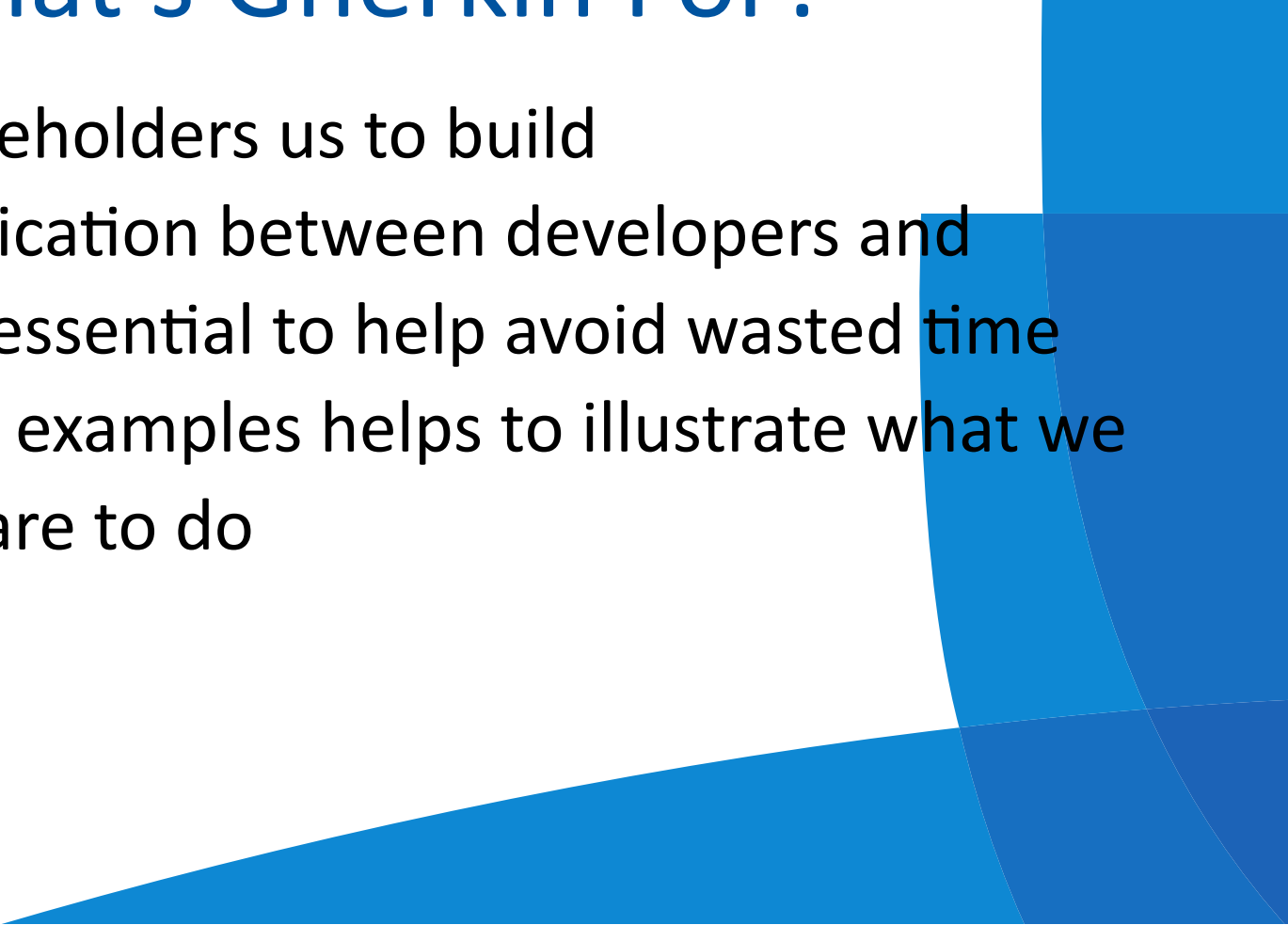


# What's Gherkin For?

- What want stakeholders us to build
  - Better communication between developers and stakeholders is essential to help avoid wasted time
  - Use of concrete examples helps to illustrate what we want the software to do
- 

# Concrete Examples

- By using real-world examples to describe the desired behaviour
  1. we stay grounded in language and terminology of the stakeholders
  2. we're speaking their language
- When talking in terms of these examples -> stakeholders can really imagine themselves using the system -> can start to give useful feedback and ideas before any code has been written

# building a credit card payment system

- A requirements: make sure users can't enter bad data
  - This can be expressed as: *Customers should be prevented from entering invalid credit card details*
  - Agile teams often call this acceptance criteria or conditions of satisfaction
  - the word acceptance is used because it tells what the system must be able to do in order for the stakeholders to find it acceptable

# building a credit card payment system

- The requirements: *Customers should be prevented from entering invalid credit card details*
- This requirements statement is useful, but:
  1. it leaves far too much room for ambiguity
  2. it leads to misunderstanding
  3. it lacks precision
- What exactly makes a set of details invalid?
- How exactly should the user be prevented from entering them?

# building a credit card payment system

- Rephrase the requirement statement in to something more precise
  - *If a customer enters a credit card number that isn't exactly 16 digits long, when they try to submit the form, it should be redisplayed with an error message advising them of the correct number of digits*
- Very clear for developer -> also stakeholder has clear view of what is going to be build
  - Stakeholders may even point out that there are other credit card types with a different number of digits -> these examples stirr the imagination

# building a credit card payment system

- Rephrasing the requirement statement in to something more precise
  - *If a customer enters a credit card number that isn't exactly 16 digits long, when they try to submit the form, it should be redisplayed with an error message advising them of the correct number of digits*
- results in an example that illustrate our requirement -> an acceptance criterion is turned into an acceptance test

# Executable Specifications

- concrete examples are easier to validate against the running system
- if expressed clearly -> the computer can check them
  - this is called automated acceptance testing
- For automated acceptance tests to be really effective:
  1. they need to be readable by the computer
  2. but also readable by our stakeholders
    - -> Gherkin



# Using Gherkin

**Feature:** Feedback when entering invalid credit card details

In user testing we've seen a lot of people who made mistakes entering their credit card. We need to be as helpful as possible here to avoid losing users at this crucial stage of the transaction.

**Background:**

**Given** I have chosen some items to buy  
**And** I am about to enter my credit card details

**Scenario:** Credit card number too short

**When** I enter a card number that's only 15 digits long  
**And** all the other details are correct  
**And** I submit the form  
**Then** the form should be redisplayed

# Using Gherkin continued

**Feature:** Feedback when entering invalid credit card details

In user testing we've seen a lot of people who made mistakes entering their credit card. We need to be as helpful as possible here to avoid losing users at this crucial stage of the transaction.

**Background:**

**Given** I have chosen some items to buy

**And** I am about to enter my credit card details

**Scenario:** Expiry date must not be in the past

**When** I enter a card expiry date that's in the past

**And** all the other details are correct

**And** I submit the form

**Then** the form should be redisplayed

# About Gherkin

- Available in a lot of natural languages
- use the .feature file extension
- saved as plain text
- meaning and structure is imposed by using keywords
- some keywords are:
  - Feature, Background, Scenario, Given, When, Then
  - And, But, \*, Scenario Outline, Examples

# Syntax check gherkin file

- In this chapter only feature files are written without complementing tests
- To check the gherkin structure -> dry run the feature file

```
$ java -cp ".:jars/*" cucumber.api.cli.Main -g step_definitions -
```

# About features

- Gherkin files begin with the Feature keyword
  - keyword doesn't affect the behavior of the Cucumber tests
  - it just gives a place to put some summary documentation

`Feature:` This is the feature title

`This is the description of the feature, which can span multiple lines.`

`You can even include empty lines, like this one:`

`In fact, everything until the next Gherkin keyword is include in the description`

- By convention name the feature file by converting the feature's name to lowercase characters and replacing the spaces with underscores

## A Template for Describing a Feature

Although feature descriptions are often helpful documentation, they're not mandatory. If you're struggling to work out what to say, the following template can be a great place to start:

In order to <meet some goal>

As a <type of stakeholder>

I want <a feature>

By starting with the goal or value that the feature provides, you're making it explicit to everyone who ever works on this feature why they're giving up their precious time. You're also offering people an opportunity to think about other ways that the goal could be met. Maybe you don't actually need to build this feature at all, or you could deliver the same value in a much simpler way.

This template is known as a *Feature Injection* template, and we are grateful to Chris Matts and Liz Keogh for sharing it with us.

# About scenarios

- each feature contains several scenarios
  - scenario is a single concrete example
  - all the scenarios together describe the expected behavior of the feature
- if the system behaves as described in the scenario
  - then the scenario will pass
  - if not, it will fail
- typically a feature has between five and twenty scenarios



# About scenarios cont.

- use scenarios to explore edge cases and different paths through a feature
- Scenarios all follow the same pattern:
  1. Get the system into a particular state
  2. Poke it (or tickle it, or...)
  3. Examine the new state
- So, start with a context -> describe an action -> check outcome was what was expected

# Given, When, Then

- use the keywords Given, When, and Then to identify the three different parts of the scenario

**Scenario:** Successful withdrawal from an account in credit

**Given** I have \$100 in my account # the context

**When** I request \$20 # the event(s)

**Then** \$20 should be dispensed # the outcome(s)

# And, But

- Each of the lines in a scenario is known as a step
- Add more steps to each Given, When, or Then using the keywords And and But

**Scenario:** Attempt withdrawal using stolen card

**Given** I have \$100 in my account

**But** my card is invalid

**When** I request \$50

**Then** my card should not be returned

**And** I should be told to contact the bank

# And, But

- If you don't want to use And or But -> repeat keyword

**Scenario:** Attempt withdrawal using stolen card

**Given** I have \$100 in my account

**Given** my card is invalid

**When** I request \$50

**Then** my card should not be returned

**Then** I should be told to contact the bank

- Less readable

# Shorthand

- If you think gherkin is too verbose

**Scenario:** Attempt withdrawal using stolen card

- \* I have \$100 in my account
- \* my card is invalid
- \* I request \$50
- \* my card should not be returned
- \* I should be told to contact the bank

# Stateless

- Each scenario must be able to be executed independently of any other scenario
- Don't share state between scenarios -> Cucumber won't check this
  - Extremely bad practice
- When writing a scenario:
  1. assume that it will run against the system in a default, blank state
  2. using Given steps to set up all the state for a scenario



**Matt says:**

## **Take Care with Your Naming Scenarios**

Even though they can't make your tests pass or fail, scenario names are surprisingly important to get right. Here are some reasons why it's a good idea to pay attention to them:

- When your tests break, it's the failing scenario's name that will give you the headline news on what's broken. A concise, expressive name here can save everyone a lot of time.
- Once you have a few scenarios in a feature file, you don't want to have to read the detail of the steps unless you really need to do so. If you're a programmer, think of it a bit like method naming. If you name the method well, you won't need to read the code inside it to work out what it does.
- As your system evolves, your stakeholders will quite often ask you to change the expected behavior in an existing scenario. A well-composed scenario name will still make sense even if you add an extra Then step or two.

A good tip is to avoid putting anything about the outcome (the Then part) of the scenario into the name and concentrate on summarizing the context and event (Given and When) of the scenario.

# Try this

1. Converting some of the concrete examples you wrote down earlier for your own project into Gherkin
2. Show them to someone who knows nothing about your project, and ask them what they think your application does
3. Practice describing what you're doing with Given/When/Then while you're doing everyday things such as starting your car, cooking breakfast, or switching channels on the TV. You'll be surprised how well it fits



# Comments

# This feature covers the account transaction and hardware-driver

Feature: Withdraw Cash

In order to buy beer

As an account holder

I want to withdraw cash from the ATM

# Can't figure out how to integrate with magic wand interface

Scenario: Withdraw too much from an account in credit

Given I have \$50 in my account

# When I wave my magic wand

And I withdraw \$100

Then I should receive \$100

- comments can quickly go stale and become confusing
- use sparingly -> use as notes for testers and programmers
- think of a comment as something more temporary

# Spoken languages

- other supported languages to express gherkin scripts

```
java -cp "jars/*:." cucumber.api.cli.Main --i18n help
elp
ar
bg
...
en
en-Scouse
en-au
en-lol
en-old
en-pirate
en-tx
...
nl
...
```

# Spoken languages cont.

```
$ java -cp "jars/*:." cucumber.api.cli.Main --i18n nl
| feature | "Functionaliteit" |
| background | "Achtergrond" |
| scenario | "Scenario" |
| scenario_outline | "Abstract Scenario" |
| examples | "Voorbeelden" |
| given | "* ", "Gegeven ", "Stel " |
| when | "* ", "Als " |
| then | "* ", "Dan " |
| and | "* ", "En " |
| but | "* ", "Maar " |
| given (code) | "Gegeven", "Stel" |
| when (code) | "Als" |
| then (code) | "Dan" |
| and (code) | "En" |
| but (code) | "Maar" |
```

# Spoken languages cont.

- Putting a # language: comment on the first line tells Cucumber which language is used

# Speaking dutch

```
$ nano features/checkout-nl.feature
```

```
# language: nl
```

Functionaliteit: Hollandse checkout

Achtergrond:

Gegeven een gebruiker heeft een aantal producten uitgezocht  
En staat op het punt zijn creditcardnummer in te voeren

**Scenario:** Een creditcardnummer moet bestaan uit 16 digits

Als de gebruiker meer dan 16 digits intoetst

En de andere gegevens zijn correct ingevoerd

Dan moet de invoer opnieuw op het scherm getoond worden met e

# Speaking dutch

```
$ ./cucumber.sh
```

You can implement missing steps with the snippets below:

```
@Gegeven("^een gebruiker heeft een aantal producten uitgezocht$")
public void eenGebruikerHeeftEenAantalProductenUitgezocht() throw
    // Write code here that turns the phrase above into concrete
    throw new PendingException();
}
```

```
@Gegeven("^staat op het punt zijn creditcardnummer in te voeren$")
public void staatOpHetPuntZijnCreditcardnummerInTeVoeren() throws
    // Write code here that turns the phrase above into concrete
    throw new PendingException();
}
```

```
@Als("^de gebruiker meer dan (\\d+) digits intoetst$")
public void deGebruikerMeerDanDigitsIntoetst(int arg1) throws Thre
```



Joe asks:

**So, Can I Mix Spoken Languages in My Features?**

A project can have a mix of features that use different spoken languages, yes. However, remember that the setting is for a *feature*, so all the scenarios in a feature have to use the same spoken language.