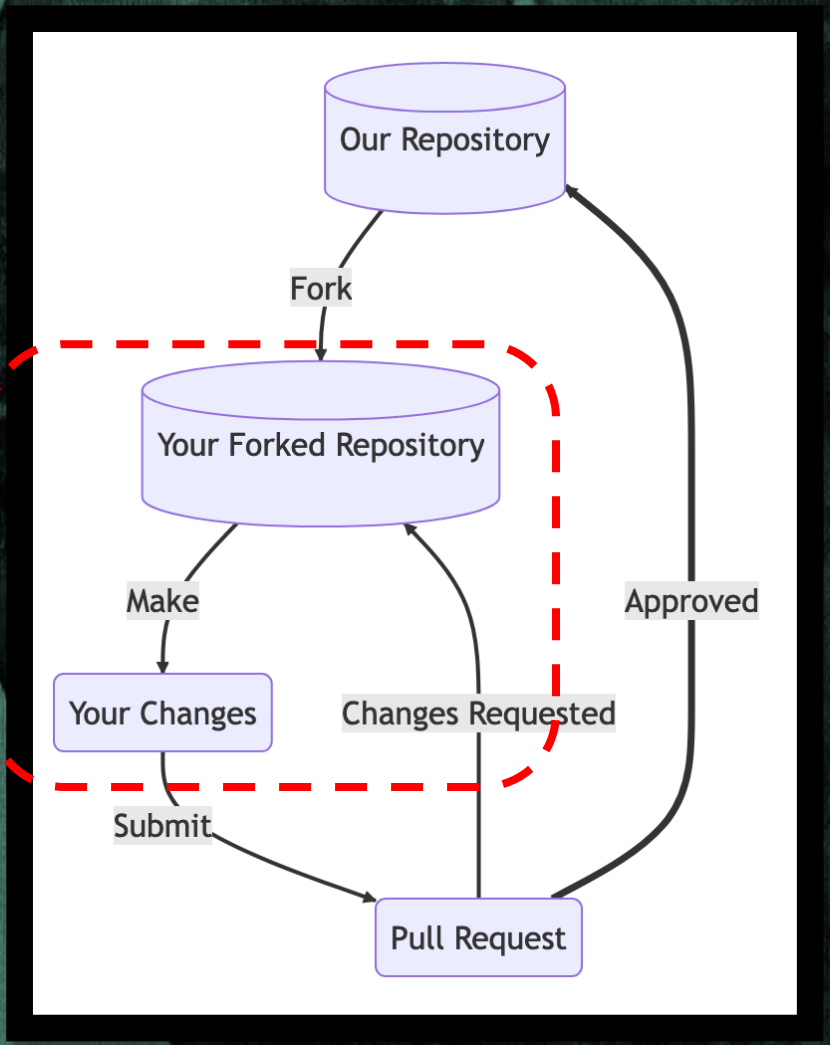
2.

- We identified a community, multi-project need for a best practice standard in continuous testing was identified.
- A new issue ticket was created to scope out the best practice needs and potential solution: https://github.com/NASA-AMMOS/slim/issues

3. Engage and Get Community Input

Contributing a Continuous Testing Best Practice.

Overview of Our Open Source Standards Development Process



1

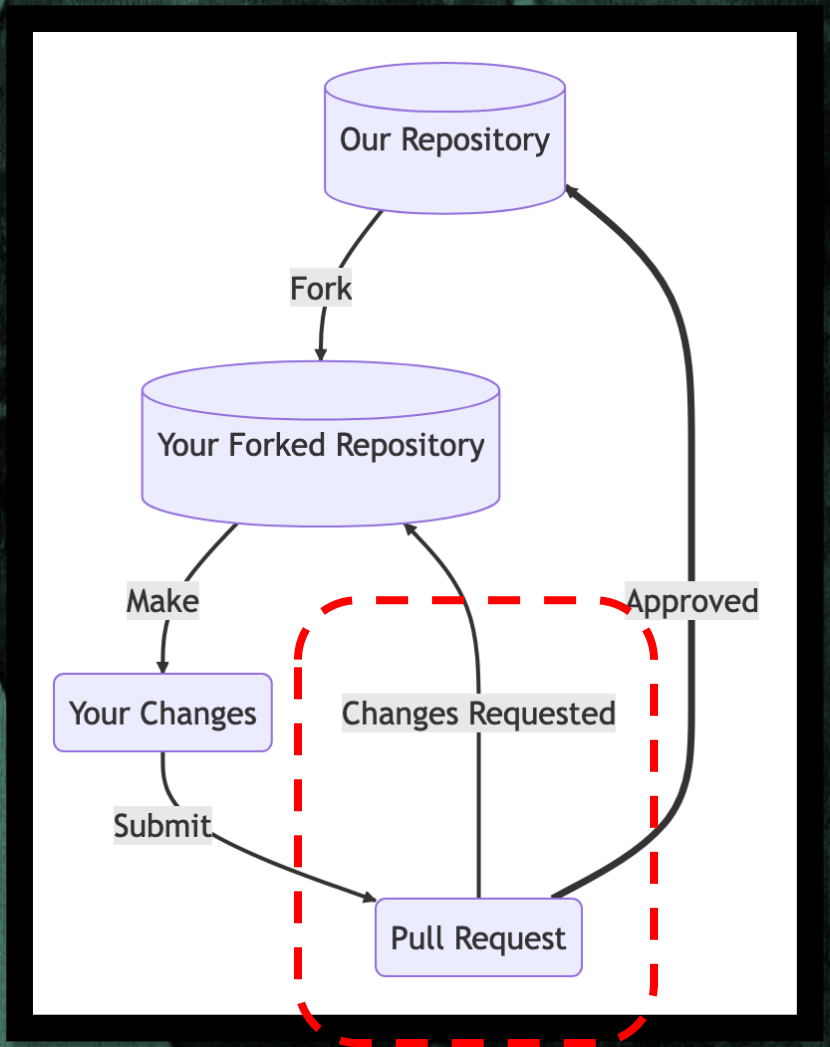
- Forked the SLIM repository: https://yunks128.github.io/slim/docs/guides/software-lifecycle/continuous-testing/
- Made proposed contributions to aid in continuous testing best practices, such as continuous testing templates, how to auto-generate test codes, continuous testing process automation

2. Develop Best Practice Standard

3. Engage and Get Community Input

Contributing a Continuous Testing Best Practice.

Overview of Our Open Source Standards Development Process



1. Identify a Best Practice Need

- Solicited community input to create a pull request for community input: https://github.com/NASA-AMMOS/slim/pull/144

2. Deve Practice Standard

3. Engage and Get Community Input

In-Depth | Continuous Testing Recs.

Continuous testing plan template

TESTING.md (Template) | SLI

yunks128.github.io/slim/docs/guides/software-lifec...

Jet Propulsion Laboratory
California Institute of Technology
SLIM

Software Lifecycle > Continuous Testing > TESTING.md (Template)

TESTING.md (Template)

```
# [INSERT PROJECT NAME HERE] Testing

## Introduction
This document provides an overview of the testing architecture for [INSERT PROJEC

---

## Types of Testing

The below list of tests are included in our testing setup. Further details are pr

- [ ] Unit Tests
- [ ] System Tests
  - [ ] Integration Tests
  - [ ] Security Tests
  - [ ] Performance Tests
  - [ ] User Interfaces Tests

### Unit Tests

Our unit tests ensure code is tested at a function, method, or sub-module level.

View existing to add new tests to:

**Location(s):**
- [INSERT PATH TO UNIT TEST FOLDER ON REVISION CONTROL]
- [INSERT PATH TO UNIT TEST FOLDER ON REVISION CONTROL]
- ...

View or modify the testing schedule per:

**Testing Frequency:**
- [INSERT TRIGGER OF WHAT KICKS OFF YOUR TESTS, E.G. CODE CHANGES, COMMITS, ETC.]
- [INSERT TIMING OF WHEN YOUR TESTS KICK OFF, E.G. NIGHTLY, EVERY WEEK, ETC.]

#### Contributing Unit Tests

To contribute unit tests, we recommend:
- Leveraging the [INSERT YOUR UNIT TESTING FRAMEWORK OF CHOICE] framework
```

Automated test code generation (LLM + Robot Framework)

Continuous Testing | SLIM

yunks128.github.io/slim/docs/guides/software-lifec...

2.2 Robot Framework and LLM Synergy

In scenarios where you are already well-versed in **Robot Framework**, leveraging the synergy between Robot Framework and **LLM (Llama2)** can yield significant benefits. Specifically, using LLM to auto-generate Robot Framework pseudocode streamlines the process of creating integration test cases. Here's an example:

1. **Generating Robot Framework Pseudocode with LLM**:
   - Use Llama2 to generate test case pseudocode in Robot Framework syntax.

```
*** Settings ***
Documentation    Example test suite
Library          SeleniumLibrary
*** Test Cases ***
Valid Login
    Open Browser    https://dummy-website.com    chrome
    Input Text      username_field    valid_username
    Input Text      password_field    valid_password
    Click Button    login_button
    Page Should Contain    Welcome, User!

Invalid Login
    Open Browser    https://dummy-website.com    chrome
    Input Text      username_field    invalid_username
    Input Text      password_field    invalid_password
    Click Button    login_button
    Page Should Contain    Invalid credentials
```

2. **Direct Revision and Enhancement**:
   - Revise the Robot Framework pseudocode as needed:
     - Add additional steps.
     - Include assertions for edge cases.
     - Incorporate custom keywords or libraries.
3. **Test Execution**:
   - Run the tests locally or integrate them into your CI pipeline.

By combining LLM's natural language capabilities with Robot Framework's structured format, you can efficiently create and adapt test cases.

2.2.1 Example - Robot Framework and LLM Synergy

Continuous testing process automation (pre-commit)

Continuous Testing | SLIM

yunks128.github.io/slim/docs/guides/software-lifec...

3. Automate Your Tests

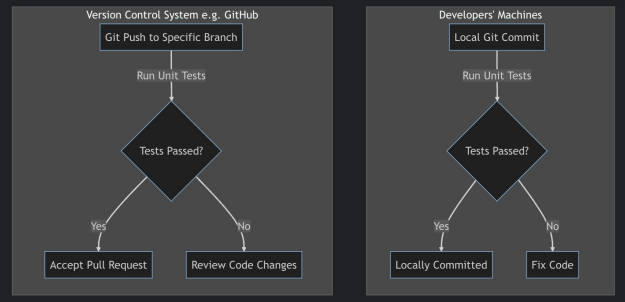
Our recommendation is to automate as many of your tests as possible. For tests that can't be automated, we suggest scheduling specific times for personnel to run manual tests.

3.1 Unit Test Automation

Please consult our Testing Frameworks guide for a choice of unit testing tools we recommend. Once selected, we recommend automating the execution of your unit tests in both of the following ways:

1. Execute unit tests locally on your developers' machines upon local Git commits
2. Execute unit tests upon Git pushes to given Git branches on your version control system (VCS) - hosted on GitHub.com or alternate

This idea is represented in the following diagram:



To make the above automation a reality, we recommend using pre-commit, a framework that manages and maintains multi-language pre-commit hooks that can be used on the client side as well as the server (VCS) side. Here's how to set it up:

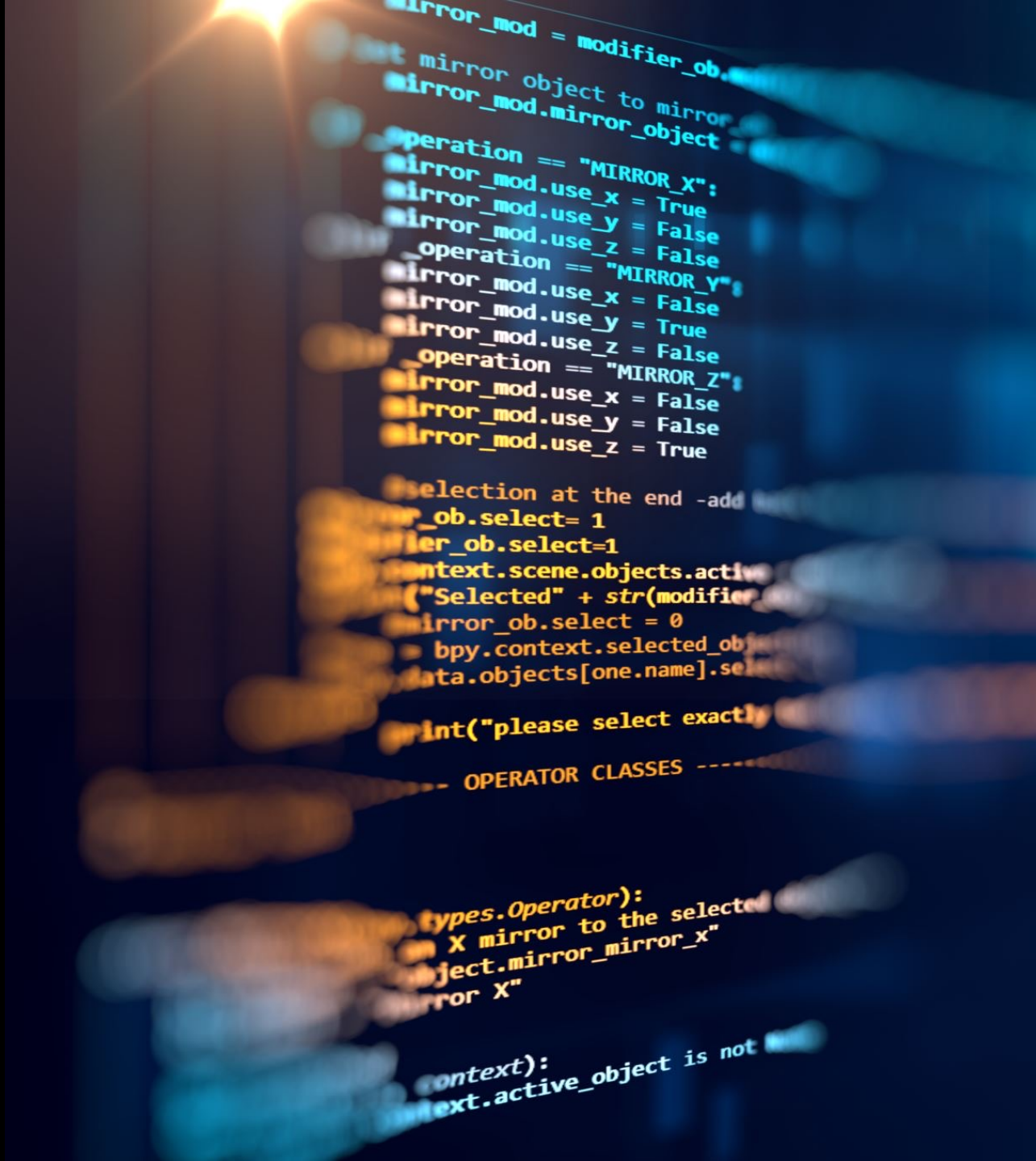
Developers' Machines

- **Step 1:** Install `pre-commit` on your local machine. If you are using Python, you can install it via pip:

```
pip install pre-commit
```

In-Depth | Automation in Continuous Testing

- Writing unit tests can be **time-consuming and tedious**, especially when testing large software applications with numerous components.
- Can we generate a test script automatically from the source code?



In-Depth | Writing Tests With LLMs

Tests show nearly 80%+ test coverage generation

Table 1: Compilation status of the generated unit tests

	LLM	% Compilable	% Compilable (after fix)	#Test Methods	#Test Classes
HumanEval	**GPT-3.5-Turbo**	43.1%	81.3%	1,117	130
	StarCoder	70.0%	76.9%	948	123
	Codex (2K)	37.5%	100%	697	160
	Codex (4K)	44.4%	99.4%	774	159
	Evosuite	100%	NA	928	160
	Manual	100%	NA	1,303	160

Codex (GitHub Copilot), Starcoder (Huggingface), Evosuite (for Java); Siddiq et al. 2023

```python
from typing import Union


class Calculator:

    def __init__(self):
        '''Empty init'''
        pass

    def divide(self, number1: Union[int, float], number2: Union[int, float]
        '''
        Parameter:
            number1 (int or float): First number to division
            number2 (int or float): Second number to division

        Returns:
            Float: Returns a division of two numbers informed
        '''

        self.number1 = number1
        self.number2 = number2

        try:
            self.result = self.number1 / self.number2

        except TypeError:
            raise TypeError

        except ZeroDivisionError:
            raise ZeroDivisionError

        return self.result

    def multiply(self, number1: Union[int, float], number2: Union[int, floa
        '''
        Parameter:
            number1 (int or float): First number to multiplication
            number2 (int or float): Second number to multiplication

        Returns:
            Float or Int: Returns a multiplication of two numbers informed
        '''

        self.number1 = number1
        self.number2 = number2

        if isinstance(self.number1, str) or isinstance(self.number2, str):
            raise TypeError

        self.result = self.number1 * self.number2
        return self.result

    def sum(self, number1: Union[int, float], number2: Union[int, float]):
        '''
        Parameter:
            number1 (int or float): First number to sum
            number2 (int or float): Second number to sum

        Returns:
```

1

```python
import unittest
from Calculator import Calculator


class TestCalculator(unittest.TestCase):

    def setUp(self):
        self.calculator = Calculator()

    def test_divide(self):

        self.assertEqual(self.calculator.divide(10, 5), 2)
        self.assertEqual(self.calculator.divide(-10, 5), -2)
        self.assertEqual(self.calculator.divide(10, -5), -2)
        self.assertRaises(ZeroDivisionError, self.calculator.divide, 10, 0)
        self.assertRaises(TypeError, self.calculator.divide, 10, "5")
        self.assertRaises(TypeError, self.calculator.divide, "10", 5)

    def test_multiply(self):

        self.assertEqual(self.calculator.multiply(10, 5), 50)
        self.assertEqual(self.calculator.multiply(-10, 5), -50)
        self.assertEqual(self.calculator.multiply(10, -5), -50)
        self.assertRaises(TypeError, self.calculator.multiply, 10, "5")
        self.assertRaises(TypeError, self.calculator.multiply, "10", 5)

    def test_sum(self):

        self.assertEqual(self.calculator.sum(10, 5), 15)
        self.assertEqual(self.calculator.sum(-10, 5), -5)
        self.assertEqual(self.calculator.sum(10, -5), 5)
        self.assertRaises(TypeError, self.calculator.sum, 10, "5")
        self.assertRaises(TypeError, self.calculator.sum, "10", 5)

    def test_subtract(self):

        self.assertEqual(self.calculator.subtract(10, 5), 5)
        self.assertEqual(self.calculator.subtract(-10, 5), -15)
        self.assertEqual(self.calculator.subtract(10, -5), 15)
        self.assertRaises(TypeError, self.calculator.subtract, 10, "5")
        self.assertRaises(TypeError, self.calculator.subtract, "10", 5)


if __name__ == "__main__":
    unittest.main()
```

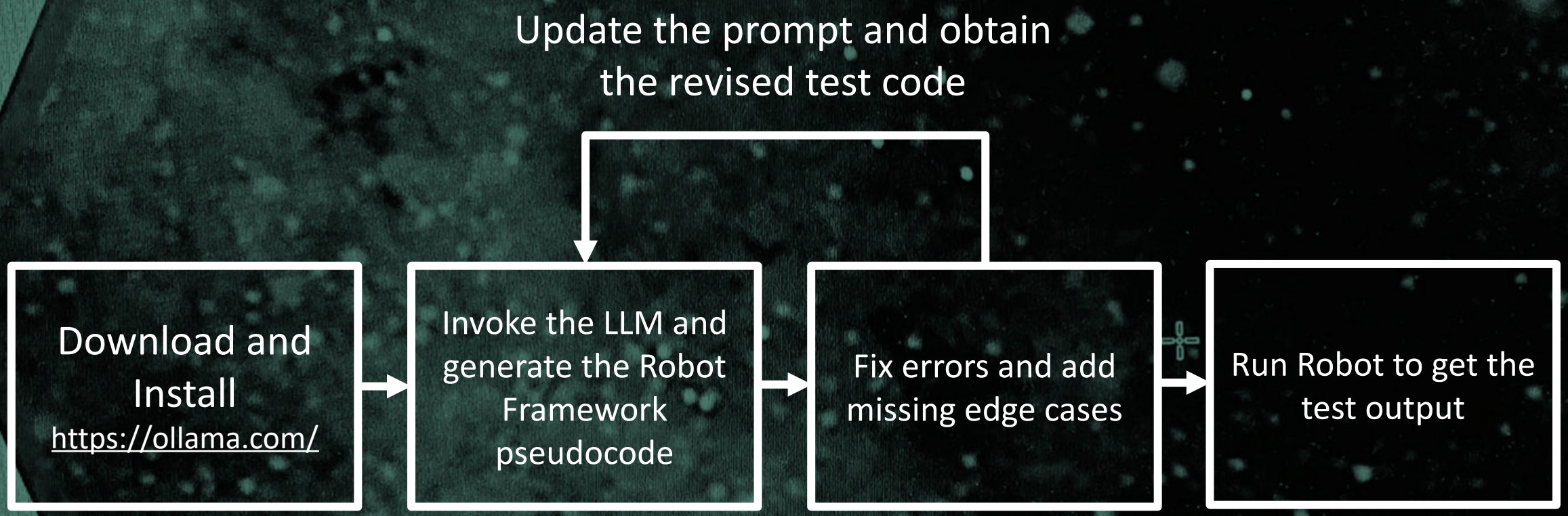
2

```python
import unittest

class TestCalculator(unittest.TestCase):
    def test_divide_correctly_handles_types(self):
        calculator = Calculator()
        self.assertRaises(TypeError, calculator.divide,
        self.assertRaises(TypeError, calculator.divide,
number")
        self.assertEqual(calculator.divide(2, 3), 2)

    def test_multiply_correctly_handles_types(self):
        calculator = Calculator()
        self.assertRaises(TypeError, calculator.multiply, 1, "not a
number")
        self.assertRaises(TypeError, calculator.multiply, 1.0, "not a
number")
        self.assertEqual(calculator.multiply(2, 3), 6)

    def test_sum_correctly_handles_types(self):
        calculator = Calculator()
        self.assertRaises(TypeError, calculator.sum, 1, "not a number")
        self.assertRaises(TypeError, calculator.sum, 1.0, "not a number")
        self.assertEqual(calculator.sum(2, 3), 5)

    def test_subtract_correctly_handles_types(self):
        calculator = Calculator()
        self.assertRaises(TypeError, calculator.subtract, 1, "not a
number")
        self.assertRaises(TypeError, calculator.subtract, 1.0, "not a
number")
        self.assertEqual(calculator.subtract(2, 3), -1)

    def test_divide_raises_zero_division_error(self):
        calculator = Calculator()
        with self.assertRaises(ZeroDivisionError):
            calculator.divide(0, 1)

    def test_multiply_raises_zero_division_error(self):
        calculator = Calculator()
        with self.assertRaises(ZeroDivisionError):
            calculator.multiply(0, 1)
```

Simple calculator unit test example

Which one is generated by AI?

In-Depth | Writing Tests With LLMs + Robot Framework



Example: An LLM Prompt

“Generate a Robot Framework script to perform MFA (Multi-Factor Authentication) login. The script should navigate to the login page, fill in the username and password fields, generate a TOTP code using the provided secret, enter the TOTP code, click the 'Sign in' button, and verify that the login was successful by checking the welcome message. Script only."

In-Depth | LLM Test Generation Output In Action.

```
(base) kyun@MT-510874 llm_unit_test % ollama run codellama "Generate a Robot Framework script to perform MFA (Multi-Factor Authentication) login. The script should navigate to the login page, fill in the username and password fields, generate a TOTP code using the provided secret, enter the TOTP code, click the 'Sign in' button, and verify that the login was successful by checking the welcome message. script only"
```

Here's the test output:

```
% robot --pythonpath . tests
```

LOG

Tests Report

Generated
20240304 06:28:06 UTC-08:00
3 days 4 hours ago

Summary Information

Status:	All tests passed
Start Time:	20240304 06:28:01.875
End Time:	20240304 06:28:06.285
Elapsed Time:	00:00:04.410
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	1	1	0	0	00:00:02	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Tests	1	1	0	0	00:00:04	
Tests . Mfa Login	1	1	0	0	00:00:04	

Test Details

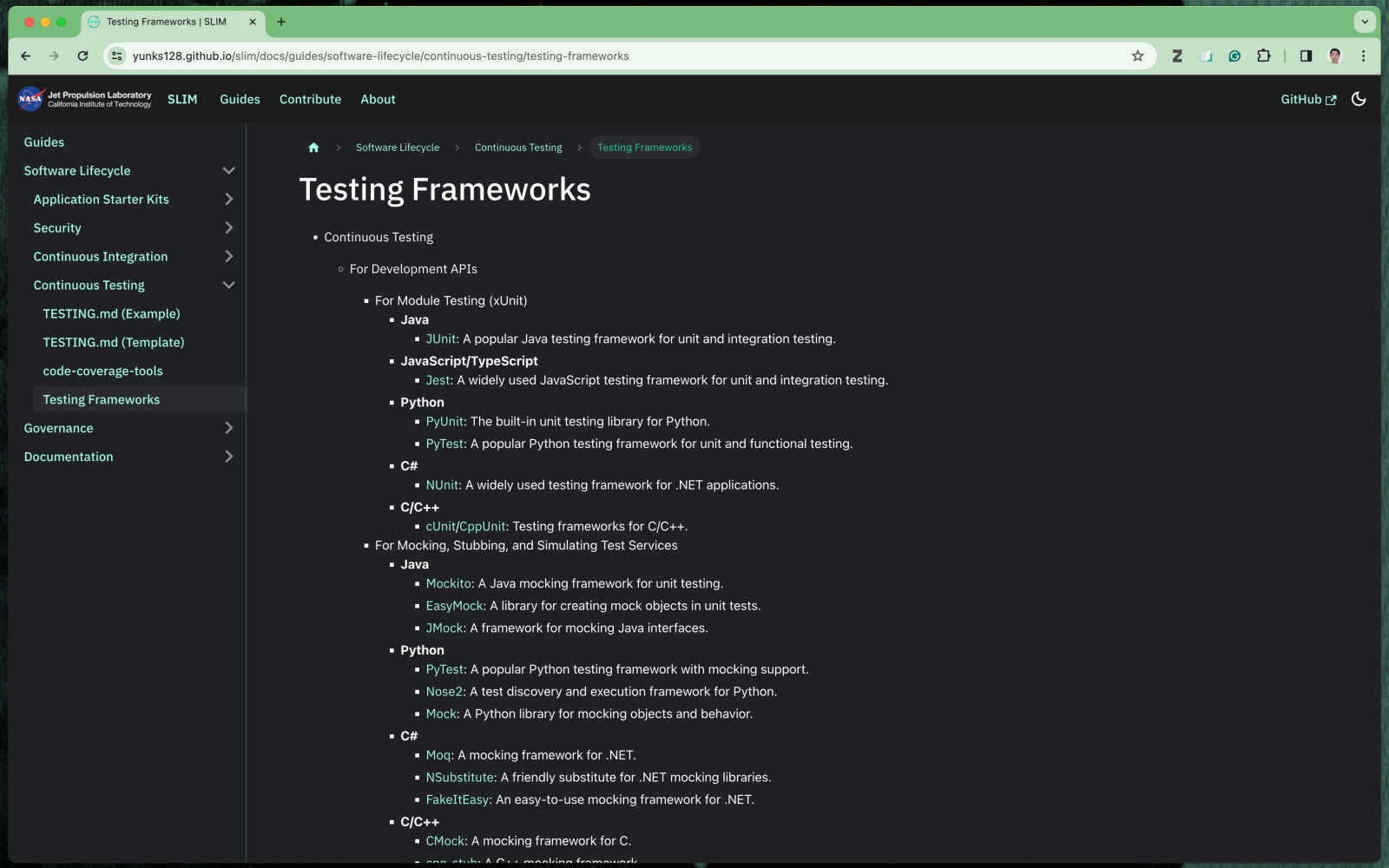
All | Tags | Suites | Search

Status:	1 test total, 1 passed, 0 failed, 0 skipped
Total Time:	00:00:01.831

Name	Documentation	Tags	Status	Message	Elapsed	Start / End
Tests . Mfa Login . Login with MFA			PASS		00:00:01.831	20240304 06:28:04.384 20240304 06:28:06.215

In-Depth | Tools and Frameworks for Running Tests

- Development APIs and Module Testing
- Mocking, Stubbing, and Simulating Test Services
- HTTP Services
- Static Analysis
- Dynamic Analysis and Test Coverage
- Complexity Analysis and Runtime Performance
- Security Testing
- Web Applications Testing

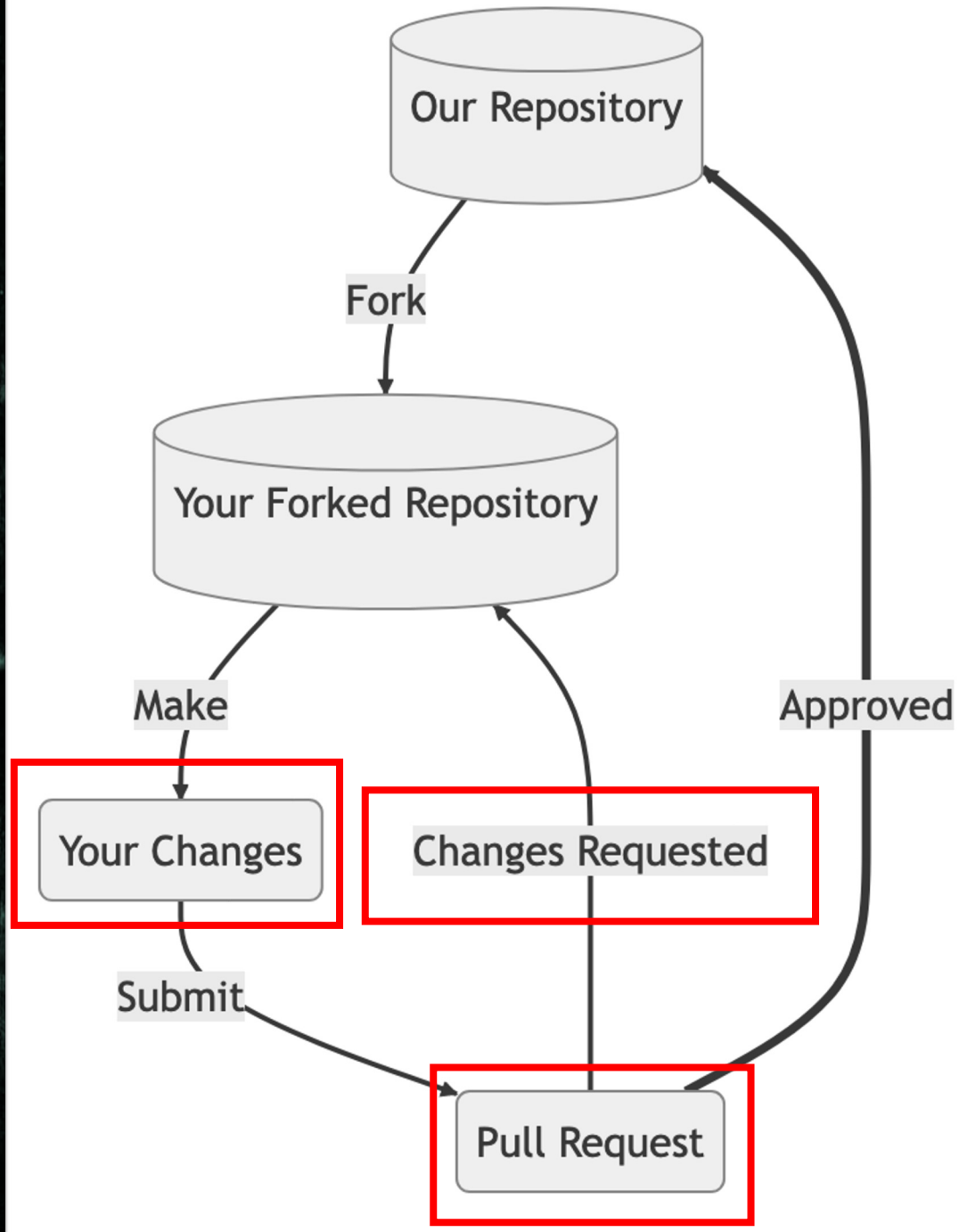


https://yunks128.github.io/slim/docs/guides/software-lifecycle/continuous-testing/testing-frameworks

Lessons Learned.

- Start Early: Begin testing from the outset of development to catch issues early. Use our **template**!
- Automate Wisely: Recommending automation tools like **LLMs and Robot Framework** makes infusion of best practices easier.
- Continuous Improvement: Regularly refine testing processes and our best practice guide to adapt to evolving project needs, for example: **Automate test execution**!

Join us in our effort to improve our latest guide:
https://github.com/NASA-AMMOS/slim/pull/144



Contributing to SLIM: Continuous Integration Best Practice

Presenter: John Engelke

Contributing a Continuous Integration Best Practice.

Overview of Our Open Source Standards Development Process

1. Identify a Best Practice Need

- **Through Community Engagement, We Identified Software Challenges:**
  - Reliability and reproducibility
  - A focus on scientific solutions (less on delivery)
  - Many project- or developer-level contributions
  - Managing integration with other (micro-)services
  - Compartmentalize outputs and reporting
  - Purpose-driven software or (micro-)services

Identified Needs Such As:

- Complexity
  - Moving pieces
  - Dependency tracking, testing and reporting
  - Modular deployments
- Deployments
  - Rapid development with ease of spin up
  - Build tooling non-standardization
- Auditing
  - Traceability
  - Security

3. Engage and Get Community Input

Contributing a Continuous Integration Best Practice.

Overview of Our Open Source Standards Development Process

1. Identify a Best Practice Need

2. Develop Best Practice Standard

3. Engage and Get Community Input

Contribution Model

- Leveraged SLIM's community contribution model
- Created a ticket where we designed an architectural solution satisfying earlier needs
- Iterated many versions a best practice guide to outline our CI recommendation, including with tooling suggestions

Proposed Best Practice

- Reference Architecture
  - Implementation guide for CI Best Practices
  - UML diagrams delineate concerns and approach (Single Source of Truth, Fail Fast/Fix Fast, Visibility/Open Results, On-commit Testing)
- Tooling Recommendations
- Starter Kits (templates) with Turnkey Standards
  - Built-In CI Tooling with Repository Publishing Automation
- Documentation As Code, Testing As Code

Contributing a Continuous Integration Best Practice.

Overview of Our Open Source Standards Development Process

1. Identify a Best Practice Need

2. Develop Best Practice Standard

3. Engage and Get Community Input

Gathered Community Feedback

- Solicited project and user feedback through discussions and pull-request comments
- Updated iterative standards until satisfactory results

In-Depth | CI Reference Architectures

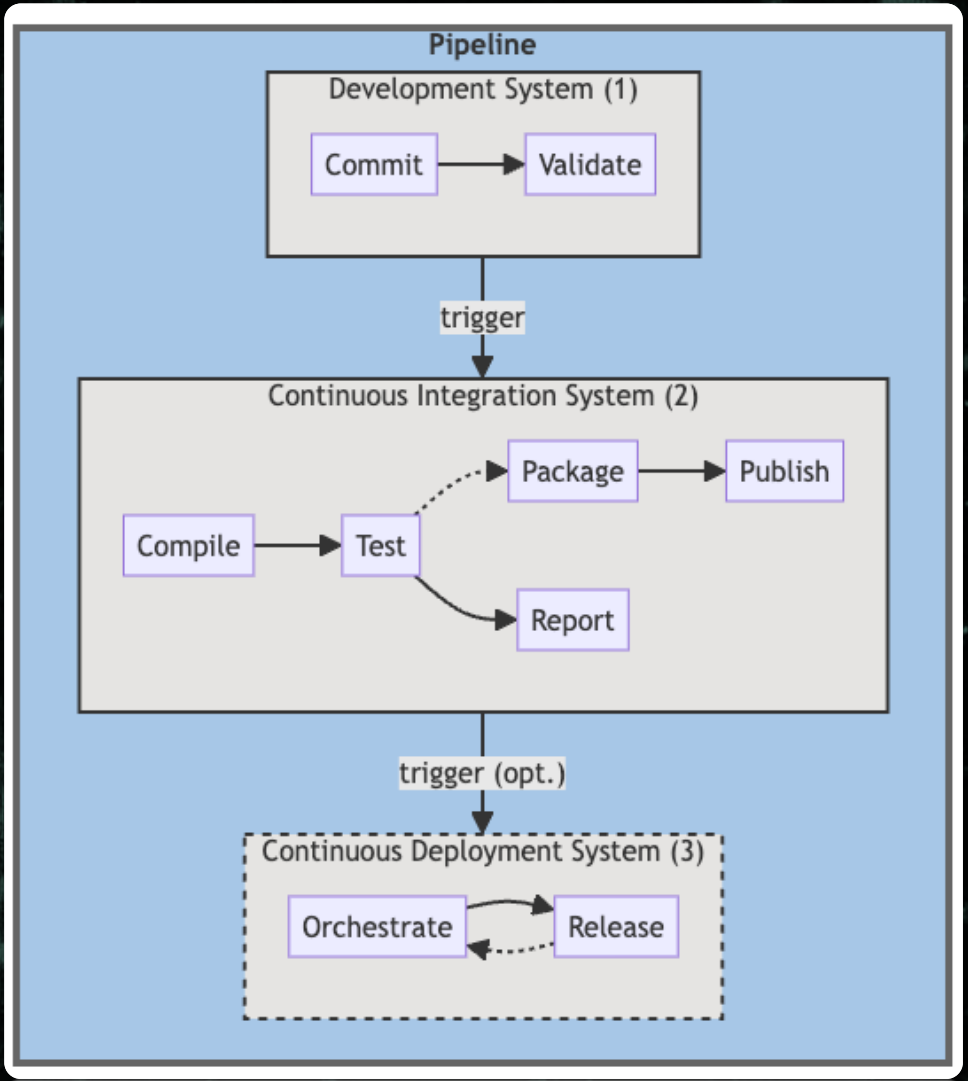
Philosophy and Practice

SLIM Documentation

- https://nasa-ammos.github.io/slim/
- ‘See our Best Practice Guides’ -> ‘Software Lifecycle’ -> ‘Continuous Integration’ -> ‘CI Reference Architectures’

CI Reference Architectures

- https://nasa-ammos.github.io/slim/docs/guides/software-lifecycle/continuous-integration/reference-architecture



In-Depth | CI Tooling Recommendations

Philosophy and Practice

SLIM Documentation

- https://nasa-ammos.github.io/slim/
- ‘See our Best Practice Guides’ -> ‘Software Lifecycle’ -> ‘Continuous Integration’ -> ‘CI Tools and Frameworks’

CI Reference Architectures

- https://nasa-ammos.github.io/slim/docs/guides/software-lifecycle/continuous-integration/continuous-integration-frameworks

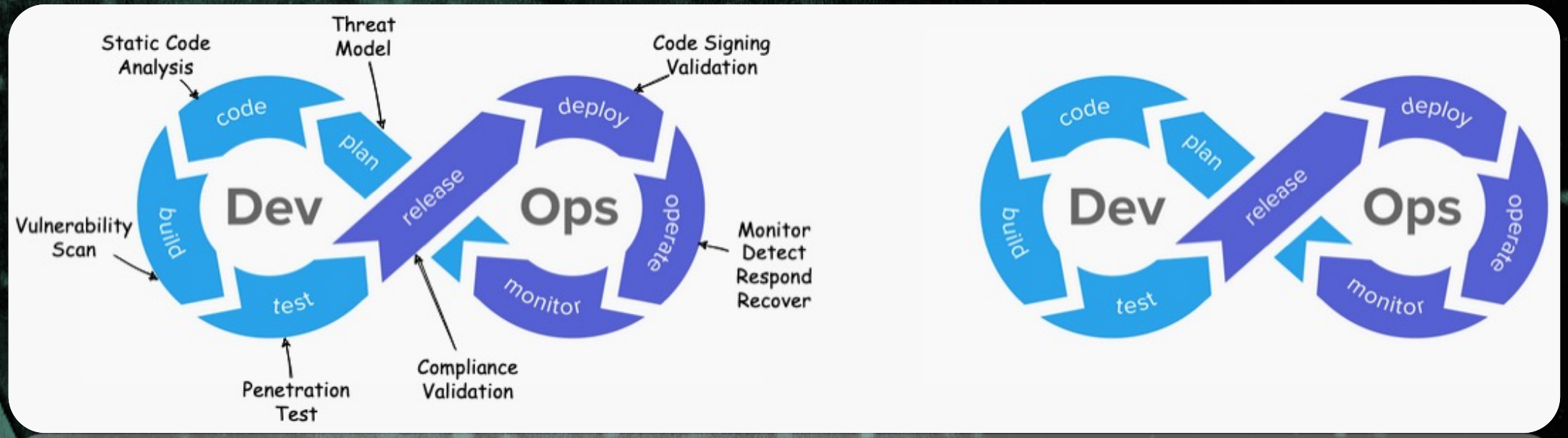
CI Tools and Frameworks

- Continuous Integration
  - For Analysis and Testing
    - For verification, notification and assembly
      - Using Git hooks
      - Using Github Actions
      - Implementing build tooling (e.g. Maven plugins, SetupTools, Make)
      - Using Checksum hashing
    - For credentialing
      - Implementing keystore Jenkins Credentials Binding Plugin
      - Using ssh
      - Using oauth
    - For executing and reporting tests
      - Using Jenkins plugins
      - Using TravisCI Build Addons
      - Using (Java) Maven plugins
      - Using (Python) SetupTools
      - Using (C#) NUnit
      - Using (C/C++) Make/Cmake
      - Using (Node.js) npm-test
      - Using (any) Testrail Connector
  - For Compilation
    - For build integration and reporting
      - Using Jenkins
      - Using TravisCI
    - For dependency management and packaging
      - Using (Java) Maven
      - Using (Python) SetupTools/Pip
      - Using (C#) NuGet
      - Using (C/C++) Make/Cmake
      - Using (Node.js) npm
      - Using (any) Ant
  - For Orchestration
    - For deploying services
      - Using workflows
        - Using Puppet
        - Using Ansible
        - Using Chef
      - Using scripts
        - Using Python
        - Implementing platform scripting [shell, Powershell]
    - For cloud or datacenter deployments
      - Using Terraform
      - Using Kubernetes
      - Using CloudFormation
      - Using SaltStack
  - For Release Management
    - For packaging
      - Using Docker
      - Implementing archiving (tar, zip, gz)
      - Using RPM
      - Using JAR, WAR
    - For releasing software
      - Using Jenkins ()
      - Using TravisCI
      - Using Github Actions
    - For storing build artifacts
      - Using software repositories
        - Using Artifactory
        - Using Nexus
      - Using OSS repositories
        - Using (Java) Maven Central
        - Using (Python) PyPi
        - Using (C#) Nuget
        - Using (C/C++) yum, dnf
        - Using (Node.js) npm

In-Depth: CI SLIM Starter Kits

Shift Left Philosophy

- Embed security (and more generally) quality and release checks into development [1]
- Starter Kits Provide Baseline CI Tools



[1] *Open Worldwide Application Security Project (OWASP), s.v. "OWASP DevSecOps Guideline v-0.2,"* accessed March 5, 2024, https://owasp.org/www-project-devsecops-guideline/latest/

In-Depth | CI SLIM Starter Kits

Java Starter Kit

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoc

    <modelVersion>4.0.0</modelVersion>

    <name>JPL - AMMOS - IDS - Sample Projects - SLIM Starterkit Java Simple</name>
    <description>Sample Projects -- A sample Java project using Maven to demonstrate a simple application configuratio

    <parent>
        <groupId>gov.nasa.ammos</groupId>
        <artifactId>parent-ammos</artifactId>
        <version>1.0.0</version>
    </parent>

    <groupId>gov.nasa.jpl.ammos.ids.sample_projects</groupId>
    <artifactId>maven-simple</artifactId>
    <version>${revision}</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <!-- Tagging for CI -->
        <semver>1.0.0</semver>
        <revision>${semver}${buildnum}${sha1}${changelist}</revision>
        <!-- BUILD PARMS -->
        <!-- VERSIONING -->
        <version.junit>4.13.2</version.junit>
        <version.slf4j>1.7.36</version.slf4j>
    </properties>

    <scm>
        <connection>scm:git:ssh://git@github.com:NASA-AMMOS/slim-starterkit-java.git</connection>
        <developerConnection>scm:git:ssh://git@github.com/NASA-AMMOS/slim-starterkit-java.git</developerConnection>
        <url>https://github.com/NASA-AMMOS/slim-starterkit-java/tree/main</url>
    </scm>

    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.gmaven</groupId>
                <artifactId>groovy-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${version.junit}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
```

Automation-based, CI-friendly Versioning

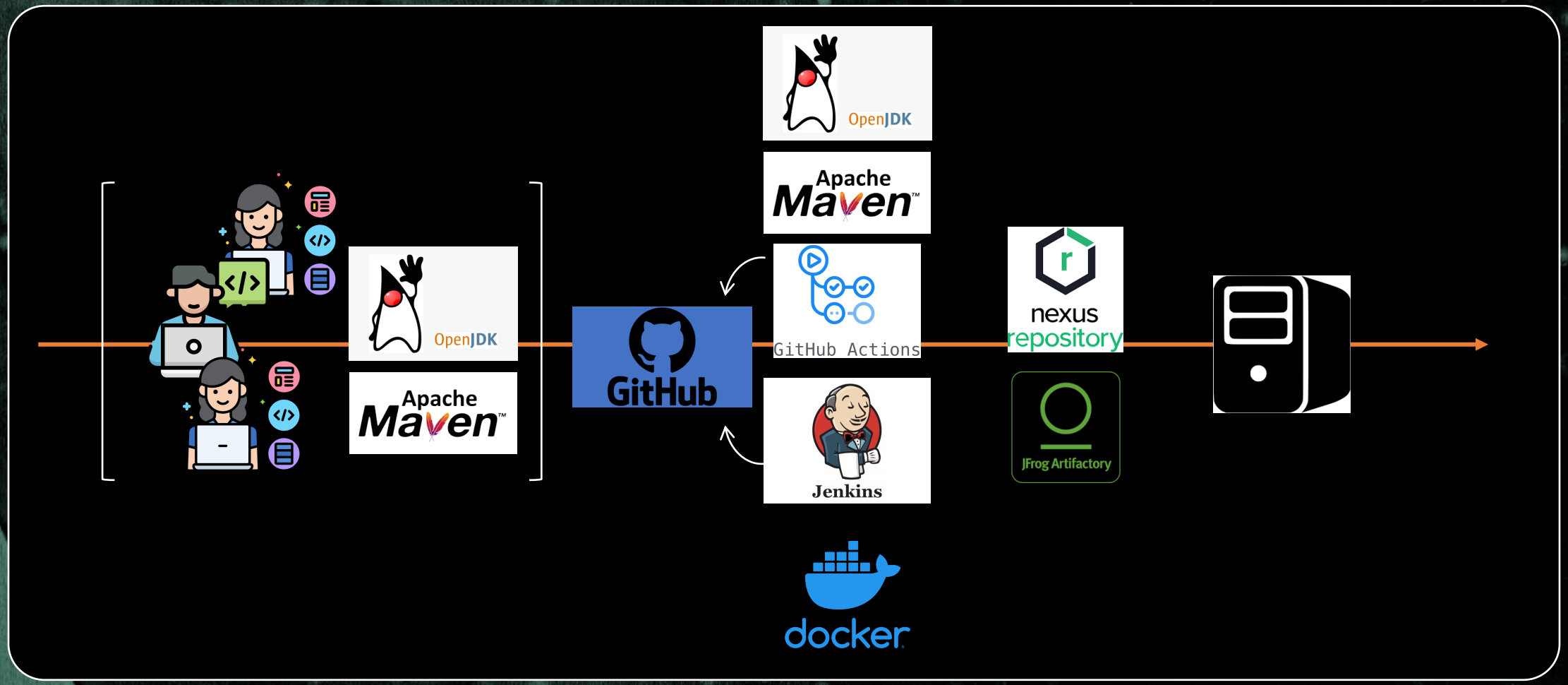
- Build/archive on code push
- Release/publish on code tag

Toolchain

- JDK
- Git
- Jenkins/GH Actions
- Maven
- Artifactory/Nexus

In-Depth | CI SLIM Starter Kits

Java Starter Kit



In-Depth | CI SLIM Starter Kits

Java Starter Kit

Tracking System Touchpoints

- Workstation: Code, at modification time – Semantic Version
- CVS: Git, at commit time – Commit Hash
- CI Service: Jenkins, at build time – build number
- Artifact Repository: Artifactory, at publish – SNAPSHOT flag

Identify code origins precisely from 1st commit onward

Versioning Examples

- **SNAPSHOT:** mars-3.8.0b52-bb59d69-SNAPSHOT.jar
- **Release:** mars-3.8.0.jar + *version file* in JAR contents

In-Depth | CI SLIM Starter Kits

Python Starter Kit

SLIM Standards in a Single Repository

- Instant, development-ready GitHub Application
  - Rapid implementation via GH Templates
  - Plug-and-Play CI
  - Automated repository publishing on tag
- Documentation as Code
  - Ticket and Pull Request templates
  - Ready for Small and Large Teams
- Testing as Code
- Gateway to key Shift Left security features

In-Depth | CI SLIM Starter Kits

Python Starter Kit: Documentation Integration

README.md

[INSERT YOUR LOGO IMAGE HERE (IF APPLICABLE)]

[INSERT YOUR REPO / PROJ NAME HERE]

```
[INSERT A SINGLE SENTENCE DESCRIBING THE PURPOSE OF YOUR REPO / PROJ]
```

[INSERT YOUR BADGES HERE (SEE: https://shields.io)] Best Practices from SLIM

[INSERT SCREENSHOT OF YOUR SOFTWARE, IF APPLICABLE]

[INSERT MORE DETAILED DESCRIPTION OF YOUR REPOSITORY HERE]

[INSERT LIST OF IMPORTANT PROJECT / REPO LINKS HERE]

Features

- [INSERT LIST OF FEATURES IMPORTANT TO YOUR USERS HERE]
- Python build tooling based on PEP-517 and PEP-518 standards
- Build, release and publish automation takes place automatically using GitHub Actions.

jpl-jengelke NASA-AMMOS/

- .github
- slim_sample_project
- tests
- .gitignore
- CHANGELOG.md
- CITATION.md
- CODE_OF_CONDUCT.md
- CONTRIBUTING.md
- LICENSE
- MANIFEST.in
- README.md
- pyproject.toml
- requirements.txt
- setup.cfg
- setup.py

In-Depth | CI SLIM Starter Kits

Python Starter Kit: Scanning Features

Software Composition Analysis

This section contains links to sample actions, templates and configurations that analyze and validate composition of Open Source Software (OSS) components in software systems. Identifying software and licensing vulnerabilites and ensuring routine software updates is an OSS cybersecurity best practice.

Dependabot

A GitHub ecosystem tool for dependency version and security vulnerability analysis.

Automated Dependency Updates

This Dependabot task provides an automated check for OSS component updates and automatically creates pull requests to commit new versions.

Starter Kit:

- SLIM Starterkit Python -- Dependabot Script to install in your GitHub repo

In-Depth | CI SLIM Starter Kits

Python Starter Kit: Built-In Package Index Publishing

- Traceability with versioning through Setup Tools SCM (Git-synchronized)
- GitHub Actions Containerized Builds
  - Code Testing
  - Package Validation
- PyPi (Python Package Index) release on Tag
  - Integrated PyPi documentation metadata
- Fail Fast alerts on build/release error
- Security Alerts

Shift Left Fast: A Deployable Python App in 10 Minutes

Friday, March 15, 2024 - 13:45 to 14:00 | Ballroom DE

Questions?

Join Us @ https://nasa-ammos.github.io/slim

Speaker Contacts

Rishi Verma (rishi.verma@jpl.nasa.gov)
Kyongsik Yun (kyongsik.yun@jpl.nasa.gov)
John Engelke (jengelke@jpl.nasa.gov)

A Special Thanks To

- Lan Dang
- The SCaLE Team
- The SLIM community