

Techniques

# Story decomposition and elaboration



# Story elaboration

**Story Elaboration** is used to define the detailed design and acceptance criteria for a story as needed to deliver a working solution.

Any story that is too large or insufficiently understood to elaborate, estimate, or deliver is ready for decomposition.

---

**Agile Extension to the  
BABOK® Guide**

# “breadth-before-depth”

It is the most common agile approach to story decomposition.

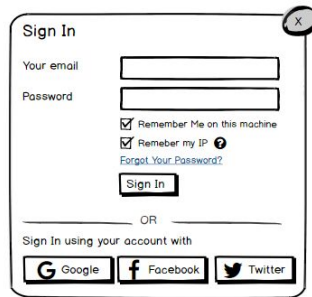
- First, describe a high level view of what business goals are
  - Second, decompose them into smaller components that provide increments of customer value (sometimes called minimal marketable features or MMFs)
  - Third, decompose those into stories
  - Finally, elaborate stories into smaller increments with acceptance criteria
-

# Story template

## Value statement

As a <persona> who <context>  
I want <a feature>  
So that <value>

## Visual reference



Sign In

Your email

Password

☒ Remember Me on this machine




☒ Remember my IP ⓘ

[Forgot Your Password?](#)

Sign In

OR

Sign In using your account with

 Google  Facebook  Twitter

## Acceptance criteria

1. Criterion 1
2. Criterion 2

## Context

Lorem ipsum dolor sit  
amet...

## Assumptions

1. Assumption 1
2. Assumption 2

# Techniques BDD



Photo: [https://unsplash.com/@anmol\\_kerketta](https://unsplash.com/@anmol_kerketta)

# How do we do it?

- First, the team explores the domain outlined in the story, and collaborates to produce concrete examples that describe the behaviour they want
  - Sometimes that discussion is hard, because it highlights all the misunderstandings and assumptions you'd normally discover much later
  - But it is a good thing, because it reduces re-work (waste)
  - Next, the team agrees on those examples as acceptance tests
-

# Who participates?

Three perspectives need to be involved in the conversation:

- Business - people who know the problem domain
- Development - people who develop a solution
- Testing - people who do quality control on the output

This type of meeting is called **3 Amigos**.

---

# BDD

The process described before is the foundation of behaviour driven development (BDD) - a software development practice that puts **the creation of real business scenarios** as acceptance tests before the start of development.

It uses a customer readable, domain specific language to specify the intended customer behaviour which satisfies the customer need. It increases speed of delivery by developing only what is needed and creates opportunity for user **acceptance test automation**.

---



# Techniques Gherkin scenarios



Photo: <https://unsplash.com/@harshalhirve>

# Why “gherkin”?

Because of the framework called **Cucumber**.

**Cucumber** is a test automation tool and framework that understands **Gherkin scenarios** and maps the written scenarios to a set of automated tests.

---

# Why “gherkin”?

**As a** customer who receives SMS bills  
**I want to** see the due date and payment amount in the SMS  
**So that** I know when and how much to pay

1. Due date is present ✓

2. Payment amount is present ✗

Cucumber

Automated tests

# Gherkin scenarios

A gherkin scenario consists of:

- **Given**: what must be true for the scenario to start
- **When**: what is the trigger/activity that we cater for
- **Then**: what is the expected result

We use this syntax to add examples for each acceptance criterion.

---

# Given

**Given** puts the system in a known state.

It's a set of key pre-conditions for a scenario.

Pre-condition is something that is assumed to be true during the scenario execution and is not intended to be in scope of verification within a test.

---

# Given

## Dos

- user is in a certain state
- certain elements are present
- another scenario passed

## Don'ts

- any user interactions:  
**Given** *I press the button*
- any third party dependencies that cannot be performed by the tester:  
**Given** *Google is down*

# When

**When** is the key action a user will take. It's the action that leads to an outcome.

This is the main interaction for the scenario, the actions that the tester should perform or mimic.

# When

## Dos

- user interactions
- system interactions
- timer events

**Given** my bill preference is paper

**When** I log in

**Then** I see an option to switch to SMS

## Don'ts

- static pre-conditions

**Given** I'm logged in

**When** my bill preference is paper

**Then** I see an option to switch to SMS



# Then

**Then** is the observable outcome.

It's what happens after the user performs the action. This is where the actual check is done: is the result as expected or not?

# Then

## Dos

- Observable outcomes that can be validated
- Changes in object's state
- Objects appearing / disappearing
- Text or values presented or calculated

## Don'ts

- Vague results that cannot be easily interpreted

**Then** the layout looks good

# And and But

**And** (and **But**) is a human readable way to repeat the previous key word.

**Given** my preference is paper

**Given** I don't have a phone number on profile

**When** I click change to SMS bill

**When** I agree to T&Cs

**Then** I see a warning message

**Then** I see a form to add a phone number

---

**Given** my preference is paper

**But** I don't have a phone number on profile

**When** I click change to SMS bill

**And** I agree to T&Cs

**Then** I see a warning message

**And** I see a form to add a phone number

# Scenarios

**Scenario** is used to group the statements & give them a title.

A feature or user story is likely to have multiple scenarios. Giving each scenario a title means the team can understand what is being tested without having to read all the Given/When/Thens.

Some people follow “the Friends format” e.g.

- The one where .... the user has no mobile number.
  - The one where .... the user has a mobile number.
-

# Scenarios

**Scenario:** the one where the user has no mobile number

**Given** my preference is paper

**But** I don't have a phone number on profile

**When** I click change to SMS bill

**And** I agree to T&Cs

**Then** I see a warning message

**And** I see a form to add a phone number

# Feature files

A **Feature** is used to give a title for the whole feature/piece of functionality.

A feature contains a set of scenarios grouped by a common solution component.

Technically, it is a text file with an extension *.feature*

---

Cucumber will look for these files in the repository to pull the Scenarios and Steps definitions from it.

# Background

**Background** sets the context for all scenarios within a Feature file.

If you find that scenarios have common Given/Ands, Background can be used to eliminate the repetition.

Background is run before each of your scenarios. Scenarios can still have their own Given/When/Thens.

---

# Background

## Background:

**Given** my preference is paper

**Scenario:** the one where the user has no mobile number

**Given** I don't have a phone number on profile

**When** I click change to SMS bill

**And** I agree to T&Cs

**Then** I see a warning message

**And** I see a form to add a phone number

**Scenario:** the one where the user has a mobile number

**Given** I have a phone number on profile

**When** I click "change to SMS bill"

**And** I agree to T&Cs

**Then** I see a confirmation message

**And** my preference if updated

**Scenario:** the one where the user has no mobile number

**Given** my preference is paper

**And** I don't have a phone number on profile

**When** I click change to SMS bill

**And** I agree to T&Cs

**Then** I see a warning message

**And** I see a form to add a phone number

**Scenario:** the one where the user has a mobile number

**Given** my preference is paper

**And** I have a phone number on profile

**When** I click "change to SMS bill"

**And** I agree to T&Cs

**Then** I see a confirmation message

**And** my preference if updated



# Scenario outlines

**Scenario outlines** are used to combine a set of similar scenarios.

It allows to use placeholders in the scenario text and supply with test data that will replace those placeholders.

It mean you can have one scenario and a set of test data rather than having multiple similar scenarios, therefore it makes the feature file much more readable.

---

# Scenario outlines

**Given** my account number is “1234”  
**When** I try to find my account by number “4321”  
**Then** I see an error message

**Given** my account number is “1234”  
**When** I try to find my account by number “abcd”  
**Then** I see an error message

**Given** my account number is “1234”  
**When** I try to find my account by number “1234”  
**Then** I see a success message

**Scenario outline:** search for account

**Given** my account number is <number>  
**When** I try to find my account by number <query>  
**Then** I see a <type >message

## Examples:

number	query	type	
1234	4321	error	
1234	abc	error	
1234	1234	success	

# Tags

**Tags** can be used to group acceptance tests from multiple features or to add extra metadata to scenarios to filter them for execution.

**@jira-103 @automated @critical**

**Scenario:** the one where the user has a mobile number

**Given** my preference in paper

**And** I have a phone number on profile

**When** I click “change to SMS bill”

**And** I agree to T&Cs

**Then** I see a confirmation message

**And** my preference if updated

---

# Test data

→ Strings:

**Given** my account number is "1234"

→ PyStrings:

**Given** my account description is:

"""

Main account for property on:

1. Main road

2. Secondary road

"""

---

→ Data table:

**Given** I have the accounts:

account number	account name
1234	main
4321	secondary

# Why “gherkin”?

- Supports for automation
  - Structures thoughts
  - Makes requirements less ambiguous
  - Allows flexibility of documentation by creating a lightweight, living requirements and testing document
-

# Techniques Spikes



Photo: <https://unsplash.com/@mpayne66>

# Spikes

Spikes are an invention of Extreme Programming (XP).

They are special type of backlog items that are used **to gain the knowledge** necessary to reduce the risk of a technical approach, better understand a requirement, or increase the reliability of a story estimate.

---

# Types of spikes

There are three types of Spikes:

- **Functional**: analyses a story and determines how to break it down into smaller stories or tasks, or identify where the risk and complexity exists.
  - **Technical**: determines feasibility or impact of a story or task to understand the technical design.
  - **Exploratory**: explores organisational risks or impacts for a particular initiative or backlog item.
-



# Spike template

1. Goal
  2. Timebox
  3. Considerations
  4. Showstoppers
  5. Expected results
-

# Spike example

**As a** customer who receives SMS bills  
**I want to** see the due date and payment amount in the SMS  
**So that** I know when and how much to pay



But can we even theoretically get this data to include in an SMS?

Let's have a spike!

- 1. Goal:**  
Learn how to retrieve the data and make it available for SMS sending system
- 2. Timebox:**  
2 days
- 3. Considerations:**  
Connection needs to be secured
- 4. Showstoppers:**  
Once you learn that data is not accessible
- 5. Expected results:**  
Integration POC

# Considerations

## Strengths

- Specific activities and time-box provides focus for the team to get clarity
- Gives permission to spend time on value-driven research
- When used early in team formation, can help team members build and share knowledge about each other and the technology to be used for the solution

## Limitations

- Can be too long a time-box or too large an item to have clear objectives and outcome
- The term can be incorrectly used to reference follow-up conversations
- If used too frequently, this indicates the product backlog refinement is not meeting team needs

# Techniques Ceremonies





# Backlog grooming

**Goal:** to align backlog with product vision

**Who is involved:** BA and PO

**BA role:**

- Work with the PO/stakeholders to arrange the backlog so it reflects the product's vision and is ready for refinement with the team
- Add details to the stories during the grooming session or plan the elaboration of the stories
- Get ready for a refinement session

# Backlog refinement

**Goal:** to familiarise the team with the backlog items, answer questions, get estimations, make sure the backlog items reach definition of ready

**Who is involved:** the team

**BA role:**

- Present the backlog and the user stories in details
- Answer the questions the team may have
- Note down questions to clarify after the session
- Facilitate estimation
- Update stories: add estimates, assumptions, uncovered details

# Definitions

**Definition of Ready** is a set of criteria the team agrees must be satisfied to consider an item "ready" for the next iteration.

**Definition of Done** is a set of criteria which must be met before a backlog item is considered done.

---

**Agile Extension to the  
BABOK® Guide**



# Sprint (iteration) planning

**Goal:** to plan the next delivery iteration

# Iteration

A defined time period when increments of a product are developed and tested and made ready to deliver to the customer.

---

**Agile Extension to the  
BABOK® Guide**

# Sprint (iteration) planning

**Goal:** to plan the next delivery iteration

**Who is involved:** the team

**BA role:**

- Present the backlog and the user stories
- Facilitate estimation
- Update stories: add estimates, assumptions, uncovered details
- Make sure the team is comfortable with the scope they commit to

# Sprint review

**Goal:** to review what has been developed during the sprint and call out any outstanding work

**Who is involved:** the team

**BA role:**

- Document outstanding work for the tickets not done
  - Update the backlog based on the team decisions to unblock further progress
-

# Sprint showcase

**Goal:** to present the results of the sprint to the stakeholders

**Who is involved:** the team

**BA role:**

→ No specific role/may present as any other team member

# Retrospective

**Goal:** to reflect on the previous iteration and discuss opportunities for improvement

**Who is involved:** the team

**BA role:**

- No specific role - participates as a team member
  - May use BA process improvement techniques to suggest activities/improvements
-

# Retrospective

Norman Kerth's four questions\*:

1. What did we do well, that if we don't discuss we might forget?
2. What did we learn?
3. What should we do differently next time?
4. What still puzzles us?

---

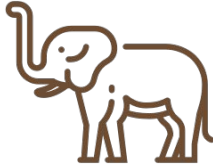
**\* Project Retrospectives: A Handbook for Team Reviews**

# Delivery horizon Mapping the Principles





# Principles on delivery horizon



## **See the whole**

Develop shared understanding of how individual stories advance outcomes.

Find a way to measure progress against goals.

Each item is prioritised to achieve the overall goal.

## **Think as a Customer**

---

Define who the customer is for each story and align the value delivered to customer experience.

Consider how processes and product being delivered meet the needs of the customers.



# Principles on delivery horizon



## **Analyze to Determine What is Valuable**

Refine the backlog based on feedback and learnings.

Consider factors that impact creation of value (team happiness, defects rate, velocity, quality of improvements).

## **Get Real Using Examples**

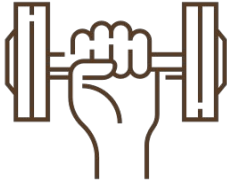
Use examples when refining stories and writing AC.

---

Make acceptance scenarios aligned with customer experience



# Principles on delivery horizon



## **Understand what is doable**

Consider needs, expected outcomes, constraints, risks to refine stories.

Set expectations of what can be accomplished.

## **Stimulate Collaboration and Continuous Improvement**

---

Encourage close collaboration e.g. 3 amigos.

Leverage the strengths of cross-functional teams.



# Principles on delivery horizon



## **Avoid Waste**

Focus on stories that deliver maximum value first.

Maximise work not done.

# Agile BA Study guide



Igor Arkhipov, CBAP