# Don't write code your Users haven't asked for.

## Behavior-Driven Development (BDD)

Acceptance-Test Driven Development

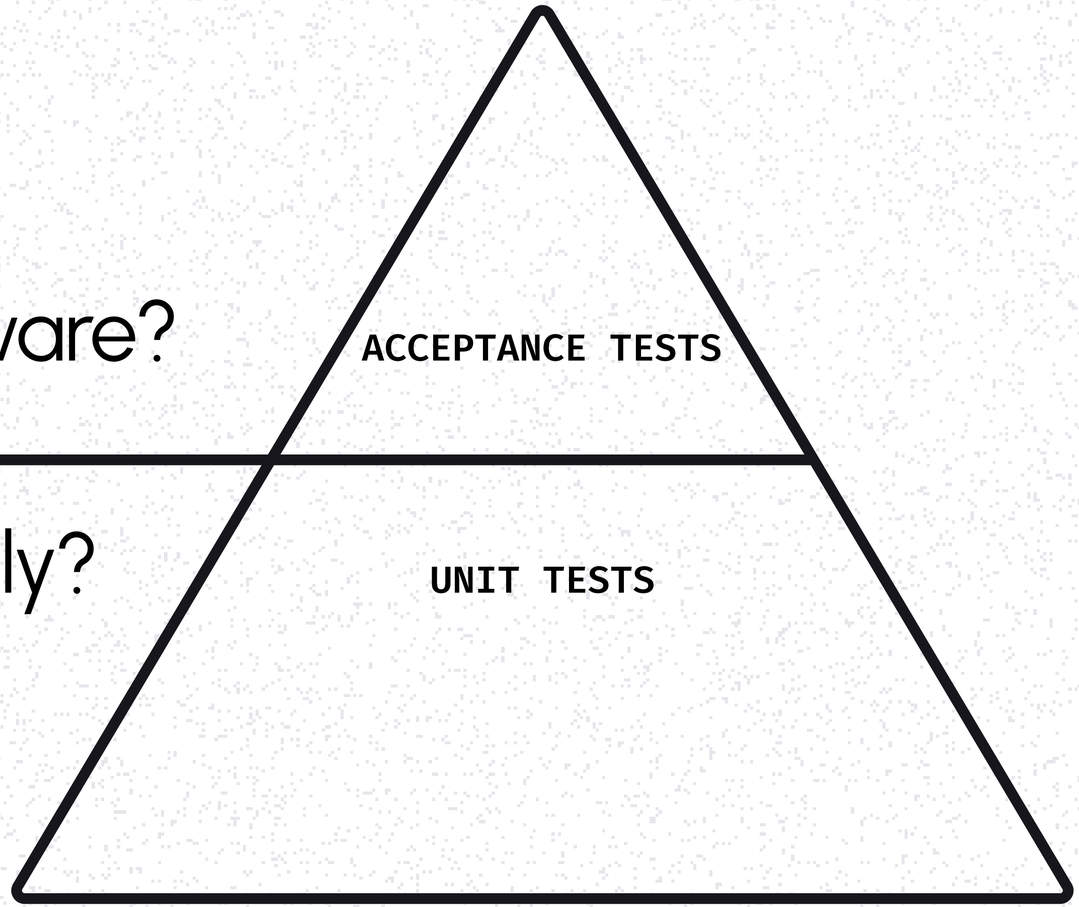
Specification by Example

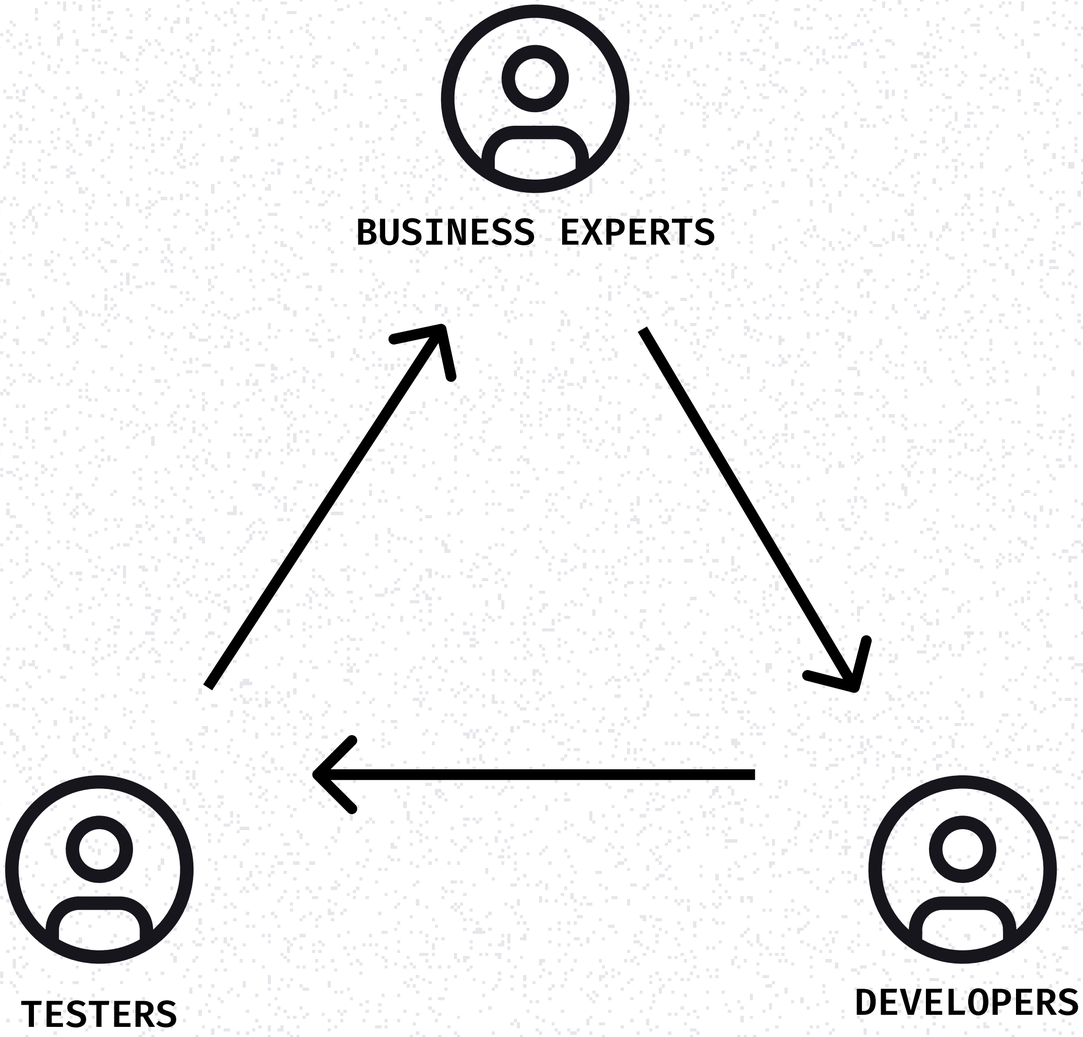
Executable Specifications

1. What is BDD?
2. How to do BDD?
3. {cucumber}

Have we built the correct software?

Have we built software correctly?



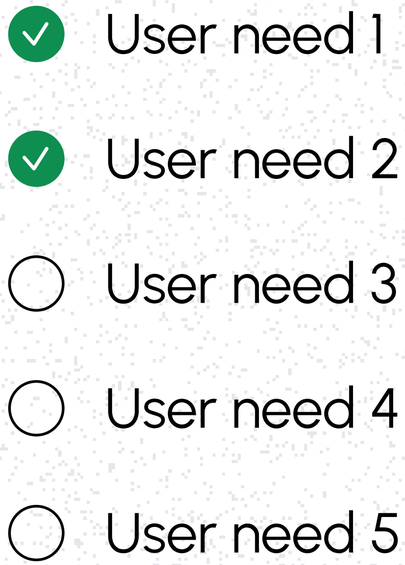

BUSINESS EXPERTS
TESTERS
DEVELOPERS

Vague Wish → User Story → Executable Specification → Tests & Code

# Automated Definition of Done

Starting from writing specifications we precisely know when all User needs are met.

We stop writing code when all specifications pass.



# Executable specifications

**Expressed in a language understood by business and developers.**

Given an available flight on “my preferred date”

When I search for flights from “London” to “Paris” on “my preferred date”

Then I am offered the flight

# Use keywords

**Separate preconditions from actions, and from expectations.**

Given an available flight on “my preferred date”

When I search for flights from “London” to “Paris” on “my preferred date”

Then I am offered the flight

# Be abstract

Don't reveal implementation details.
They should remain true for any implementation of the system: Web App, REST API, R Package, etc.

Given an available flight on "my preferred date"

~~When I click "Search" button and type London and Paris and select my preferred date~~

Then I am offered the flight

# Be abstract

**Don't reveal implementation details.**
**They should remain true for any implementation of the system: Web App, REST API, R Package, etc.**

Given an available flight on my preferred date

When I search for flights from "London" to "Paris" on "my preferred date"

Then I am offered the flight

## Bad Acceptance Criteria

- Use "flights" table in the DB
- Have 2 search fields and "Search" button
- Results are displayed in a table

## Good Specifications

Given an available flight on my preferred date

When I search for flights from "London" to "Paris" on "my preferred date"

Then I am offered the flight

# Execute them with cucumber


Given an available flight on “my preferred date”

When I search for flights from “London” to “Paris” on “my preferred date”

Then I am offered the flight

## 3 elements of {cucumber}

1. Feature files – test cases.
2. Steps implementation.
3. Hooks (optional)

tests/testthat/flight_booking.feature

```
Feature: Flight booking
  The flight booking system enables users to search for and book both direct and indirect flights.
  Users can select one-way or return trips based on available options, search by date and route,
  and receive a list of available flights.

  Scenario: Book a one-way trip
    Given an available flight on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered the flight

  Scenario: Book a one-way trip
    Given 2 available flights on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered 2 flights
```

tests/testthat/setup-steps.R

```
given("an available flight on {string}", function(date, context) {
  # Setup the environment, e.g. inject test data to a test database
})

when(
  "I search flights from {string} to {string} on {string}",
  function(from, to, date, context) {
  # Run code that searches flights with given parameters,
  # Store result in `context`
})

then("I am offered the flight", function(context) {
  # Get result from `context`, assert.
})

then("I am offered {int} flights", function(n, context) {
  # Get result from `context`, assert.
})
```

tests/testthat/setup-hooks.R

```
before(function(scenario_name, context) {
  # Create a database
  # Or run shinytest2 driver
  # Or run Plumber API
})

after(function(scenario_name, context) {
  # Clean up the database
  # Or clean up the filesystem
})
```

## Cucumber tests alongside testthat

```
TESTS/
├── TESTTHAT/
    ├── TEST-CUCUMBER.R
    ├── TEST-UNIT_TEST_1.R
    ├── TEST-UNIT_TEST_2.R
    ├── SETUP-STEPS_1.R
    ├── SETUP-STEPS_2.R
    ├── SETUP-HOOKS.R
    ├── FEATURE_1.FEATURE
    ├── FEATURE_2.FEATURE
```

## Cucumber tests in own directory

```
TESTS/
├── CUCUMBER/
    ├── SETUP-STEPS_1.R
    ├── SETUP-STEPS_2.R
    ├── SETUP-HOOKS.R
    ├── FEATURE_1.FEATURE
    ├── FEATURE_2.FEATURE
├── TESTTHAT/
    ├── TEST-UNIT_TEST_1.R
    ├── TEST-UNIT_TEST_2.R
```

Behavior-Driven Development is not about tools.

## WITHOUT {CUCUMBER}

tests/testthat/test-flight_booking.R

```
describe("Flight booking", {
  it("allows booking a trip when available", {
    available_flights("my preferred date")
    flights ← search_flights("my preferred date")
    verify_found_flights(flights, n = 1)
  })

  it("allows booking a trip when 2 available", {
    available_flights(
      "my preferred date",
      "my preferred date"
     )
    flights ← search_flights("my preferred date")
    verify_found_flights(flights, n = 2)
  })
})
```

## WITH {CUCUMBER}

tests/testthat/flight_booking.feature

```
Feature: Flight booking
  Scenario: Book a one-way trip
    Given an available flight on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered the flight

  Scenario: Book a one-way tril
    Given 2 available flights on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered 2 flights
```

+ steps implementations

## WITHOUT {CUCUMBER}

tests/testthat/test-flight_booking.R

```
describe("Flight booking", {
  it("allows booking a trip when available", {
    available_flights("my preferred date")
    flights ← search_flights("my preferred date")
    verify_found_flights(flights, n = 1)
  })

  it("allows booking a trip when 2 available", {
    available_flights(
      "my preferred date",
      "my preferred date"
     )
    flights ← search_flights("my preferred date")
    verify_found_flights(flights, n = 2)
  })
})
```

### PROS

1. No extra dependencies.
2. Easy to start with.

### CONS

1. Limited space for providing context.
2. Doesn't force you to reuse test code.
3. You need to read code to understand it.

## WITH {CUCUMBER}

tests/testthat/flight_booking.feature

```
Feature: Flight booking
  Scenario: Book a one-way trip
    Given an available flight on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered the flight

  Scenario: Book a one-way tril
    Given 2 available flights on "my preferred date"
    When I search flights from "London" to "Paris" on "my preferred date"
    Then I am offered 2 flights
```

## PROS

1. Isolation from implementation details.
2. Code reuse → maintainability.
3. Understandable by everyone.
4. Live, executable documentation.

## CONS

1. New dependency.
2. Pays-off only in a certain scale.

# Real-life examples

See how {cucumber} is used to test itself,
and how it's used to test {muttest}.

GITHUB.COM/JAKUBSOB/MUTTEST/TREE/MAIN/TESTS/ACCEPTANCE
GITHUB.COM/JAKUBSOB/CUCUMBER/TREE/MAIN/TESTS/ACCEPTANCE

# Thank you!

jakubsobolewski.com

jakubsob

Jakub Sobolewski

Scenario: Learning efficient R development

Given "article" on **jakubsobolewski.com/blog**

When I read "article"

And I apply the advice from "article"

Then change-lead-time decreases by 50%

Scenario: Reusing test patterns

Given the **jakubsobolewski.com/r-tests-gallery**

When I reuse a "testing pattern"

Then change-lead-time decreases by 50%