TU Clausthal

Embedded DevOps - Presentation 2021-07-07

Behaviour Driven Development

J. Beck, M. Ibrahim, Institute for Informatics, Aeronautical Informatics

What comes next

- Knowledge transfer to pikei
- This is a short summary of important ideas anf features we will discuss further in the coming month
- Will be uploaded later onto the Github repo

Structure

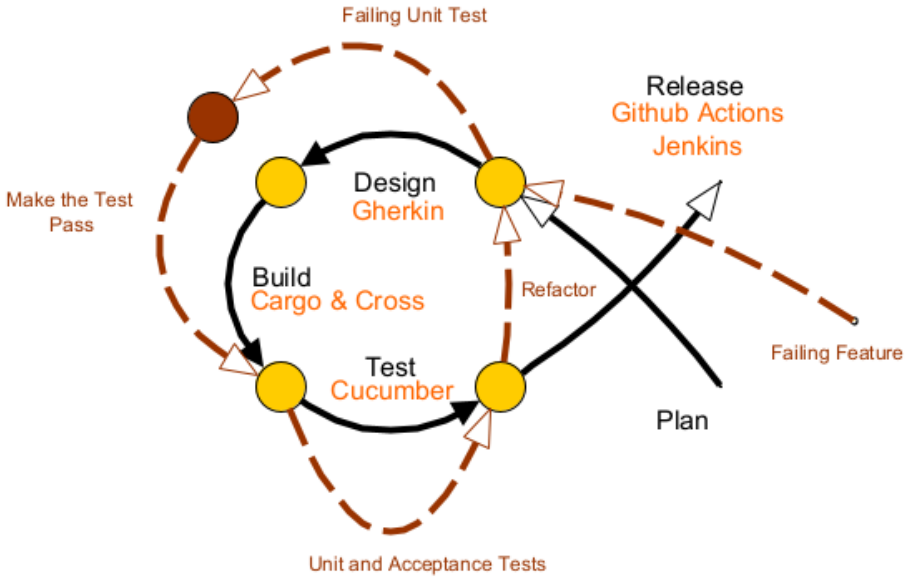
BDD Cycle and Design

From Text to Test

Jenkins/Cucumber Testing

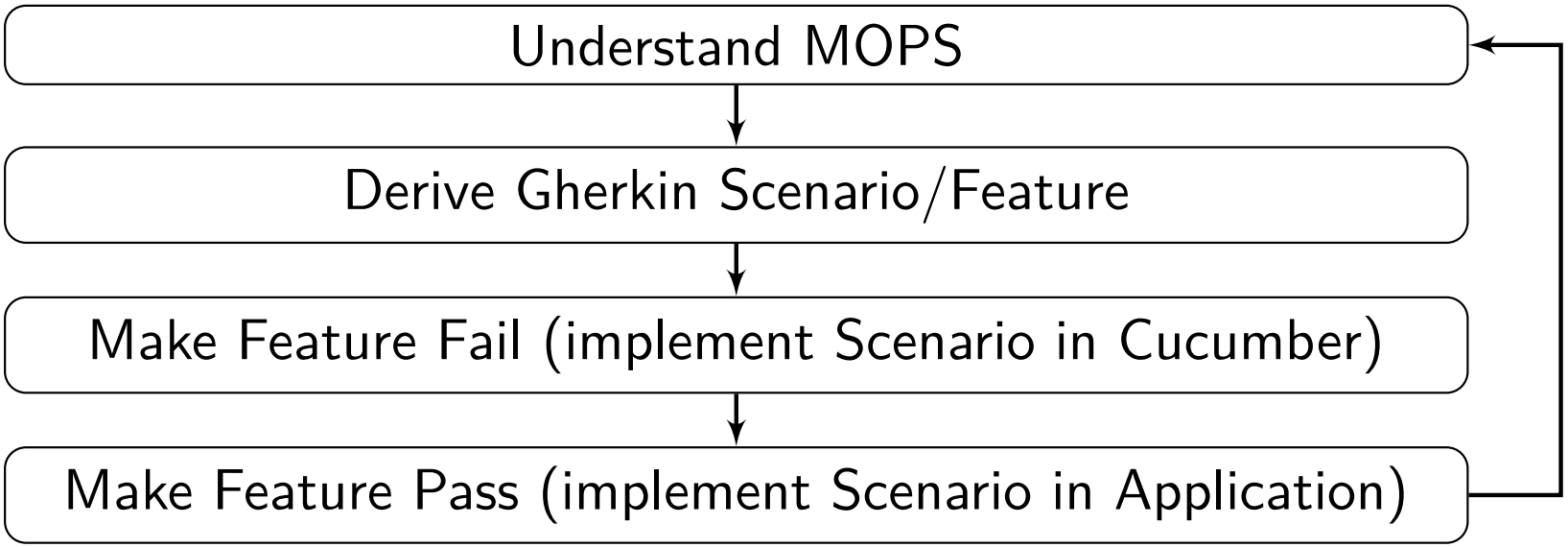
Test and Deployment

BDD Cycle and Design



- BDD tests with Gherkin language for behaviour description and Cucumber feature files as framework
  - Each greater functionality has its own feature and file
- Unit tests in code with `[\#test]` macros
  - In the same file that contains the functions that are tested
  - Ensure the correctness of important functions
- Additional testing with Rust doc-strings

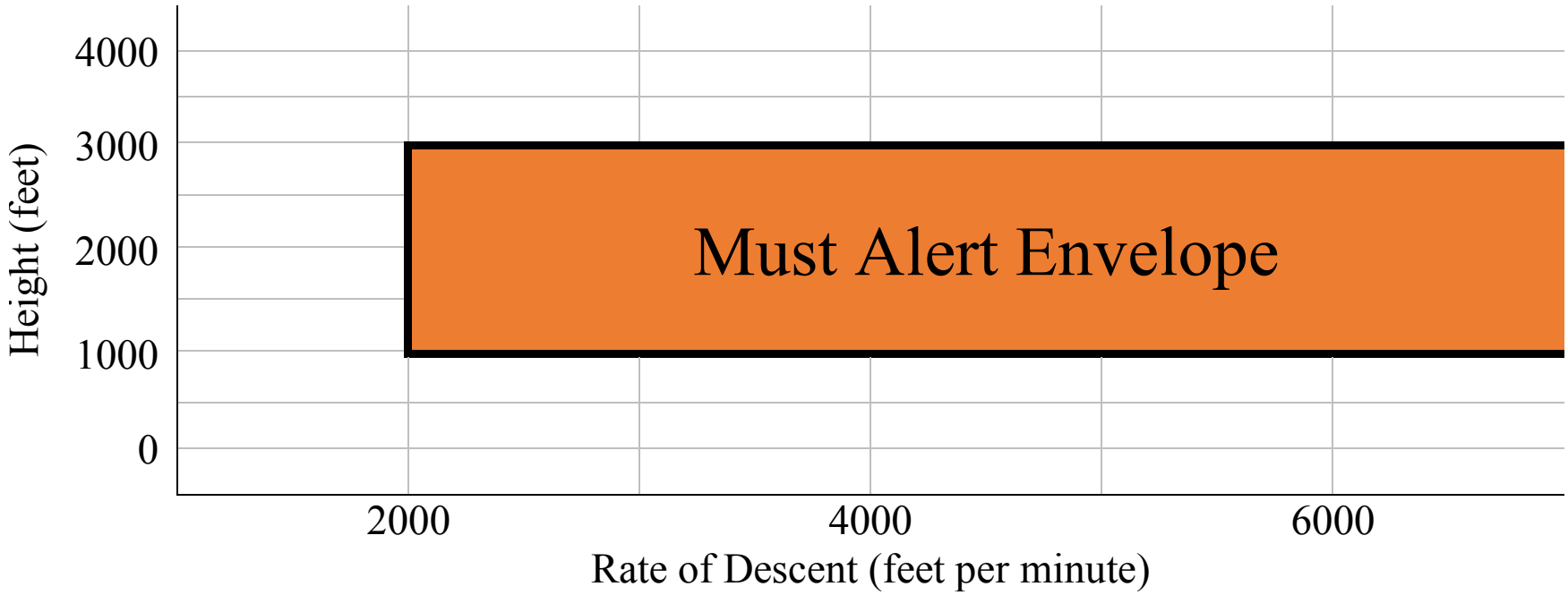
BDD Cycle


Understand MOPS
Derive Gherkin Scenario/Feature
Make Feature Fail (implement Scenario in Cucumber)
Make Feature Pass (implement Scenario in Application)

Understand MOPS

A simplified variant of MOPS_269 from DO-367

Class C Equipment shall (MOPS_269) provide a caution alert when Mode 1 is armed and the combination of rate of descent and height above terrain is within the Must Alert envelope prescribed in the figure below.



Derive Gherkin Scenario

```
Feature: Mode 1, Excessive Rate of Descent

  Scenario: Caution Alert
    Given Mode 1 is armed
    When the rate of descent is at least 2000 feet per minute
    When the height above terrain is between 1000 feet and 3000
↪   feet
    Then provide a Mode 1 caution alert
```

Implement Scenarios in Cucumber I

```
builder.given("Mode 1 is armed", |mut world, _step|{
    if ! world.taws.armed_modes.contains(&Alert::Mode1){
        world.taws.armed_modes.push(Alert::Mode1);}
    world
})
.when_regex(r"the rate of descent is at least (\d+) feet per
↪   minute", |mut world, matches, _step|{
    let rate_of_descent:f64 = matches[1].parse().unwrap();
    world.taws.roc = -rate_of_descent;
world
})
```

Implement Scenarios in Cucumber II

```
.when_regex(r"the height above terrain is between (\d+) feet and
↪   (\d+) feet", |mut world, matches, _step|{
    let (lower, upper) =
↪   (matches[1].parse().unwrap(),matches[2].parse().unwrap());
    world.height_values = vec![lower, upper, (lower+upper)/2.0 ];
    world
})
.then("provide a Mode 1 caution alert", |mut world, _step|{
    for height_value in &world.height_values {
        world.taws.height = *height_value;
        if ! world.taws.get_alerts().contains(&(Alert::Mode1,
↪   AlertLevel::Caution)){
            panic!("TAWS did not yield the Mode 1 Caution alert
↪   for {:?}", world);}}
    world
})
```

Generate Report

```
wucke13@zorn:~/doc*/u*/semester9/R*/s*/gherkin-demo-repo> cargo test cucumber
    Finished test [unoptimized + debuginfo] target(s) in 0.02s
     Running target/debug/deps/gherkin_demo_repo-93956050697d2924

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

     Running target/debug/deps/cucumber-40d8e382a3fa9586
[Cucumber v0.7.3]
Feature: Alert Priorization                                    features/alert_prioritization.feature:1:1

 Scenario: Alert Priorization                                  features/alert_prioritization.feature:11:5
  ✘ When concurrent alert conditions trigger                   features/alert_prioritization.feature:12:7
——— [!] Step failed: ——————————————————————————————————————— test/cucumber.rs:64:13
  No way to do this with the current API
————————————————————————————————————————————————————————————————————————————————

 Scenario: Alert Priority                                      features/alert_prioritization.feature:17:5
  - Given an alert can occur concurrently                      features/alert_prioritization.feature:18:7
    ⚡ Not yet implemented (skipped)

Feature: Mode 1, Excessive Rate of Descent                     features/mode_1.feature:1:1

 Scenario: Caution Alert                                       features/mode_1.feature:3:3
  ✔ Given Mode 1 is armed                                      features/mode_1.feature:4:5
  ✔ When the rate of descent is at least 2000 feet per minute  features/mode_1.feature:5:5
  ✔ When the height above terrain is between 1000 feet and 3000 feet  features/mode_1.feature:6:5
  ✔ Then provide a Mode 1 caution alert                        features/mode_1.feature:7:5

[Summary]
2 features
3 scenarios (1 failed, 1 skipped, 1 passed)
6 steps (1 failed, 1 skipped, 4 passed)

Finished in 0.5 seconds.
wucke13@zorn:~/doc*/u*/semester9/R*/s*/gherkin-demo-repo>
```

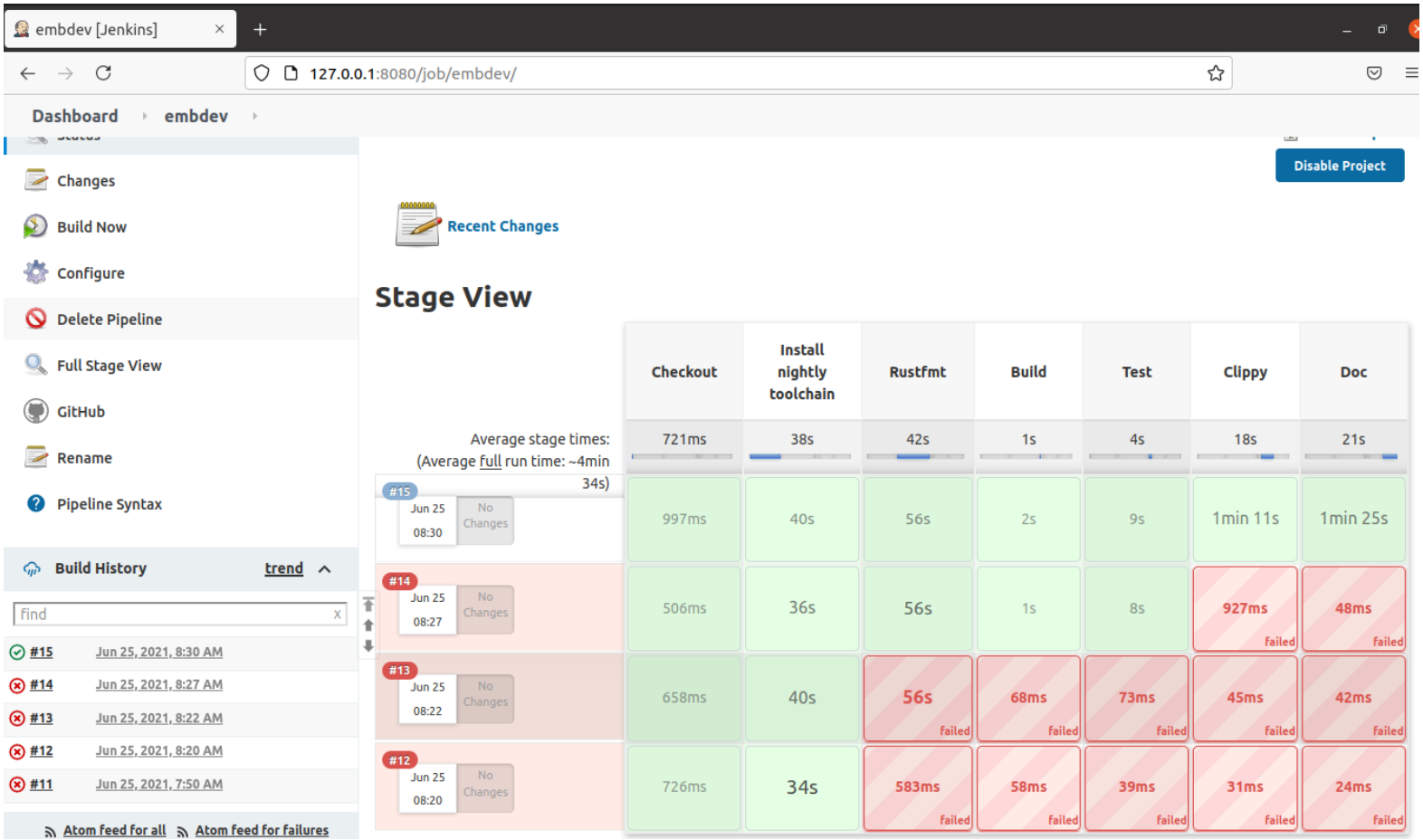
Jenkins Testing

- Docker agent containing Rust and Cargo
- Officialy maintained images available from dockerhub
- Building, testing and utility with shell Cargo commands
- No prepared commands or macros like Github actions

Simplified Example Pipleine

```
pipeline {
  agent {docker {image 'rust:latest'}}
  stages {
  stage('Checkout') {
      steps {
        git url: 'https://github.com/aeronautical-
            informatics/openTAWS'
}}
    stage('Rust Cargo') {
      steps {
        sh "cargo +nightly fmt --all -- --check"
        sh "cargo build"
        sh "cargo test"
        sh "cargo +nightly clippy --all"
        sh "cargo doc"
}}}}
```

Jenkins Execution



Test with QEMU

- Using Rust cross from the rust-embedded project
- Encapsulated QEMU environment to build and test software project
- Available with simple parameter in Github Actions
  `use-cross: true`
- Replaces the `cargo` command for shell commands i. e. in Jenkins
  - `cross build --target aarch64-unknown-linux-gnu`
  - `cross test --target mips64-unknown-linux-gnuabi64`
  - `cross run --target aarch64-unknown-linux-gnu`
- Supports many targets with most of them beeing testable
- Available in standard Rust docker image