QA
Overview

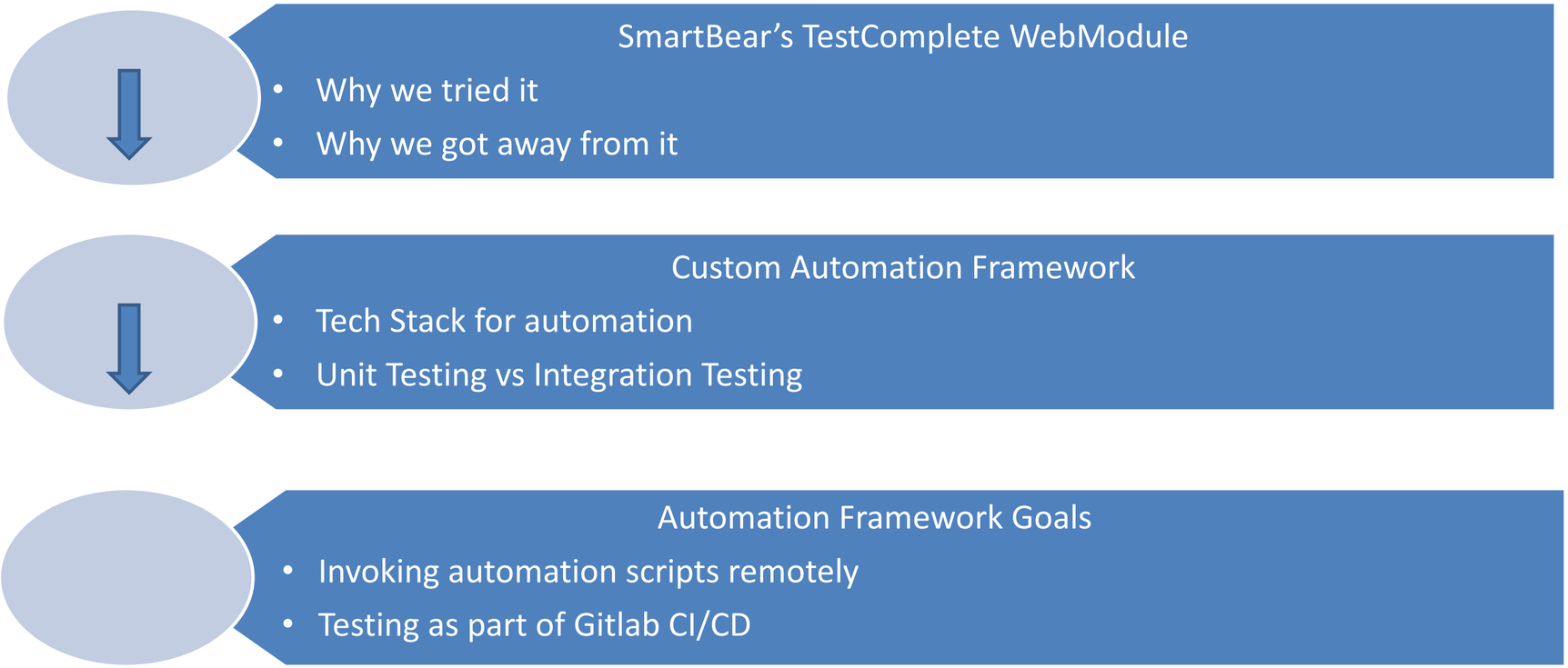
SIG
SUSQUEHANNA

Introduction

- About Me…
- Stakeholders
  - Internal vs External
  - Impact on QA
    - Business impact evaluation
- Value based decision Making at SIG
  - How we analyze value in Software development
- Roles and Responsibilities
  - Being part of a Software development team

SIG
SUSQUEHANNA

Journey



SIG SUSQUEHANNA

TestComplete: Why we tried it

Functional Testing Steps

1. List your application functions and features.
2. Decide on input data.
3. Specify expected output.
4. Create test cases.
5. Compare actual and expected output.

TestComplete: Why we tried it

- Ease of integration with tools like Jira, QAComplete etc.
- Recording automations vs hand coding them
- Cross browser testing
- Dedicated support
- Ability to script tests in multiple languages
- Remote execution of tests with QAComplete
- Allows non-technical users to create automation tests

TestComplete: Why we got away from it

- Auto generated scripts are not consistent between different applications
  - It was easier for smaller applications but got time intensive for larger data intensive applications
- Timing issues with recordings
  - This was not the case with all applications but not all applications are expected to perform similarly, and we had to incorporate WAIT times between test steps
- OCR and Coordinates based testing for some of our applications was not straight forward with TestComplete
- We realized that we need remote execution of tests without having a dependency on their tool QAComplete which requires additional licenses
- Developers preferred using IDE's like Visual studio compared to TestComplete's scripting IDE
- Additional infrastructure requirements to manage all SmartBear tools
- Automation testers that prefer programming inclined creating their own scripts rather than using TestComplete's auto generated scripts

Automation Framework

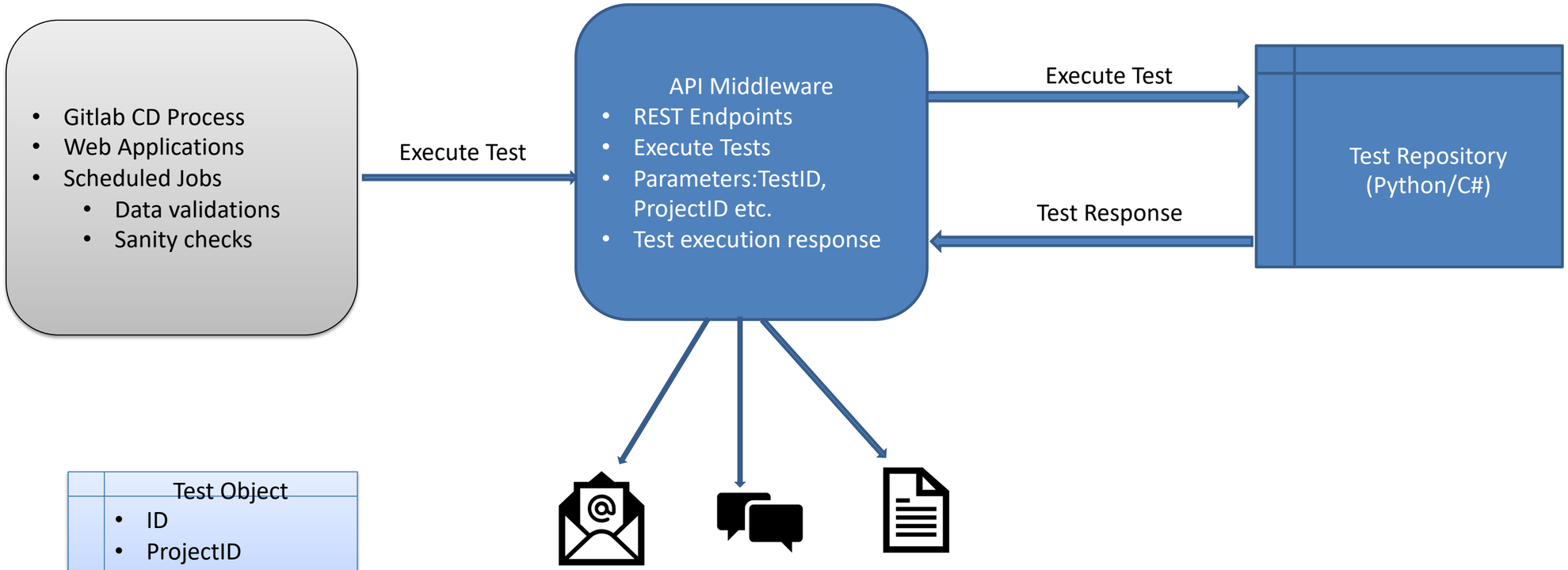
TECH STACK

SIG SUSQUEHANNA

Tech Stack

- Automation Languages: C# & Python
  - IDE: Visual Studio & Visual Studio Code
  - Framework: .Net Core, Nunit, Selenium WebDriver
  - Database: SQL Server
- API Frameworks
  - ReadyAPI
  - Asp.Net Core Web APIs

Architecture Overview



Test Object
ID
ProjectID
ResponseMetadata
ScheduleRunAt

Corporate Systems Environment

Application Types

GUI Based

- Web Applications
- Azure Cloud Web Applications
- Public WordPress sites
- Vendor GUI Applications
- Etc..

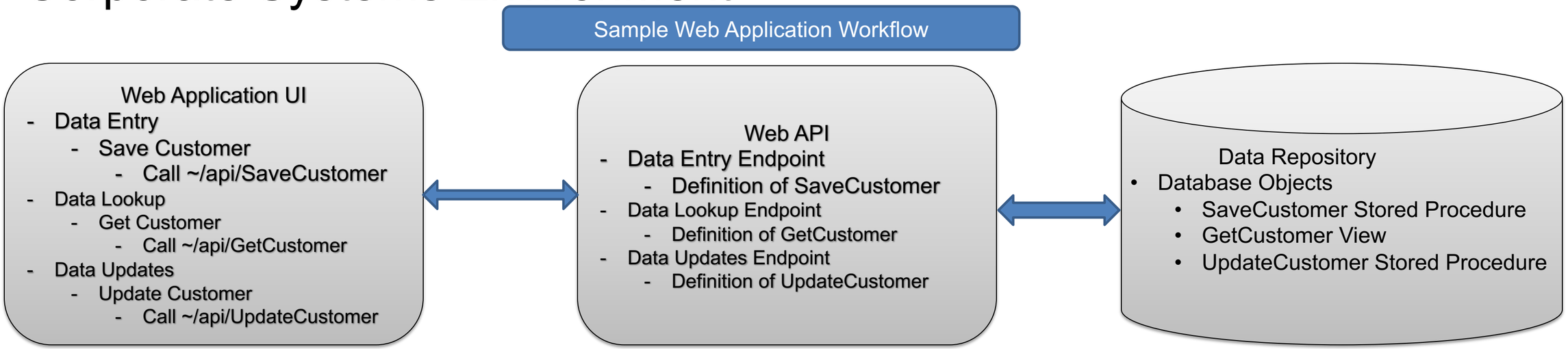
Headless

- Data Integration Jobs
- Scheduled Jobs
- Web services
- Azure Cloud Functions
- Vendor Applications
- Etc..

Unit Testing vs Integration Testing

- Team manages 150+ applications of various types and each application has connections into one more other applications
- Our focus primarily is on Integration testing
- We get most value from our tests by making them validate end to end rather than testing specific functions
- There are several common internal code libraries that are good candidates for Unit testing

Corporate Systems Environment



Test 1: Web App UI Save Data Test Script

1. Open browser with URL
    1. Select elements and set field values
    2. Submit form data
2. Validate for success message on the webpage
3. Close Browser

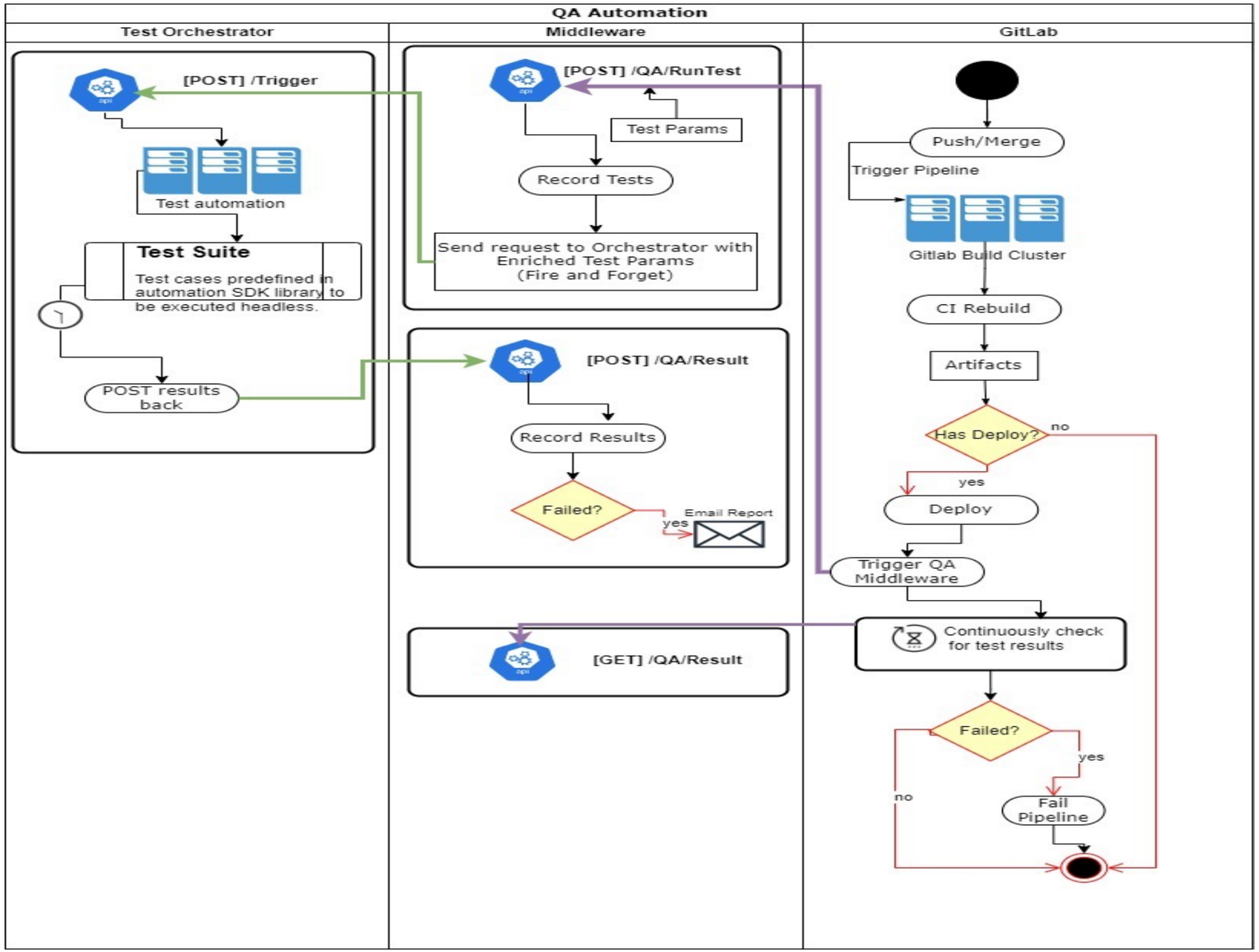
Test 1 Coverage:

- Direct Coverage: Web Application Loads, Customer data can be entered, and data submission was successful
- Indirect Coverage:
  - Since there were no API errors, we can infer that API SaveCustomer endpoint was validated
  - In the database SaveCustomer stored procedure was successful

Automation Goals

Architecture

SIG SUSQUEHANNA



- Any push or merge into a branch with configured QA job step will trigger automation test on Middleware.
- Middleware will package “Test Params” and other args into a request payload and send it over to Test Orchestrator via API endpoint. Also return a unique ID to GitLab as a response.
- Test Orchestrator, a package of various test suites that can be executed headless, will execute request test suite. After completion will post results back to Middleware.
- Middleware will record results and will send reports for failed tests.
- On the other end, GitLab will continuously check for results using the unique ID provided from QA/RunTest endpoint. If the result is a “Failure”, then it will mark the pipeline as failed and block any upstream merge requests.

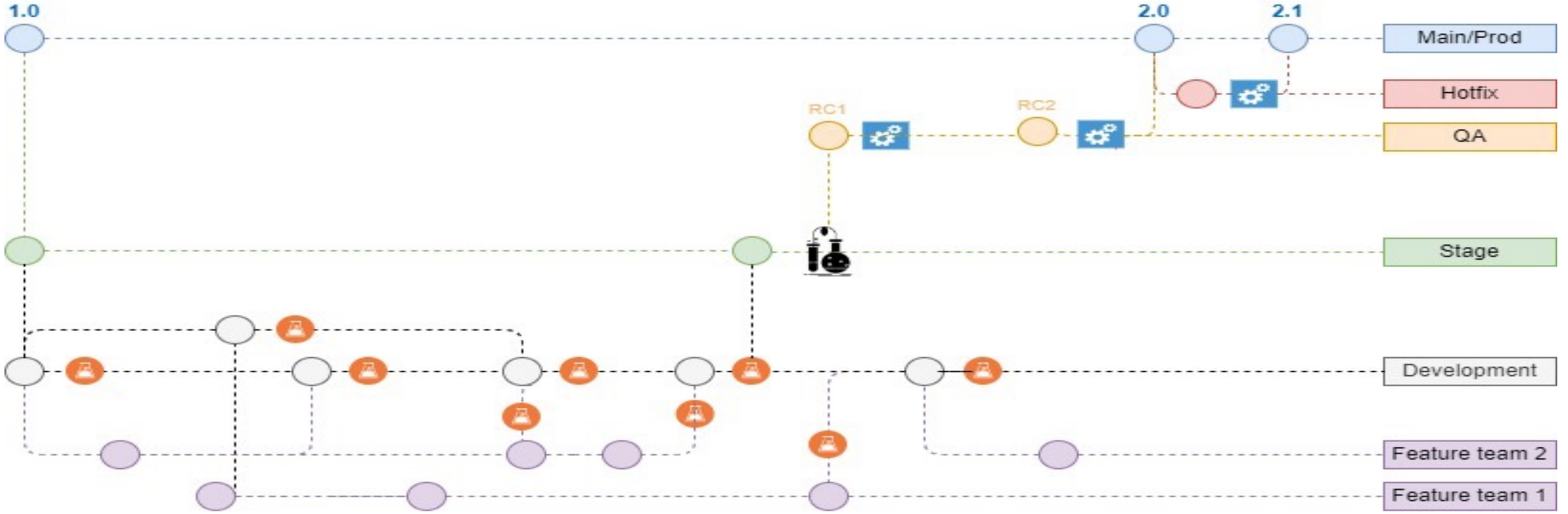
Test Suites as part of GIT Pipeline

Test suites are integrated across various pipelines to make sure QA processes are done before production release

- Continuous integration or sanity checks are performed on every push to development branch or on merge of feature branch into development. CI tests include cases like clean rebuilds (build verification), basic functions, DB connections, environment verification, etc.
- Smoke tests, that are identified as critical functions of the product, are performed at stage environment. At this level, the product is relatively stable version that could be tagged as the next release candidate after it passes these test cases. For smoke tests we validate cases like critical set of functions, load times, basic CRUD operations, specific bug fixes for release, new functional tests etc.
- Full regression tests cover all aspects of the product. These tests will includes making sure the product still functions as expected after any code changes, updated or enhancements. Set of full regression tests are triggered on any merge to QA branch or on a Hotfix branch.
- To ensure no code is pushed without regression tests, branches like QA, Hotfix and Main are protected by blocking any code commits. This helps us ensure changes coming to higher environments have already gone through CI and Smoke tests.

```yaml
stages:
  - build
  - deploy
  - QA_Test

build_job:
  stage: build
  script:
    - 'Get-Location'
    - 'dotnet restore SampleAPI.sln'
    - 'dotnet build SampleAPI.sln /t:rebuild /p:Configuration=Release'

dev_deploy_job:
  stage: deploy
  variables:
    PUBProfile: "DEV"
  environment: "DEV"
  only:
    - dev
  script:
    - powershell -File C:\GitLabRunner\Deploy.ps1

dev_run_tests:
  stage: QA_Test
  only:
    - dev
  allow_failure: false
  script:
    - Start-Sleep -s 30
    - Invoke-WebRequest http://testrunner/api/v1/test_api -Method POST -ContentType "application/json" -Body '{"api_file":"Sample_tests.json"}'
    - Start-Sleep -s 30
    - Invoke-WebRequest http://testrunner/api/v1/test_api -Method POST -ContentType "application/json" -Body '{"api_file":"Sample_tests_apikey.json"}'

QA_dev_job:
  stage: QA_Test
  variables: #Job specific variables.
    TestId: 2786
    QAProjectID: 12919
  dependencies:
    - dev_deploy_job
  only:
    - dev
  script:
    - powershell -File C:\GitLabRunner\QA_Test.ps1
```

GIT Pipeline representing modes of QA



Continuous integration testing or sanity checks. Performed on each commit on development branch or git merge into development

Smoke testing, making sure the most basic functions are working.

Fully automated regression testing.

Questions

THANK YOU

SIG SUSQUEHANNA