EN
English

University of Oviedo

School of Computer Science

SOFTWARE ARCHITECTURE

Software Architecture

Acceptance tests

Escuela de Ingeniería Informática de Oviedo

2022-23

Jose Emilio Labra Gayo
Pablo González
Irene Cid
Cristian Augusto

Acceptance tests and BDD

- Tests that can be run in front of the client
  - If the tests pass, the product is accepted
- Behaviour-Driven Development (BDD)
  - Variant of TDD
    - Acceptance test driven development
  - Behaviour = User Stories
  - Also known as: *Specification by example*
  - Goal: Executable specifications
- Some tools:
  - cucumber, jBehave, concordion

BDD - User Stories

- Simple
- Readable by domain experts (business people)
- Approved by domain experts
- Other advisable characteristics:
  - Independent (with no strong relationships)
  - Negotiable (with no specific details)
  - Valuable for the customer
  - Estimable (to add them to Sprints)
  - Small (or consider division)
  - Testable (automatic tests)

User story structure

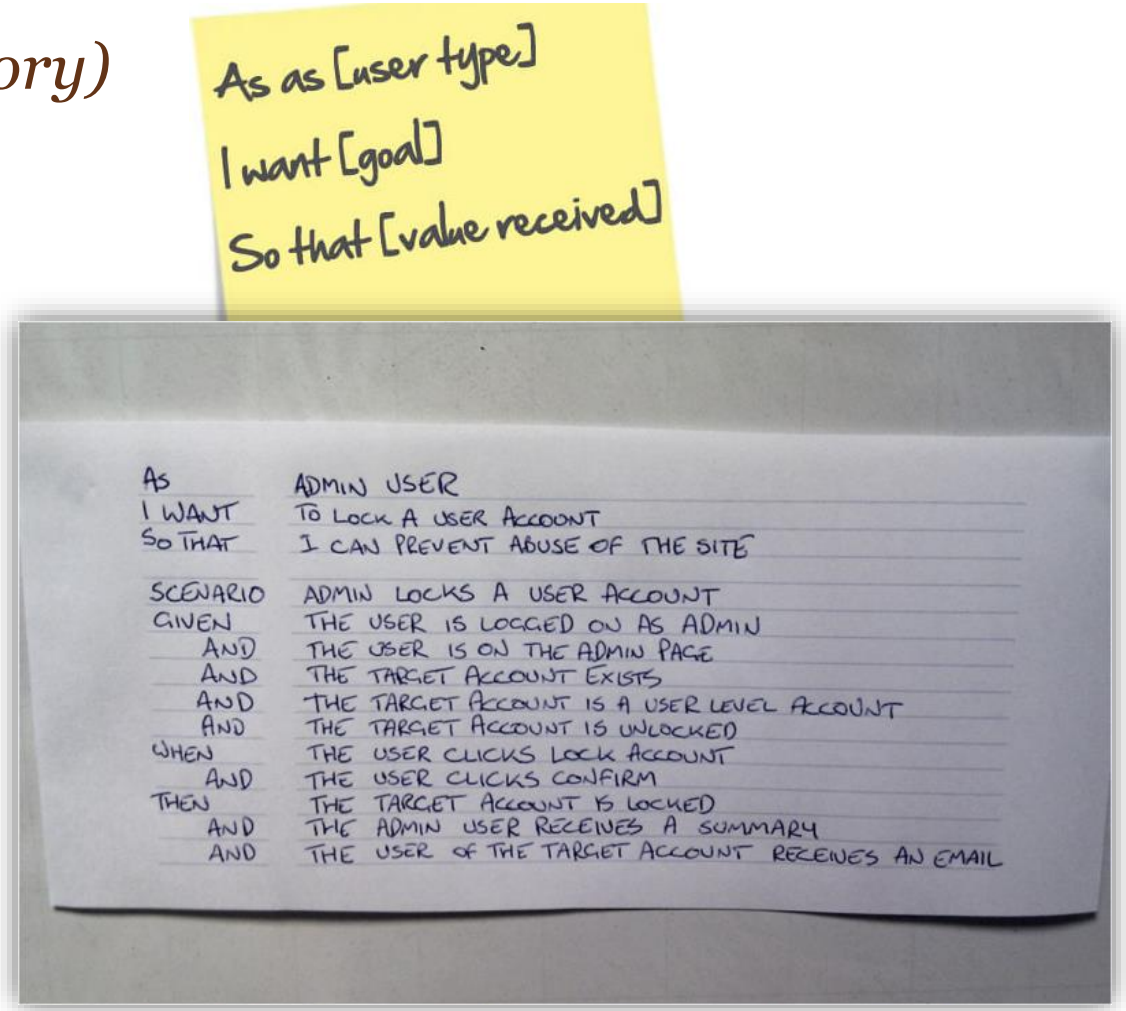
Feature: *Title (one line describing the story)*

The following structure is recommended:

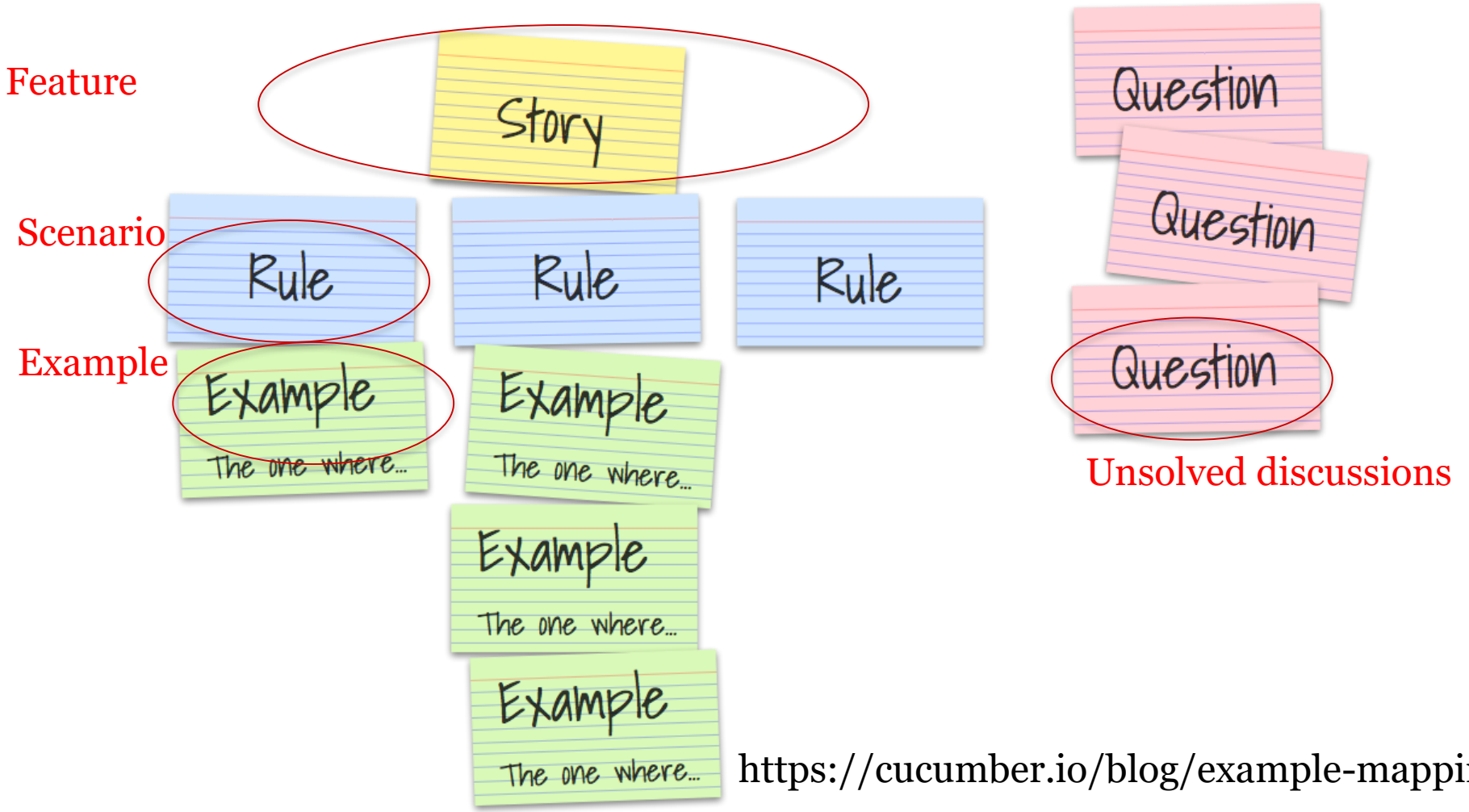
As a [role]
I want [feature]
So that [benefit]

Scenarios

Given [Context]
And [Some more context]
when [Event]
then [Outcome]
And [Another outcome]

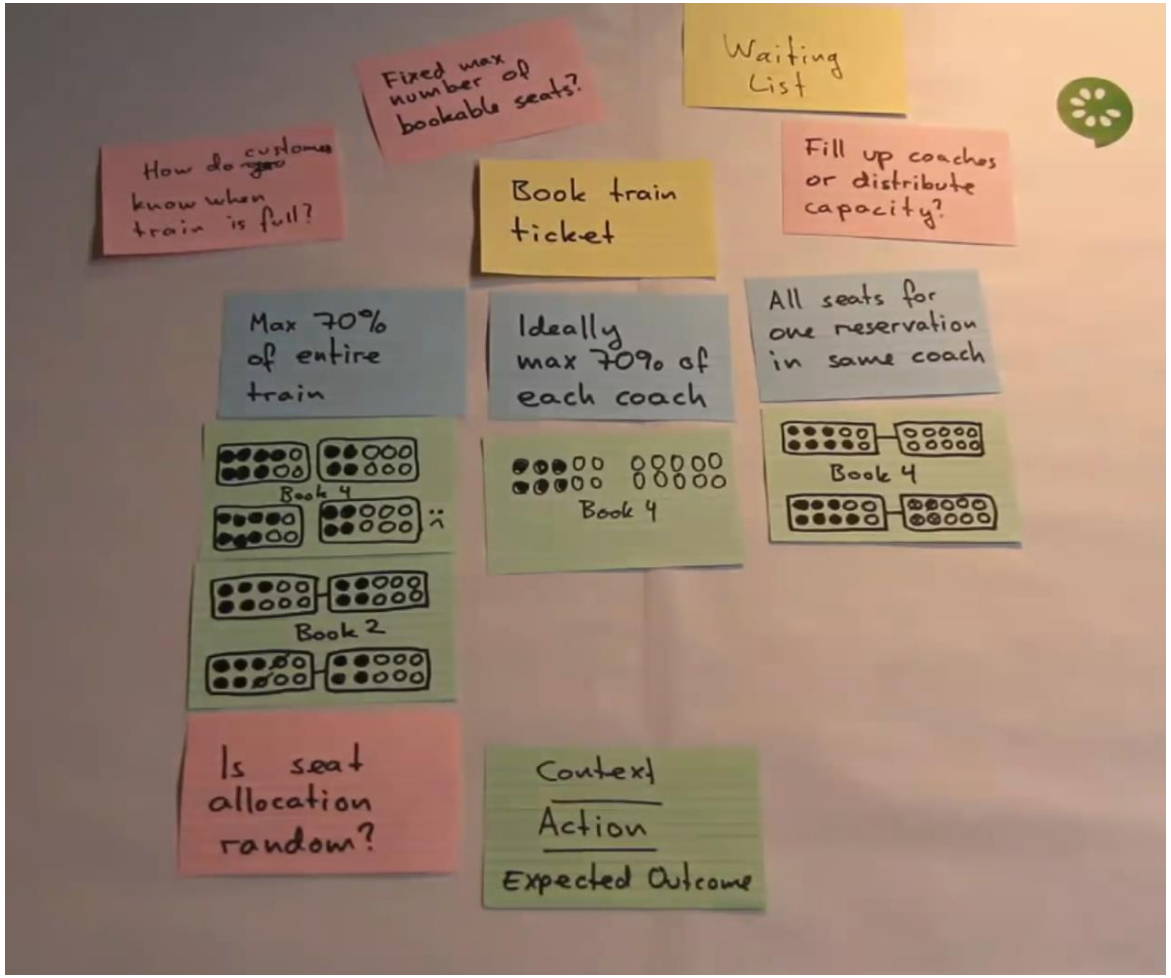


BDD – Example Mapping



https://cucumber.io/blog/example-mapping-introduction/

BDD – Example Mapping



[Introducing example mapping [video]]

BDD using Cucumber

- Cucumber = developed in Ruby (2008)
  - RSpec (Ruby), jbehave (Java)
  - Based on Gherkin
    - internal language to define user stories
  - Web: `http://cukes.info`
  - Support for multiple languages
- Java: cucumber-jvm
  - `https://github.com/cucumber/cucumber-jvm`

BDD using cucumber

- Features define some functionality
  - Gherkin language
    https://cucumber.io/docs/gherkin/
  - Can be used in several languages
- User stories are linked to step definitions
  - Step definitions can be run to validate user stories

BDD using cucumber

- Feature: Describes a system feature
  - A feature can have several scenarios
  - Scenario:
    - How must the system behave in some context
      - *Given*: Prepares scenario
      - *When*: Interact with the system
      - *Then*: Checks the state
      - *Examples*: Specific data

BDD

- Step by step guide to a user story
  - Install Cucumber
  - Write a first scenario in Gherkin
  - Write steps definitions in a chosen programming language
  - Run cucumber

BDD with cucumber

- Depends on programming language/environment
  - Java/Javascript/Python/…
  - Installation: https://cucumber.io/
- React: `https://github.com/Arquisoft/lomap_0`
  - jest-cucumber: Module to define user stories in Gherkin
    - And convert them to executable tests by Jest

```
$ npm install --save-dev puppeteer jest-cucumber
```

  - jest-puppeteer. Module to run the tests in a browser
    - It could be configured to use Selenium.

```
$ npm install --save-dev puppeteer jest-puppeteer
```

  - expect-puppeteer: Module with high level selectors for e2e tests

```
$ npm install --save-dev expect-puppeteer
```

BDD

- User Story example using Node.js

Feature: Registering a new user

Feature

Scenario: The user is not registered in the site
  Given An unregistered user
  When I fill the data in the form and press submit
  Then A welcome message should be shown in the screen

Scenario

Scenario: The user is already registered in the site
  Given An already registered user
  When I fill the data in the form and press submit
  Then An error message should be shown in the screen

Scenario

e2e/features/register-form.feature

BDD

webapp/e2e/steps/register-form.steps.js

```
test('The user is not registered in the site', ({given,when,then}) => {

  let email:string;
  let username:string;

  given('An unregistered user', () => {
    email = "newuser@test.com"
    username = "newuser"
  });

  when('I fill the data in the form and press submit', async () => {
    await expect(page).toMatch('Hi, ASW students')
    await expect(page).toFillForm('form[name="register"]', {
      username: username,
      email: email,
    })
    await expect(page).toClick('button', { text: 'Accept' })
  });

  then('A confirmation message should be shown in the screen', async () => {
    await expect(page).toMatch('You have been registered in the system!')
  });
})
```

BDD [Configuration]

- e2e/jest-config.js
  - Configure jest to execute the tests in the E2E folder
  - Tells jest the name pattern of the test files (note that they do not have a default name, so by default, they will not be found)
  - Hint: you can use the **testTimeout** option if your tests take longer than 10s (default).

```
export default {
    transform: {
        "^.+\\.tsx?$": "ts-jest"
    },
    testMatch: ["**/steps/*.ts"],
    moduleFileExtensions: ["ts", "tsx", "js", "jsx", "json", "node"],
    preset: "jest-puppeteer",
}
```

BDD [Browser Configuration]

- register-form.steps.js (beforeAll)
  - Configures how to launch the browser to perform the tests
  - We use **puppeteer** for this task
  - Can be also configured with other browsers.
  - We use **headless=true** (by default) to run the tests in the CI system but we can change it to false to run them locally.
  - The **slowMo** parameter is useful to slowdown the tests and see what is happening

```
beforeAll(async () => {
  browser = process.env.GITHUB_ACTIONS
    ? await puppeteer.launch()
    : await puppeteer.launch({ headless: false, slowMo: 50 });
  page = await browser.newPage();

  await page
    .goto("http://localhost:3000", {
      waitUntil: "networkidle0",
    })
    .catch(() => {});
});
```

BDD [Configuration – Launching the system]

- webapp/package.json
  - Configures how to launch the system
    - For testing this app we need the restapi and the webapp
  - We use the start-server-and-test library
    - This library accepts pairs of parameters (run command, url to test)
  - In order to execute the E2E tests we must build the production version with **npm run build** and then run **npm run test:e2e**

```
"test:e2e": "start-server-and-test
  npm --prefix ../restapi start http://localhost:5000/api/users/list
  prod 3000       # Equivalent to npm run prod and http://localhost:3000
  cd e2e && jest" # Runs the tests
```

BDD

- Result

```
PASS  steps/register-form.steps.ts (10.655 s)
  Registering a new user
    ✓ The user is not registered in the site (4533 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        10.772 s, estimated 15 s
```

Other example cucumber + selenium + java Spring boot from previous years:

https://github.com/arquisoft/votingSystem0

Browser-based tests

- Browser automation
  - https://cucumber.io/docs/reference/browser-automation
- Several systems
  - Selenium WebDriver - http://docs.seleniumhq.org/
  - Capybara - http://teamcapybara.github.io/capybara/
  - Watir - https://watir.com/
  - Serenity - http://serenity-bdd.info

Selenium

- Selenium IDE: Allows to record actions
  - Firefox and Chrome plugins
- Generates code to execute those actions
- Travis configuration
  - https://lkrnac.net/blog/2016/01/run-selenium-tests-on-travisci/

Bibliography and links

- User Story Mapping by Jeff Patton
  - **User Story Mapping: Discover the Whole Story, Build the Right Product, 1st Edition**
    https://www.amazon.com/User-Story-Mapping-Discover-Product/dp/1491904909
- User stories
  - Scrum. Historias de Usuario (Fernando Llopis, Universidad de Alicante)
    https://fernandollopis.dlsi.ua.es/?p=39
  - User stories with Gherkin and Cucumber (Michael Williams)
    https://medium.com/@mvwi/story-writing-with-gherkin-and-cucumber-1878124c284c
  - Cucumber 10 minutes tutorial (JS)
    https://docs.cucumber.io/guides/10-minute-tutorial/
- Browser based tests
  - Automated UI Testing with Selenium and JavaScript
    https://itnext.io/automated-ui-testing-with-selenium-and-javascript-90bbe7ca13a3