COMP1531

🐶 Software Engineering
7.2 - Requirements - Use Cases, User Stories

In this lecture

Why?

- Requirements can be very human, and express complex flows. We need ways to model this.

What?

- User Stories
- User Acceptance Tests
- Use Cases
- Use Case Representations

SDLC


REQUIREMENTS ANALYSIS
DESIGN
SDLC PHASES
MAINTENANCE
DEVELOPMENT
DEPLOYMENT
TESTING

User Stories - Overview

User Stories are a method of requirements engineering used to inform the development process and what features to build with the user at the centre.

User Stories - Structure

When a customer tells you what they want, try and express it in the form **As a < type of user >, I want < some goal > so that < some reason >**

E.G. They say:

- E.G. They say:
  - A student can purchase monthly parking passes online
- But your story becomes:
  - As a **student**, I want to **purchase a parking pass** so that **I can drive to school**

User Stories - Nature

User stories:

- Are written in non-technical language
- Are user-goal focused, not product-feature focused
  - User stories inform feature decisions

Why do we care?

- The keep customers at the centre
- Keep it problem focused, not solution focused

User Stories - Activity

Building a to-do list

User Stories - More

Read more about user stories here:

https://www.atlassian.com/agile/project-management/user-stories

How do we know we've met the user story requirement?

INVEST

- I = Independent: user story could be developed independently and delivered separately
- N = Negotiable: avoid too much detail.
- V = Valuable: must hold some value to the client
- E = Estimable: *we'll get to this in a later lecture*
- S = Small: user story should be small
- T = **Testable**

User Acceptance Criteria

- Break down a user story into criteria that must be met for the user, or customer, to accept
- Written in natural language
- Can be refined before implementation

Example

As a user, I want to use a search field to type a city, name, or street, so that I can find matching hotel options.

- The search field is placed on the top bar
- Search starts once the user clicks “Search”
- The field contains a placeholder with a grey-colored text: “Where are you going?”
- The placeholder disappears once the user starts typing
- Search is performed if a user types in a city, hotel name, street, or all combined
- The user can’t type more than 200 symbols

Best practices

- Acceptance criteria should not be too broad
- … but nor should they be too narrow
- Minimise technical detail
  - They can be more technical than the story itself, but client still needs to understand them
- While they can be updated during development, they should first be written *before* it starts

From Criteria to Testing

- *Acceptance Tests* are tests that are performed to ensure acceptance criteria have been met
- Not all acceptance criteria can easily be mapped to *automated* acceptance tests
- Acceptance tests are *black-box* tests

Example 2:

As a user, I can log in through a social media account, because I always forget my passwords

- Can log in through Facebook
- Can log in through LinkedIn
- Can log in through Twitter

Scenario Oriented AC

- The Acceptance criteria from before are often referred to a rule-based AC
- Sometimes it is preferable to have AC that describe a scenario
- This can be done in the Given/When/Then format:
  - *Given* some precondition
  - *When* I do some action
  - *Then* I expect some result

Example 3:

As a user, I want to be able to recover the password to my account, so that I will be able to access my account in case I forgot the password.

Scenario: Forgot password

Given: The user has navigated to the login page

When: The user selected forgot password option

And: Entered a valid email to receive a link for password recovery

Then: The system sent the link to the entered email

Given: The user received the link via the email

When: The user navigated through the link received in the email

Then: The system enables the user to set a new password

Which one to use?

- Rule-based acceptance criteria are simpler and generally work for all sorts of stories
- Scenario-based AC work for stories that imply specific user actions, but don't work for higher-level system properties (e.g. design)
- Scenario-based AC are more likely to be implementable as tests

Further reading

- https://www.mountaingoatsoftware.com/blog/the-two-ways-to-add-detail-to-user-stories
- https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/
- https://dzone.com/articles/acceptance-criteria-in-software-explanation-exampl

Use cases

- Represent a *dialogue* between the user and the system, with the aim of helping the user achieve a business goal
- The user initiates *actions* and the system responds with *reactions*
- They consider systems as a black box, and are only focused on high level understanding of flow

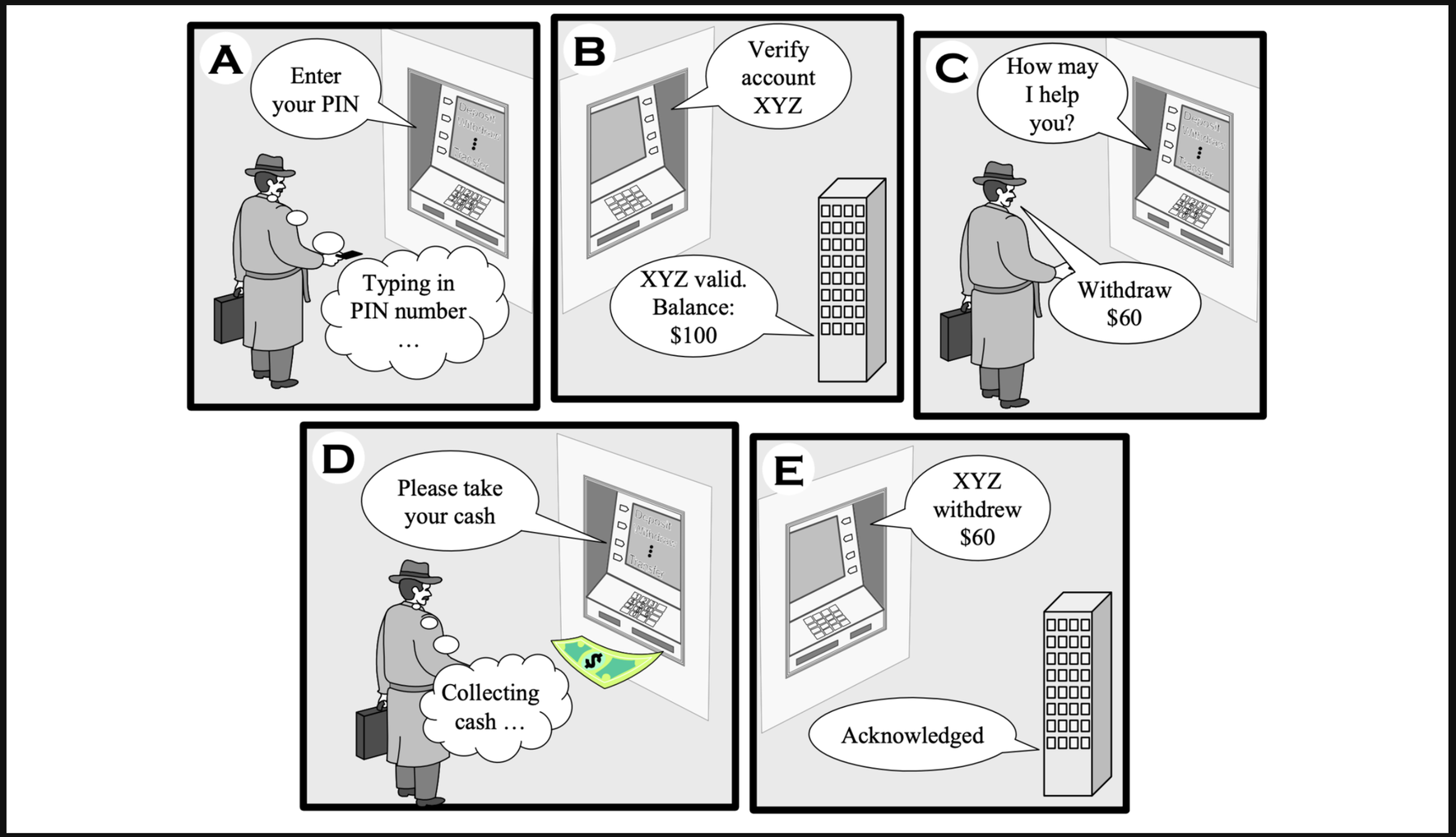
Use Case Representations

Generally you can represent use cases as:

- Informal list of steps (written)
- Diagramatic (visual)

There is a range of different approaches
that can be taken too, e.g. Cockburn style
(not required reading)

Use-case Diagrams



Use-case written form

MAIN SUCCESS SCENARIO
Step 1. ATM asks customer for pin
Step 2. Customer enters pin
Step 3. ATM asks bank to verify pin and account
Step 4. Bank informs ATM of validity and balance of account
Step 5. ATM asks customer what action they wish to take
Step 6. Customer asks to withdraw an amount of money
Step 7. ATM Dispenses money to customer
Step 8. ATM informs bank of withdrawal

Use-case (Background)

- **Use Case**: Withdraw Money
- **Goal in Context**: Customers need to withdraw money from their accounts without entering the bank
- **Scope**: ATM, banking infrastructure
- **Level**: Primary Task
- **Preconditions**: The customer has an account with the bank
- **Success End Condition**: The customer has the money they needed to withdraw
- **Failed End Condition**: The customer has no money
- **Primary Actor**: Customer
- **Trigger**: Customer puts card into ATM

Template for background

- **Use Case**: <the name should be the goal as a short active verb phrase>
- **Goal in Context**: <a longer statement of the goal, if needed>
- **Scope**: <what system is being considered black-box under design>
- **Level**: <one of: Summary, Primary task, Subfunction>
- **Preconditions**: <what we expect is already the state of the world>
- **Success End Condition**: <the state of the world upon successful completion>
- **Failed End Condition**: <the state of the world if goal abandoned>
- **Primary Actor**: <a role name for the primary actor, or description>
- **Trigger**: <the action upon the system that starts the use case, may be time event>

Use Cases In More Depth

- Can be used to model variations in steps (e.g. Insufficient funds)
- If you wish to know more about use cases, see here:
  - Software Engineering - Ivan Marsic (Chapter 2, Section 4)
  - http://www.cs.otago.ac.nz/coursework/cosc461/uctempla.htm
  - Writing Effective Use Cases - Alistair Cockburn

Optional (Monorail requirements)

Let's take the opportunity to build our requirements for a UNSW monorail. Ensure some of the requirements are expressed in terms of user stories and/or use cases.

https://tharunka.arc.unsw.edu.au/src-approves-plans-for-monorail/

Feedback