Behavior-Driven Development & Webdriverio

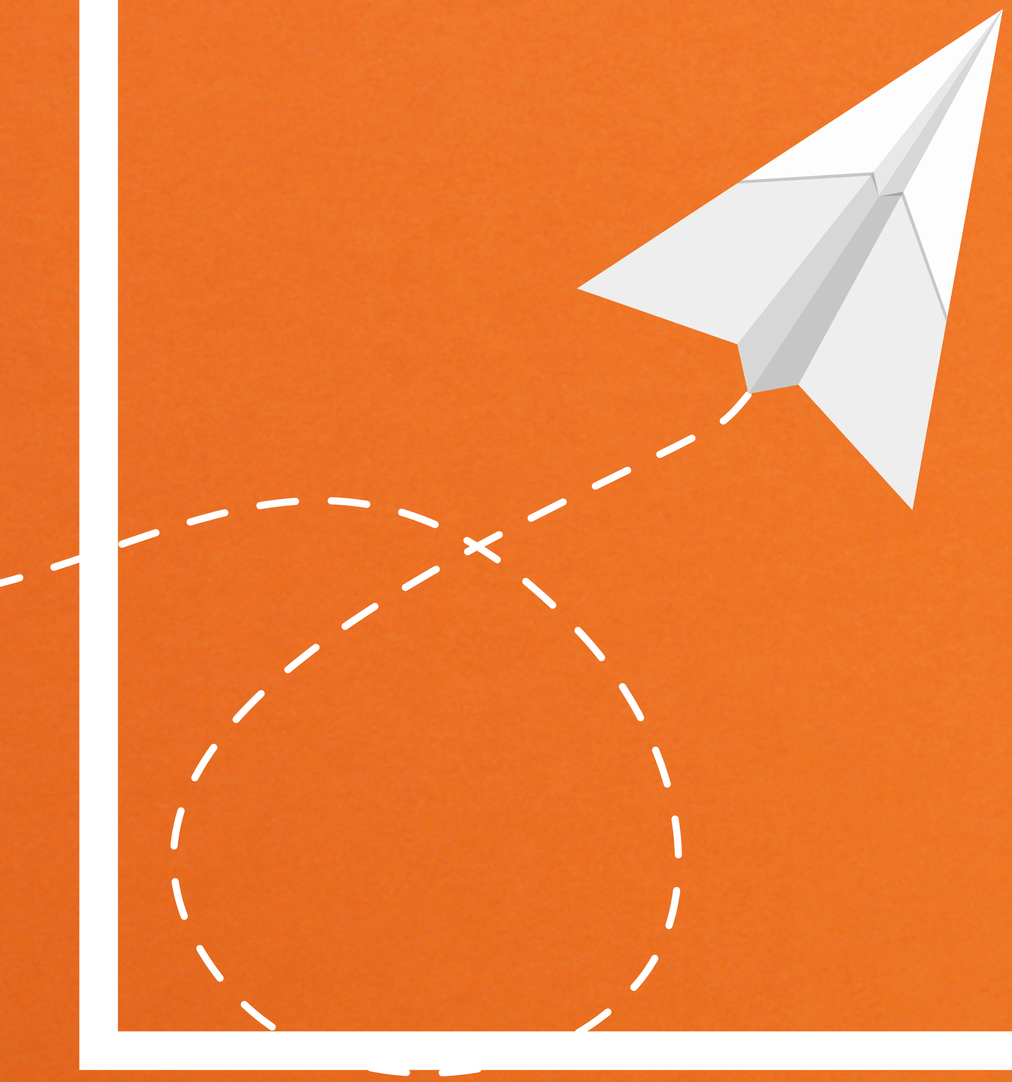
Day 1

NICK KRAMER

Hello!

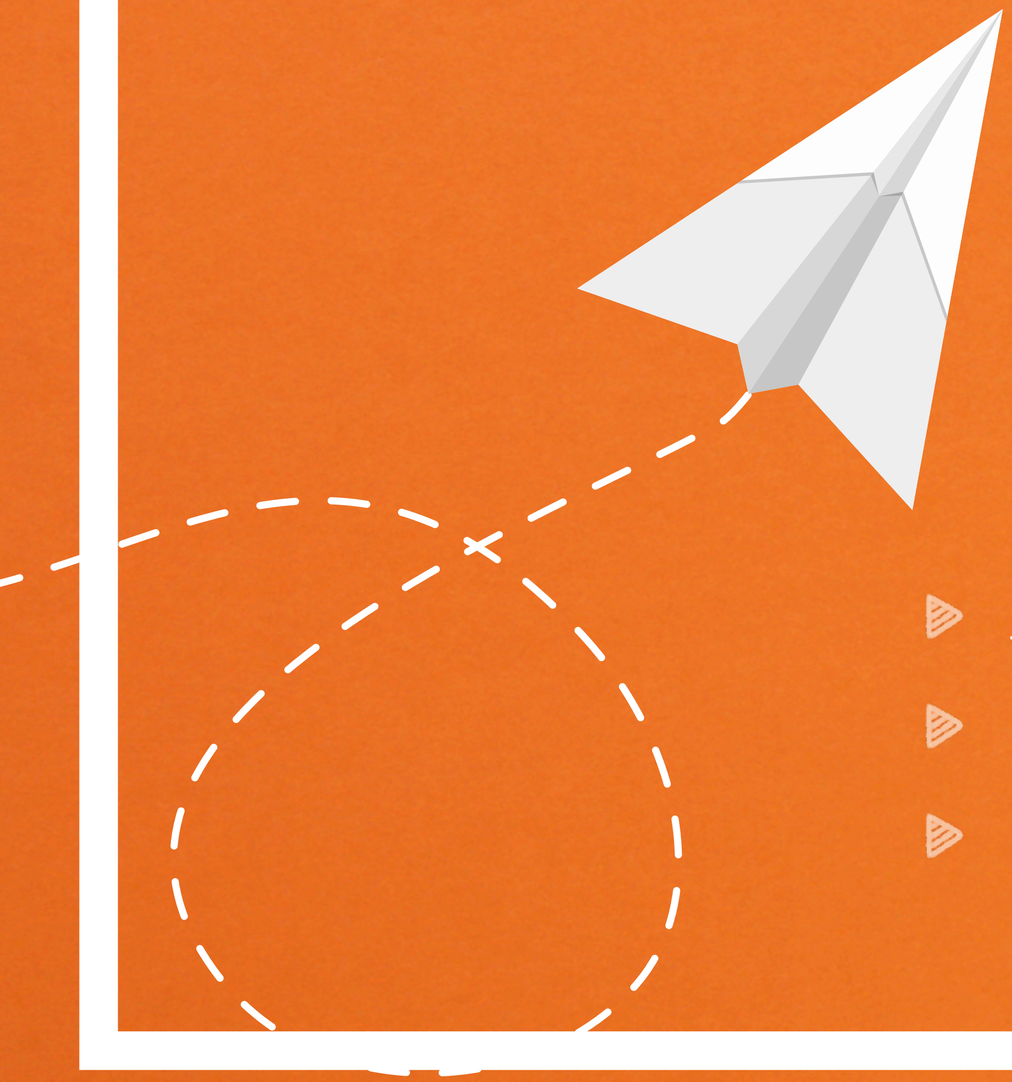
FATHER, FITNESS JUNKY, WANT-TO-BE GUITARIST

Nick.Kramer@webagesolutions.com



ROADMAP DAY 1

- TDD & BDD
- SELENIUM VS WEBDRIVERIO
- BEHAVIOR-DRIVEN DEVELOPMENT
- CUCUMBER & GHERKIN
- ACCEPTANCE CRITERIA
- INTRODUCTION TO VS CODE
- FIRST WEBDRIVERIO TESTS



LOGISTICS

- NICK.KRAMER@WEBAGESOLUTIONS.COM
- BREAKS
- LUNCH

ACTIVITY

THE GAME

Each team will have 10 minutes to draw the picture described on the board. You can use any means necessary to draw the picture-be creative

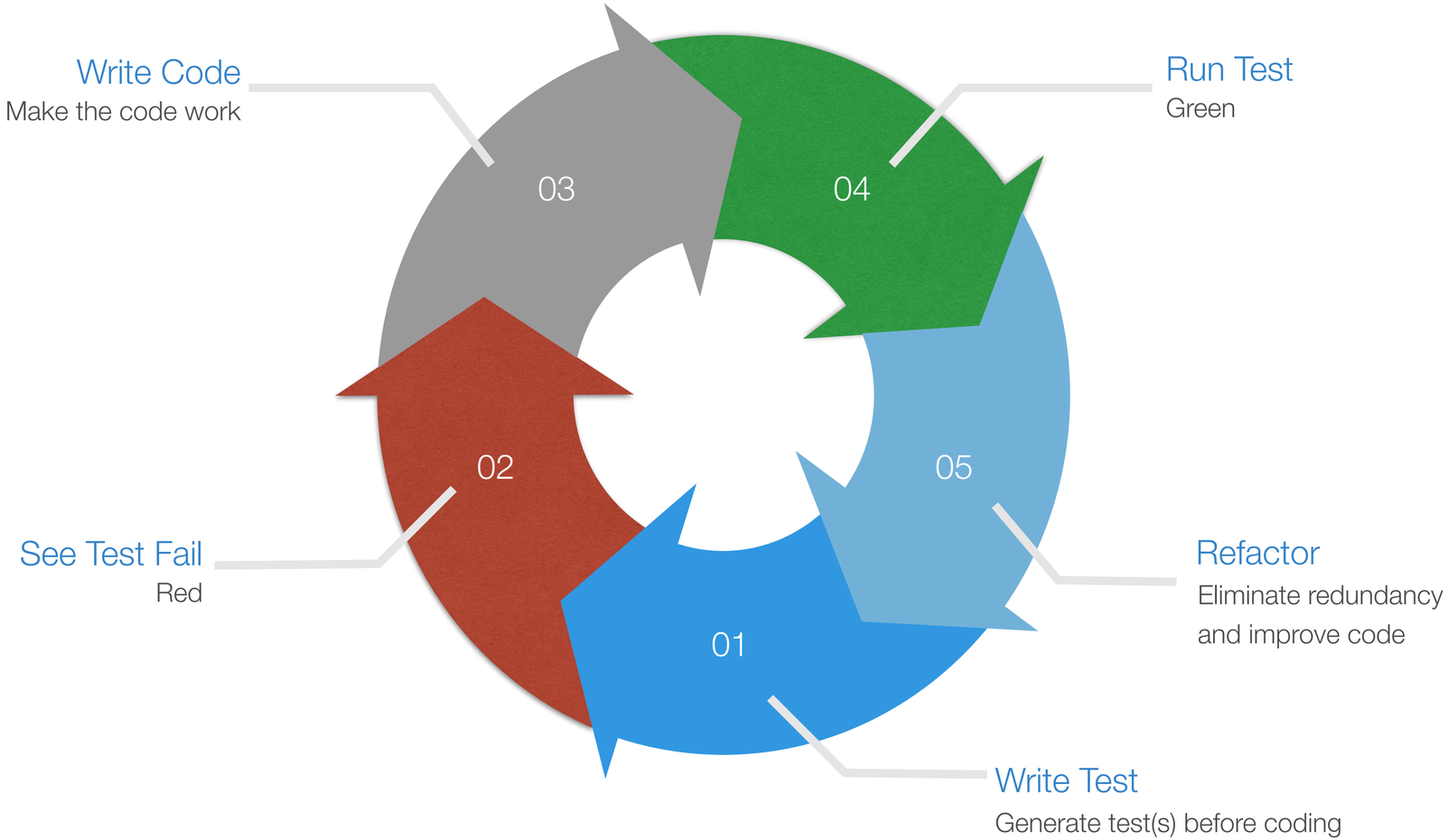
- Drawings
- Clip Art
- Shapes
- Be Creative

TDD Recap

Test-Driven Development (TDD): This approach requires developers to write failing tests before writing the code to pass those tests.

Typically broken up into five stages:

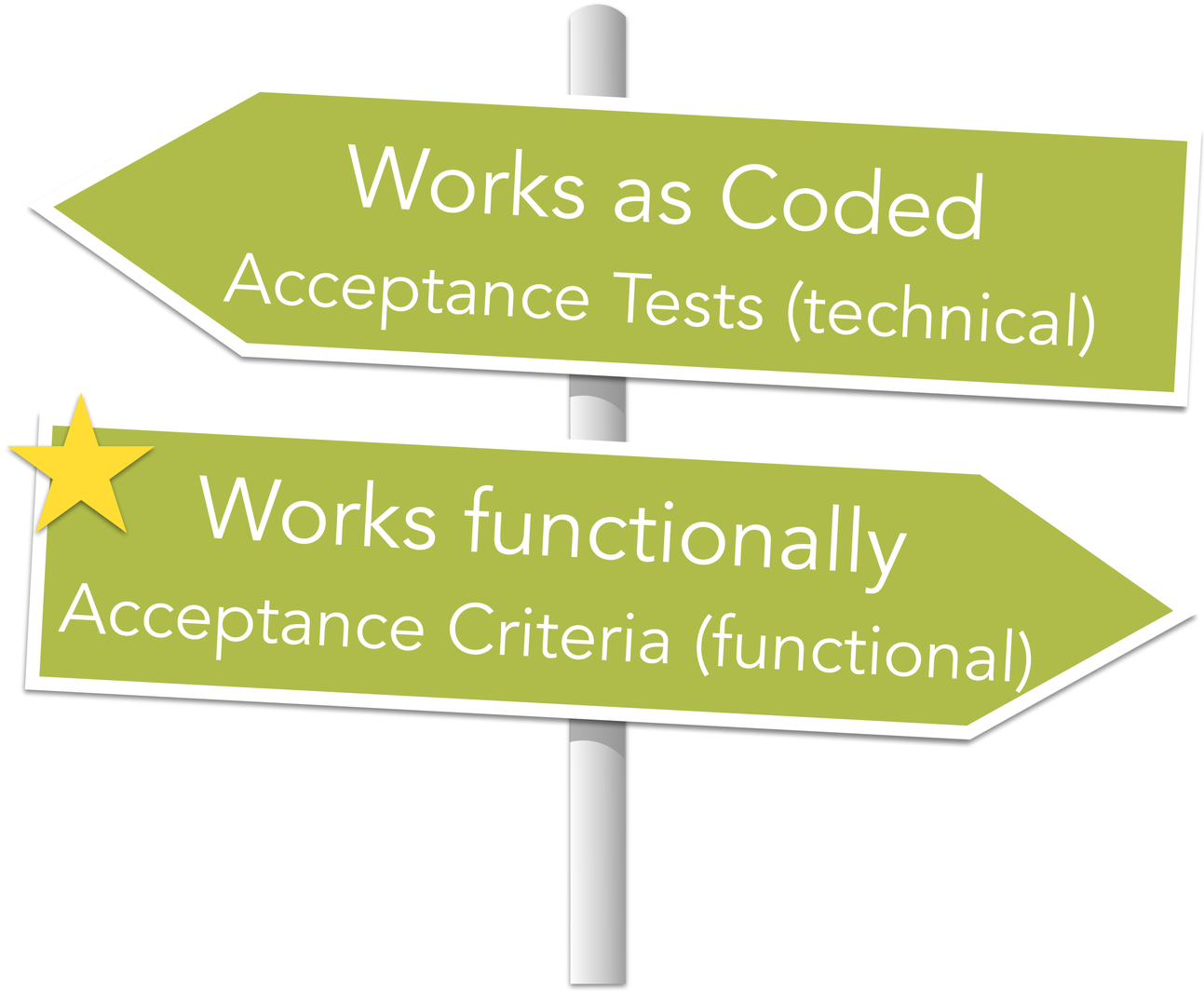
1. First, the developer writes some tests.
2. Next, the developer runs those tests, which fail (by design), because none of those features are implemented yet.
3. Next, the developer implements those tests in code, but *just enough* to pass the tests, thus validating the tests.
4. Now the developer fleshes the implementation out to be complete, using the tests to ensure the code remains on track.
5. Finally, the developer refactors their code, adds comments, cleans it up, and can do so with confidence that the tests provide an objective regression meter to identify if anything breaks in the process.



Test Driven Development (TDD)

A software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests

Acceptance Criteria



From TDD to BDD

BDD borrows much of its initial philosophy from TDD and utilizes a test-centric approach as an essential thread throughout the methodology. However, it moves beyond TDD to rethink the testing approach.

TDD

For each unit of software…

- define a test set for the unit first
- then implement the unit
- finally, verify that the implementation of the unit makes the tests succeed

BDD

Behavior-driven development specifies that tests of any software unit should be specified in terms of the ***desired behavior of the unit***.

What is the “desired behavior” in question? It’s the business requirements that spell out a valuable business outcome.

This is referred to as BDD being an "outside-in" activity within BDD practice.

one 80 SERVICES
one80services.com

TDD vs BDD CHEATSHEET

Unit Tests are the building blocks of **Test Driven Development**		Behavioural Tests are the building blocks of **Behaviour Driven Development**
TDD is an "inside-out" process. The focus is on **quality**.	FOCUS	BDD is an "outside-in" process. The focus is on **value**.
Who writes the tests? The tests are written at the same time as the code. So it is the Developer that writes the tests. **Who reads the tests?** Developers. And Testers. Its unlikely that anyone else will read them.	READER/WRITER	**Who writes the tests?** The plain English format means that the tests can be written by the person that understands the customer best: the Product Owner. **Who reads the tests?** Almost anyone: Developers, Testers, Stakeholders, Business Owner, Product Owner.
When I test a spark plug in isolation – and the test fails – I know that spark plug is faulty. When a Unit Test fails, you know exactly what has failed.	SPECIFICITY	When my motorbike doesn't start, I know that something is wrong. But I don't know what is wrong.
A Unit Test is a test of a component in isolation. This may require mocking of (potentially slow) external dependencies. As a result, Unit Tests are super-quick to run.	SPEED	Behavioural Tests are tests of the system as a whole. The system must be "put into a known state" before each and every test. Not particularly difficult to do... but not quick either.
The higher the level of code coverage, the better you sleep at night. But 100% code coverage is rare/difficult to achieve where Unit Test are concerned.	COVERAGE	Code coverage percentages for Behavioural Tests tend to be high. 100% code coverage is not uncommon.
Any change to the functionality of a system will require a change to one or more Unit Tests. (Remember: the tests change first!)	MAINTAINABILITY	Not all functional changes impact the external behaviour; the high level nature of Behavioural Tests mean that they change infrequently.
Unit Tests are highly specific to the code that they cover; they're intertwined. Unit Tests are not portable.	PORTABILITY	Behavioural Tests are not coupled to code. Without changing a single test you could: •rewrite your application in a different programming language; •refactor your monolithic application into a set of microservices.

Context

- Given the card is valid
- And the account is in credit
- And the dispenser contains cash

Event

- When the customer requests cash

Outcomes

- Then the account is debited
- And the cash is dispensed

WEB AGE solutions inc

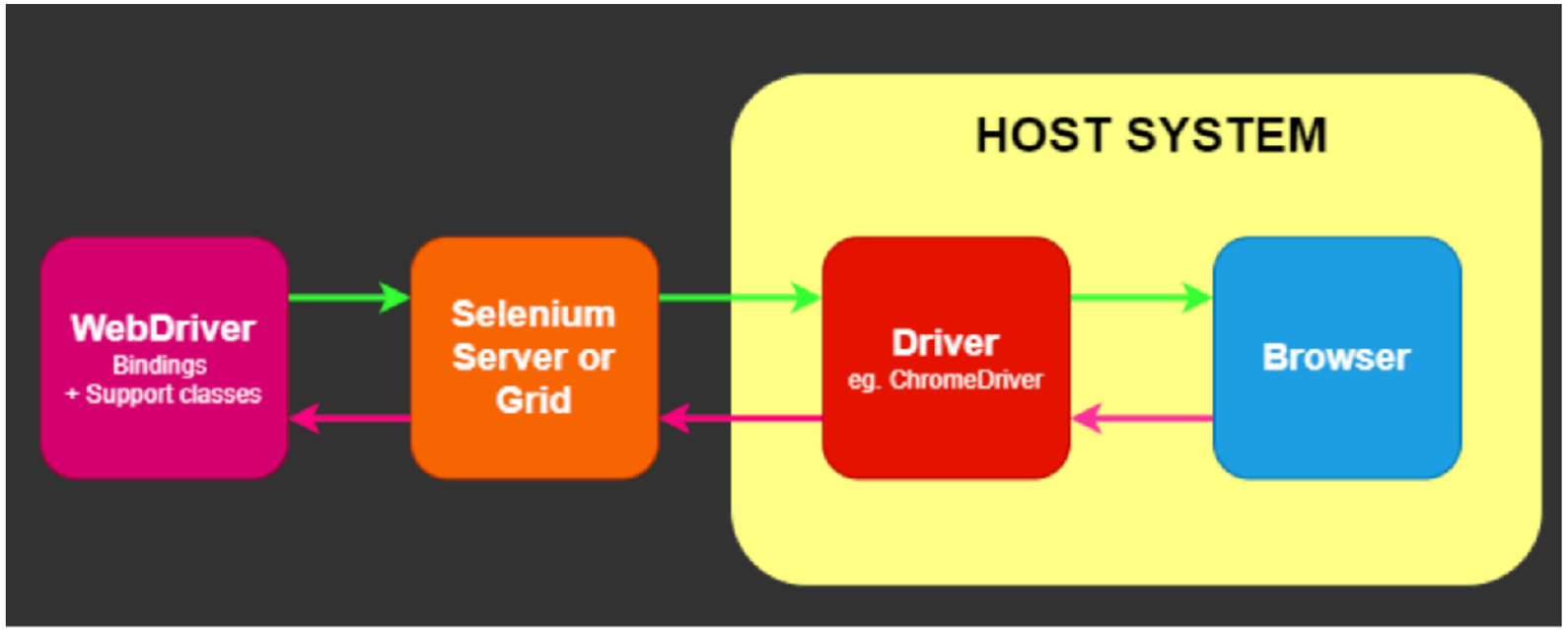
SELENIUM & WEBDRIVERIO

Selenium

- Selenium is a suite of tools and an open-source project that automates web browsers. It allows you to write programs to create browser-based tests, which you can use with other software

Selenium

- Selenium uses WebDriver protocols
- Supporting Classes written in: Ruby, Java, Python, c# …



Selenium -Pros

Selenium is a suite of tools and an open-source project that automates web browsers. It allows you to write programs to create browser-based tests, which you can use with other software

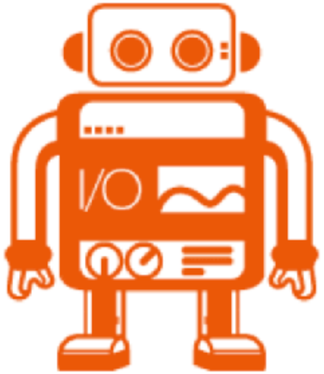
Selenium has an API for programming languages like Java, Ruby, C#, and Python in addition to JavaScript, and it's easier to debug if you're fluent in the language.

It's stable, reliable, and well-known in the industry. And it's suitable for testing out of the box and quick to set up initially.

Developers generally know the base language!

Selenium -Cons

- Selenium can be challenging to debug. Debugging requires using Chrome Developer Tools, Firebug, or another browser plug-in. Sometimes it's challenging to identify which piece of code caused an issue.
- It's also not as performant as WebdriverIO. WebdriverIO takes advantage of Node.js asynchronous calls, and they built it upon Node.js libraries.
- It can be complicated to set up and maintain
- Test on one browser at a time



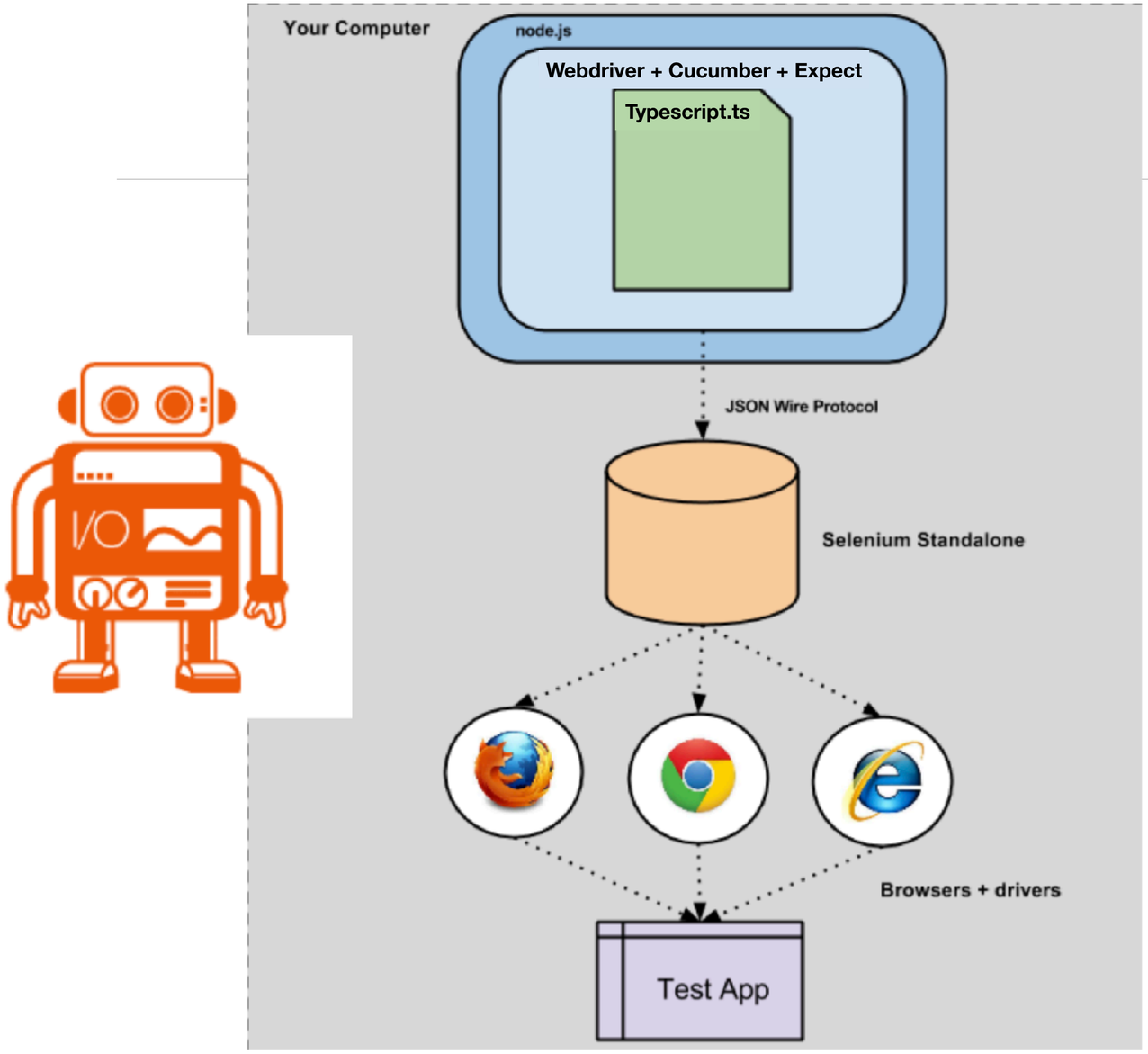
WebdriverIO

WebdriverIO provides an interface between your application and the browser, so you only have to write tests using plain JavaScript. This means you can test all your app's functionalities just as quickly using your favorite programming language.

WebdriverIO is a flexible test automation framework that uses WebDriver protocol to automate browsers. It supports multiple browsers, including:

1. Google Chrome
2. Mozilla Firefox
3. Microsoft Edge
4. Safari
5. Internet Explorer 11 (although it's deprecated)

Additionally, WebdriverIO supports mobile platforms through Appium, which allows you to automate iOS and Android apps.



WebdriverIO -Pros

WebdriverIO has several pros, including the following:

- It is fully extensible and is written to be as flexible and framework agnostic as possible. It can be applied in any context and serves not only the purpose of testing.
- Easy to use: The API is close to Selenium's and has a simple way of constructing objects. The API was also designed so that you can write tests in a more readable way, which is better for most of the developers that don't know JavaScript but need to build tests with JavaScript.
- It has a command line interface (wdio) which makes test configuration as easy and simple as possible so that a non-programmer can configure the setup.
- Allows multi-use: WebdriverIO has a Node package that makes it easy to use in all your JavaScript projects. You can use the framework on your web, mobile, desktop, or API endpoints.
- Can test on multiple browsers and native apps at the same time
- It has good support, an enthusiastic developer community, and end users, giving it an edge over Selenium.
- It can be used with webdriver css to compare css stylings of an element in the webpage.\
- Faster

WebdriverIO -Cons

- It doesn't use features like classes or modules,
- Lack of infrastructure: The API is very simple and doesn't have complex facilities like watchers, listeners, and so forth.
- It's hard to know what some of the more complicated functions and classes do. With WebdriverIO, you're limited to writing tests using TypeScript or JavaScript, and because of that, as a developer and tester, you need to have at least a basic understanding of JavaScript.

ROLES & RESPONSIBILITIES

AGILE\BDD FRAMEWORK

one 80 SERVICES
one80services.com

Release Planning

Select top product backlog items

- Define iteration length
- Create & size user stories
- Sequence user stories

Sprint Planning

- Review product backlog
- Estimate sprint backlog
- Commit work for the sprint
- User story breakdown into tasks
- Task estimation

Backlog Refinement

Add details, estimate and prioritize the Product Backlog

Daily Scrum

Each team member answers these questions:

- What have I accomplished since our last meeting?
- Is there any impediment?
- What am I planning to accomplish before our next meeting?

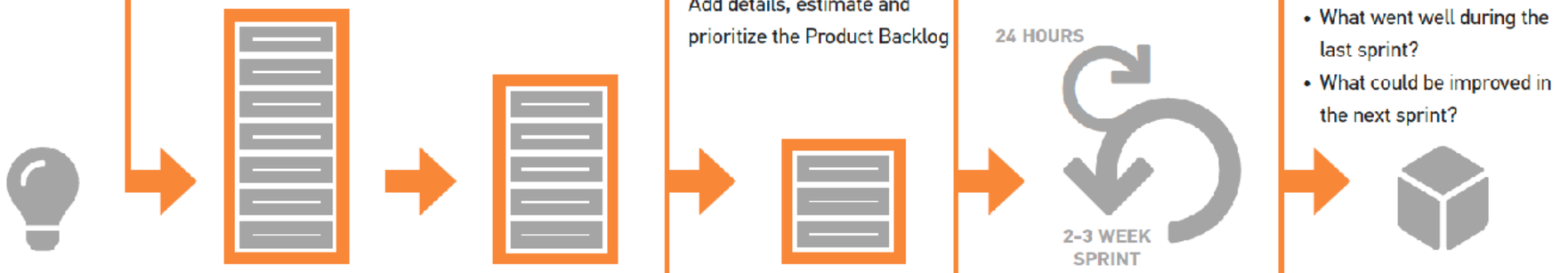
Sprint Review

- Review work completed
- Demo the work
- Review project metrics
- Priority review for next iteration/sprint

Sprint Retrospective

Inspect and Adapt:

- What went well during the last sprint?
- What could be improved in the next sprint?



Idea

Product Backlog
Prioritized items desired by the customer.

Release Backlog
Prioritized items required for this current release.

Sprint Backlog
Backlog items selected for sprint, sliced into tasks, and estimated by the team.

Sprint
Team self-organized to complete the sprint objective and goals

Potentially Shippable Product

Roles

WEB AGE solutions inc.

Roles

PRODUCT OWNER

Product Owner: Responsible for maximizing the value of the product.

SCRUM MASTER

Scrum Master: Leads and Coaches the team on Agile and BDD best practices.

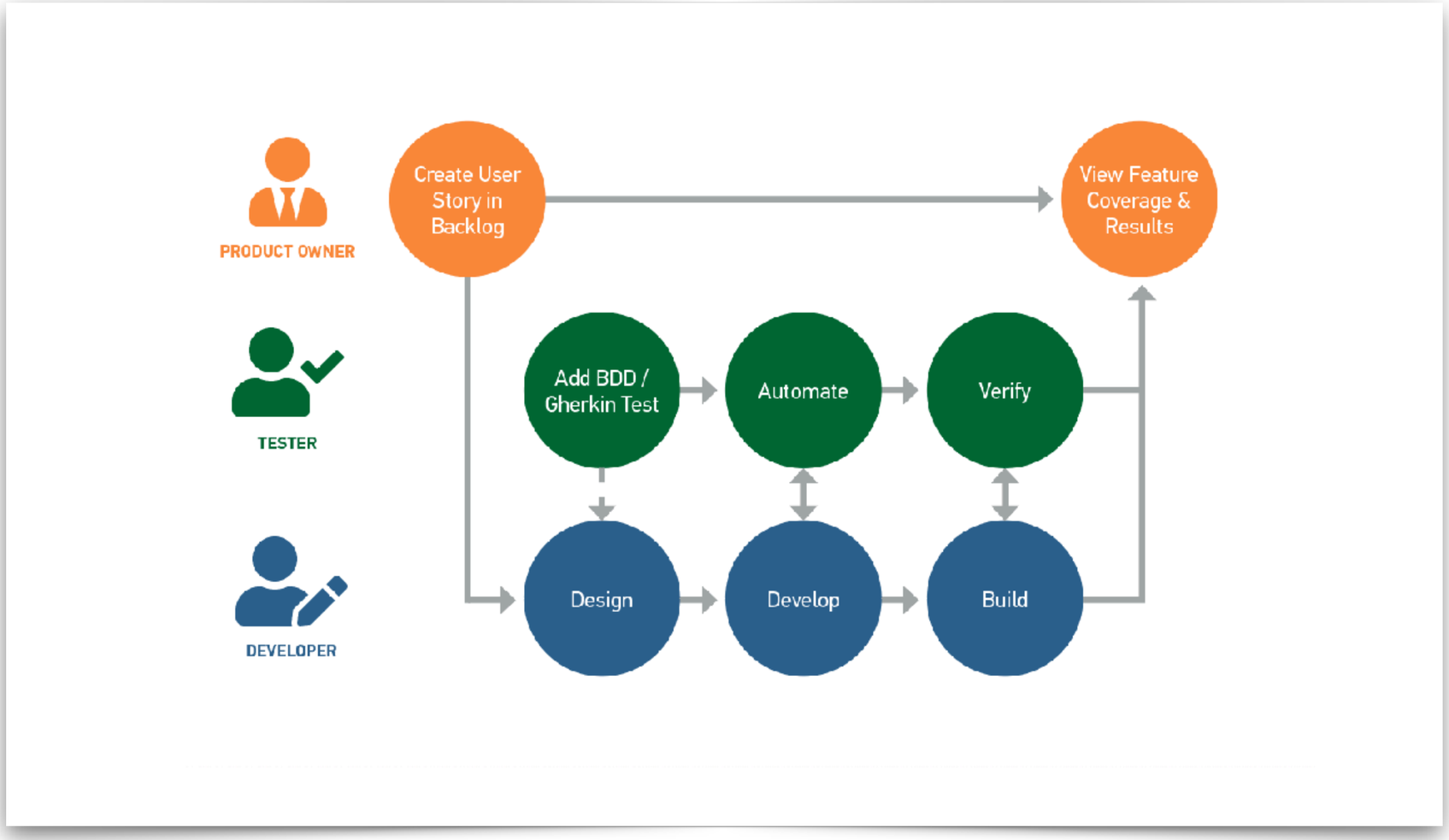
DEVELOPER

Developer: Person(s) responsibility for code delivery.

TESTER

Tester: Person(s) responsibility for ensuring that delivered code adheres to requirements.

Collaboration between the 3 Amigos

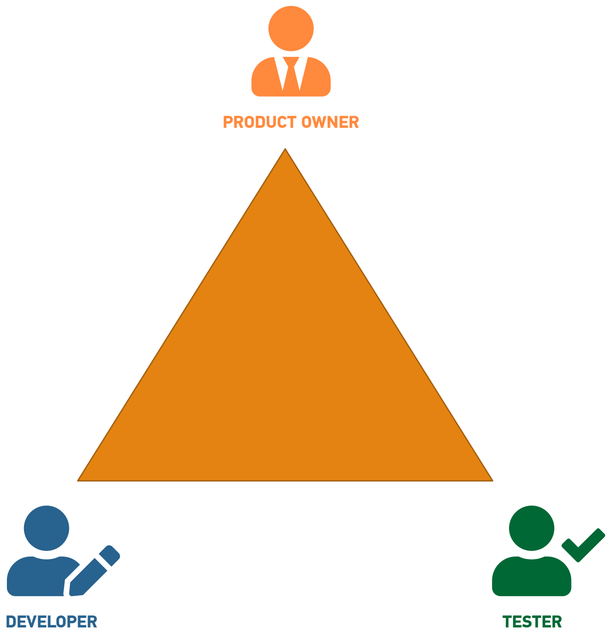


Share your experiences

Do you currently have a "**3 amigos**" step in your process?

How have you seen the interplay of technical, business, and quality perspectives on a project?

What happens when one of these is non-existent
or perhaps excessively dominant?

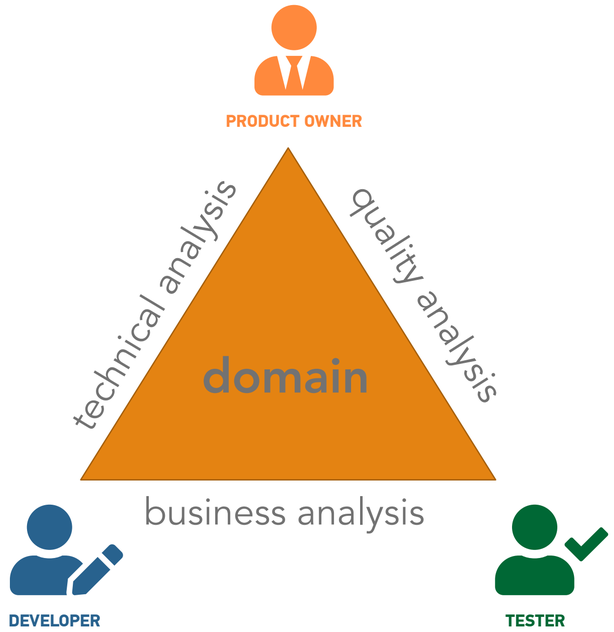


The 3 Amigos

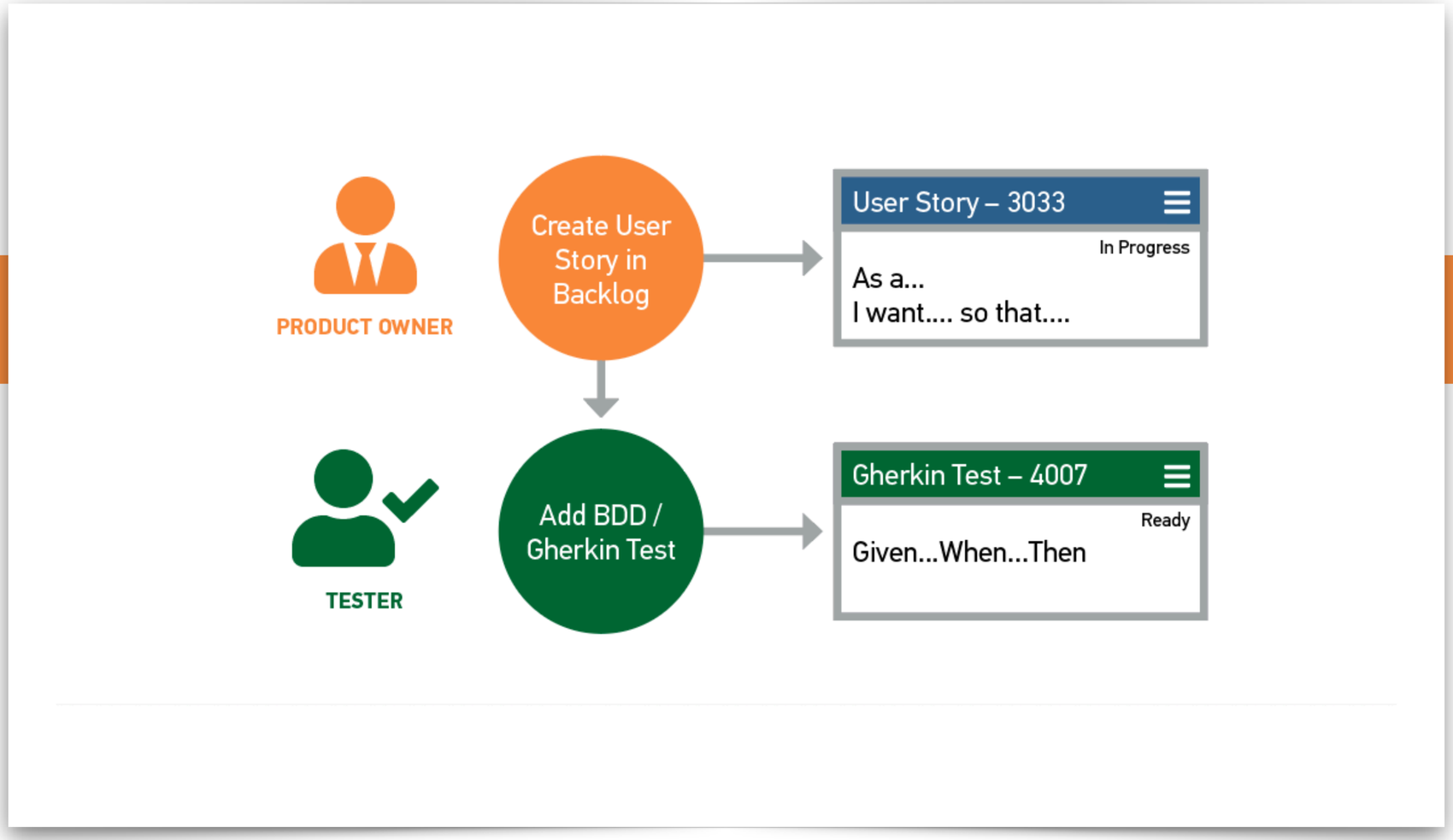
One of the biggest (and most common mistakes) is to assume that writing out the scenarios is the most import part of the process and that this responsibility can or should be given to a single person.

BDD is intended to be a **highly collaborative process**. Talking in examples requires multiple inputs and varied perspectives in order to be efficient.

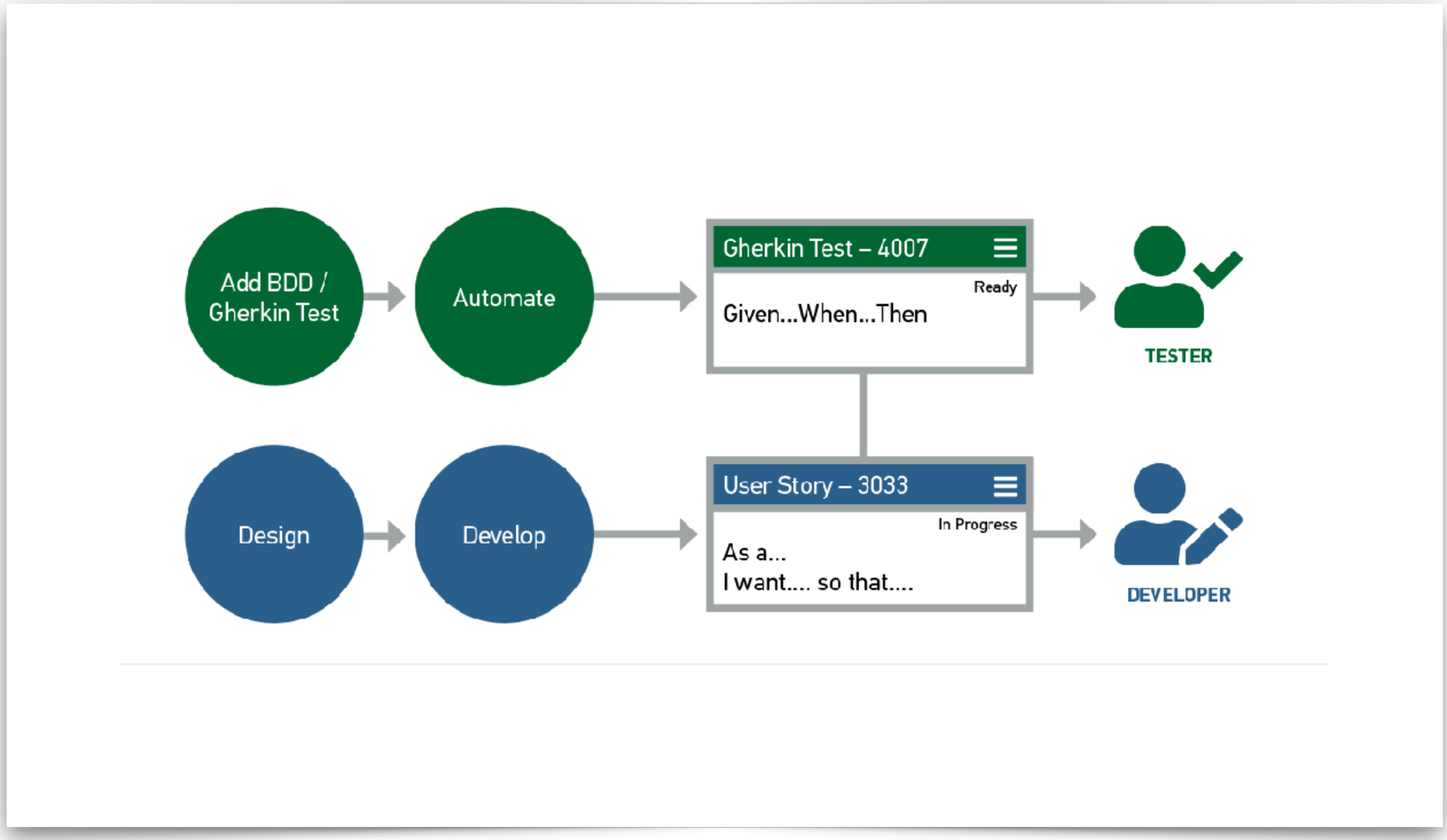
In order to effectively strike the right balance of breadth and depth, business and technical elements, you need three fundamental perspectives.



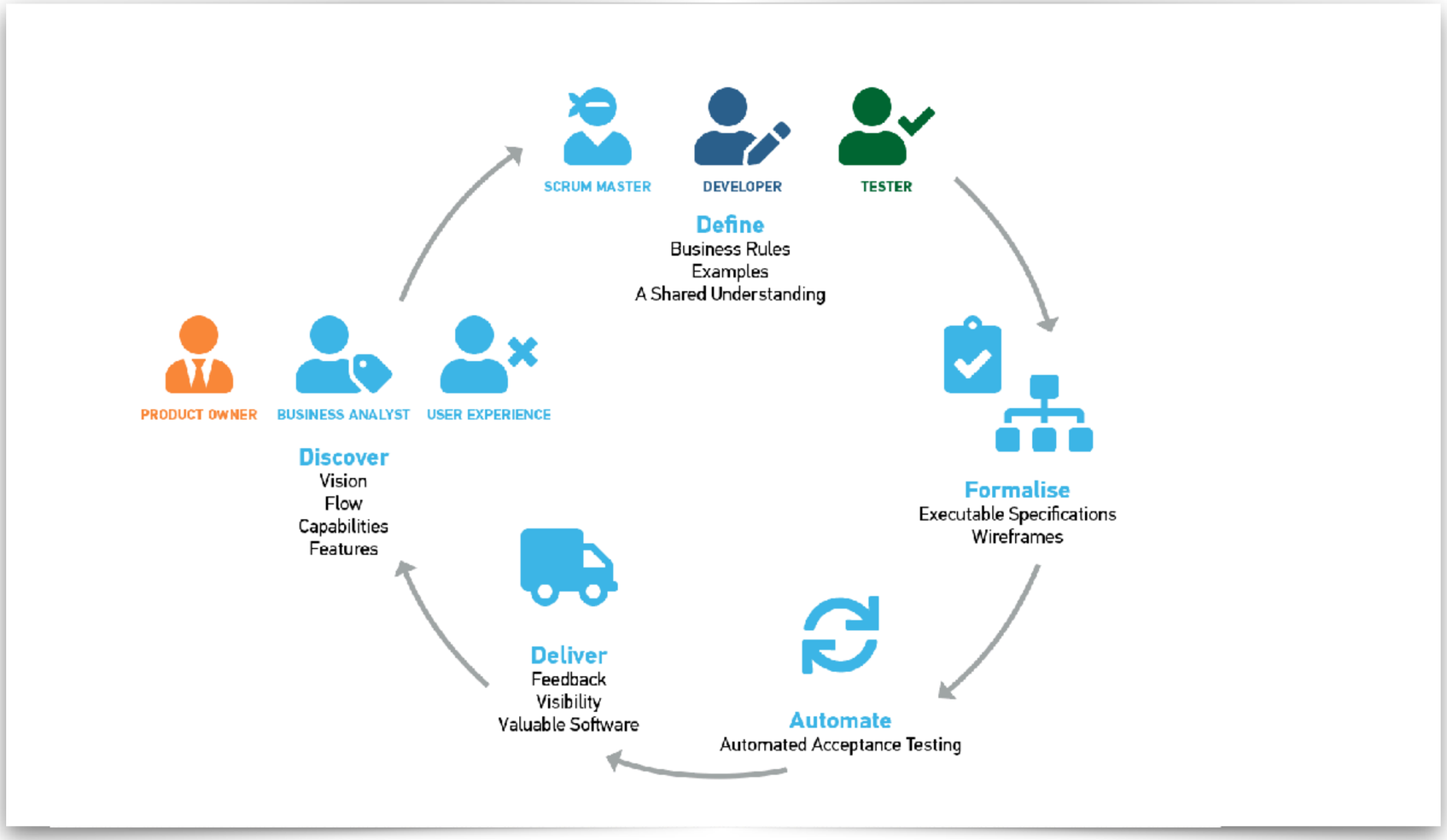
Collaboration between Product Owner and Tester



Collaboration between Tester and Developer



Enterprise BDD Lifecycle



ACCEPTANCE CRITIERIA

Acceptance Criteria

SIMPLE BDD

Each team will have 10 minutes to write down the direction on how to draw the picture provided.

- Be as specific as necessary.
- Provide step-by-step instructions

At the completion of the 10 min, each team will read their instructions while the remaining teams draw the picture from the directions.

Note: each needs a team representative who will read the instructions to the team as a whole.

Behavior-Driven Development

BDD aims to rethink how we define **acceptance tests**.

- Rather than writing a test for everything (as is often the case with classic TDD), we apply the 80/20 rule and focus on the significant behavior the business has indicated as important.
- Acceptance tests now have a real structure.
  - Given [initial context]
  - When [event occurs]
  - Then [verifiable outcomes]
- This gives us clarity regarding what we need to implement and a target to hit as we use these tests to demonstrate our capabilities during the demo at the end of the sprint.

The ability to read tests like a sentence is an important cognitive shift which enables you to *naturally* write better and more comprehensive tests.

Acceptance Criteria

Why?

When clearly stated, they:

- Define functional requirements at the beginning of each iteration
- Create a fast version of documented functionality
- Synchronize vision between developers and testers
- AC helps with the estimating process
- When written correctly can be automated (Selenium, Cucumber, Specflow …)

Note: All team members help write Acceptance Criteria; however, they are generally written\approved by the Product Owner and Tester

Driving Expected Behavior

Traditionally we have to ***interpret*** the business's needs into tests. Cucumber is the tool we will be using to execute BDD.

Gherkin provides the language to capture expectations with natural language.

Through concrete scenarios, we codify the expected behavior and build out validation tests to demonstrate that we have achieved the target.

These acceptance criteria answer the fundamental question:

Did the software or process produce the expected behavior the business is looking to produce?

Acceptance Criteria

Gherkin Format

Given-When-Then

"Given" – **Context**: describes the initial context for the example
"When" – **Event**: describes the triggering performed by an actor
"Then" – **Outcome**: describes the expected outcome (validated)

This aims to mirror the natural way people describe expected behavior. When a stakeholder is asked to provide examples of what they expect for a particular capability, they naturally say something like:

"When I do X, then Y should happen."

Acceptance Criteria

Gherkin

Gherkin statements are written as **real** examples that can be executed.

Types of Acceptance Criteria:

- Positive
- Negative
- Happy Path
- Edge Case

Given

- Describes the initial condition(s) in place prior to the event occurring
- Within the testing context, the given statement(s) put the system into a known state to support the test
- If this were a use case, givens would be pre-conditions
- Avoid talking about workflow or user actions

What would be a couple of examples of valid **Given** statements?

When

- Within the testing context, the when statement interacts with the established pre-conditions from the Given statements
- Describes the actions or steps to completed in the scenario
- **Real** actions performed by a user

What would be a couple of examples of valid When statements?

Then

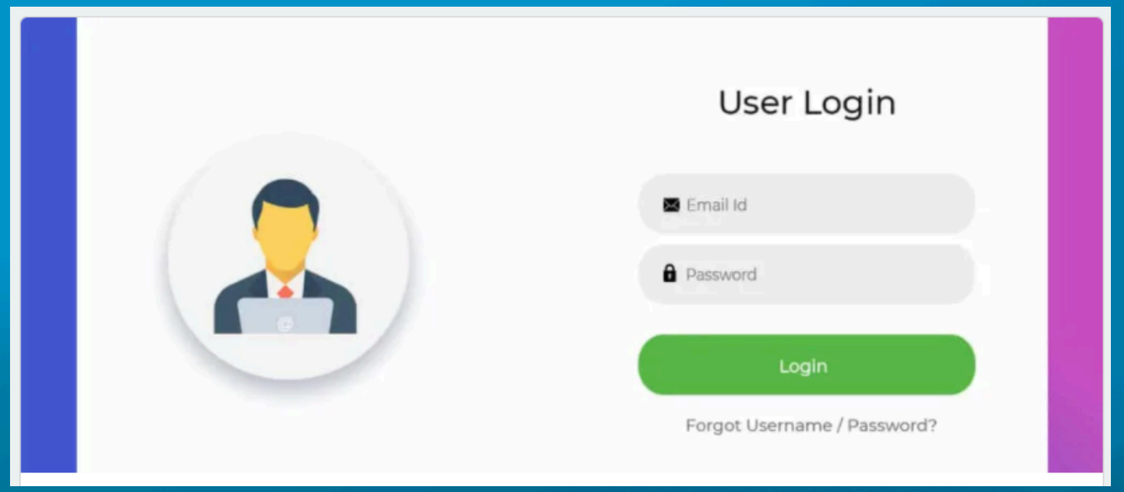
- Describes the required outcome
- Must be objective and measurable (true / false, on / off, equal / unequal)
- Verify Given + When is or is not in the output
- Value should be observable and testable

What would be a couple of examples of valid **Then** statements?

ACTIVITY

ACCEPTANCE CRITERIA

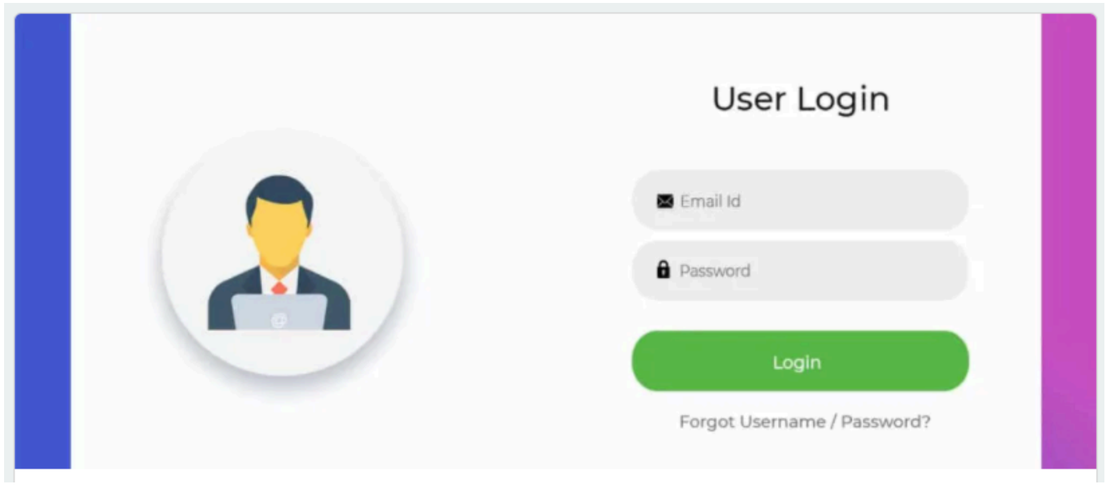
Write an Acceptance Criteria using the Gherkin format for this Screen



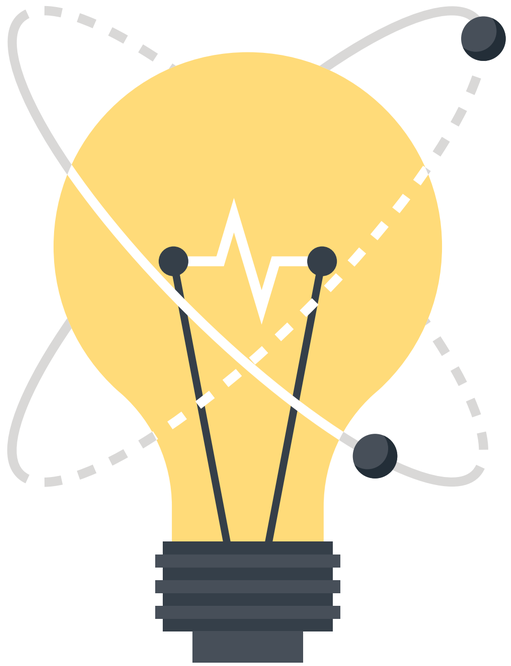
Acceptance Criteria

Example

Create one user story with multiple acceptance criteria for this screen



Lab - Acceptance Criteria



You and your business partners are real-estate professionals who need a new website. The wireframes provided have been approved by the Product Owner and UI\UX team.

As a team:

- Pick 1 of your user stories and write Acceptance Criteria

Note: Use the same cards that you created in the previous exercises

Lab - Writing Acceptance Criteria

Exercise: AC

You and your business partners are real-estate professionals who need a new website. The wireframes provided have been approved by the Product Owner and UI\UX team.

As a team review the Login Story:

- Write Acceptance Criteria for your User Stories

www.one80Training.com