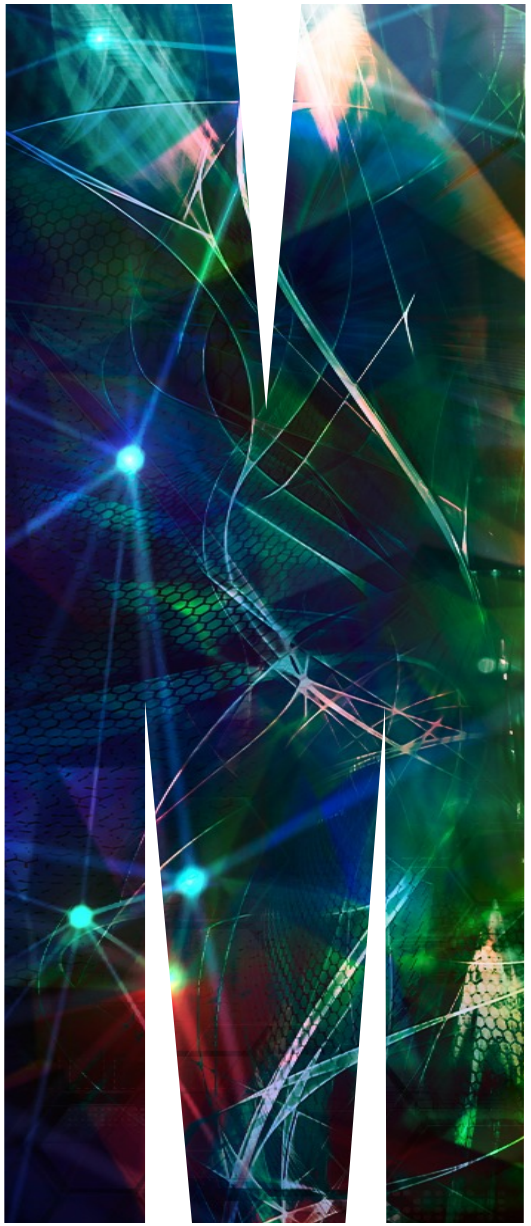
MONASH University

FIT5136 Software Engineering

Week 2 Requirements Engineering

Delvin Varghese

Date Updated: 1 August 2022



FLUX QUIZ
WEEK 1

MONASH University

Overview

Requirements Engineering

- What is a requirement?
- Requirements engineering
- Types of requirements
- Requirements engineering process
- Requirements gathering techniques

Usability design principles

- Donald Norman principles
- Ben Shneiderman's 8 golden rules

Case Study – Parallax Use Case

- Identifying actors -- available on Moodle

REQUIREMENTS ENGINEERING

MONASH University

Requirements engineering

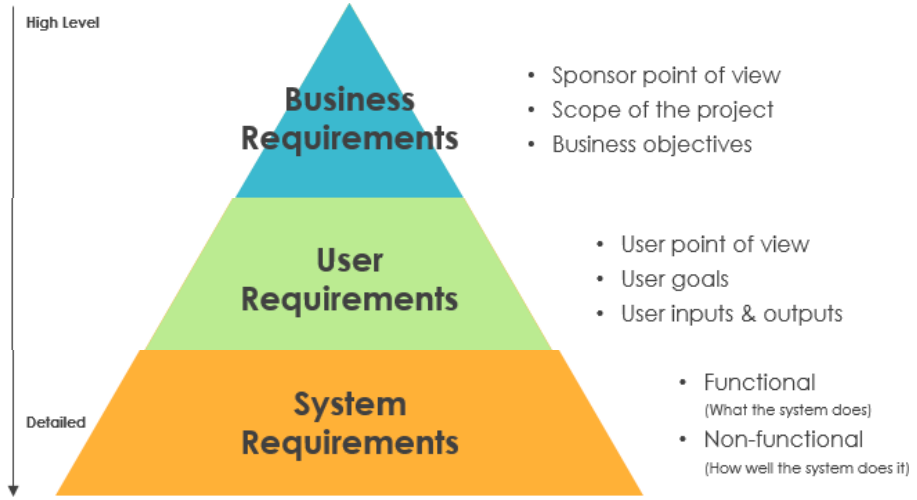
- The process of defining, documenting and maintaining requirements
- Part of software engineering process
- The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed. (In a simple way, Understand client's requirement in a more technical way and understand its limitations)
- The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.
- The system requirements provided by the client are usually informal and brief, so the process of finding the detailed requirement of the system is requirements engineering

Software Process

- A structured set of activities required to develop a software system
- Break task into smaller steps
- Many different types of software processes, but all processes involve the following activities:
  1. **Requirements and Analysis (Specification)** – defining what the system should do
  2. **Design and Implementation** – defining how the system should do it, organization of the system and implementing the system
  3. **System verification and validation** – checking that it does what the customers want
  4. **Evolution (Maintenance)** – changing the system in response to changing needs

Types of requirements

- **Business:** what is it that the business requires the system to do
- **User:** what is it that the user (or customer) requires the system to do. A user can be a stakeholder.



- **Functional:** what the system should functionally do
- **Non-functional:** quality of service system required to provide
- **Constraint:** things that system should prohibit the user to do
- **Implementation:** platform, etc requirements of the system

System Requirement
Be careful about developer's misconceptions

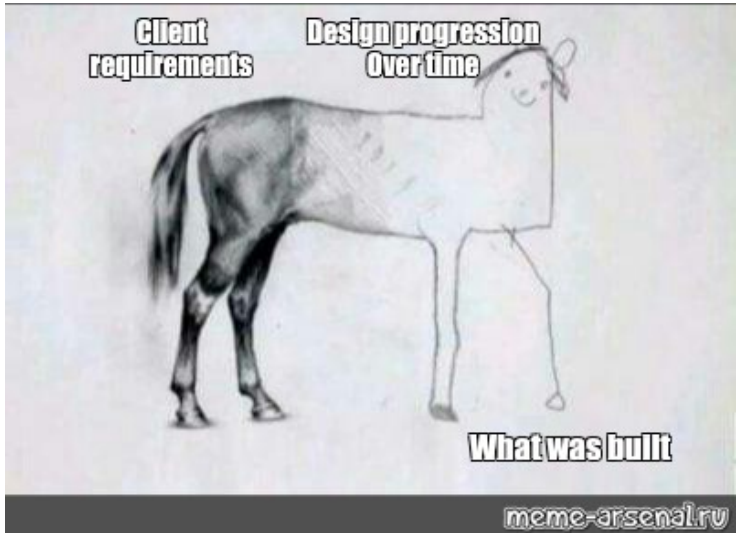
Functional requirements

- Describe functionality or system services
- Describe how the system should react to particular inputs and how the system should behave in particular situations
    - Expressed as inputs and outputs

  Think about the functions/ method you create for Java/ Python
- Examples:
    - A user shall be able to search properties for all suburbs.
    - At the end of each month, the system shall generate a list of properties sold or rented
    - Each property agent shall be uniquely identified by his or her 8-digit employee number.

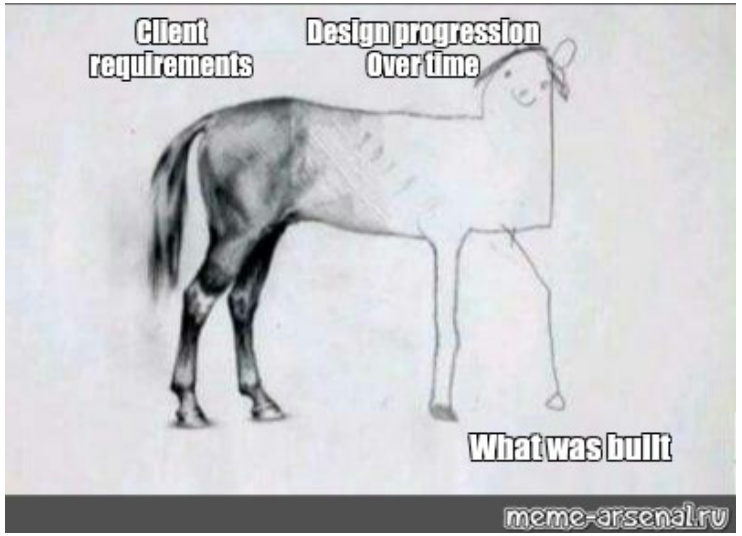
Imprecise requirements

- Requirements must not be ambiguous, incomplete and inconsistent
  - Problems arise when functional requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider “A user shall be able to search a property based on locations”



Imprecise requirements

- Requirements must not be ambiguous, incomplete and inconsistent
  - Problems arise when functional requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider “A user shall be able to search a property based on locations”
  - **One interpretation**: To search a property based on suburb
  - **Another interpretation:** To search for a property based on local government area/ street



Complete and consistent requirements

- Complete
  - They should include descriptions of all the steps required to achieve the function.
- Consistent
  - There should be no conflicts or contradictions in the descriptions.
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

Non-functional requirements

- A specific criteria that can be used to judge the operation of a system, rather than specific behaviours
- The nitty-gritty of the services or functions offered by the system. The requirements that makes the quality of the system better.
- For example, number of customers allowed, password requirements, response time, storage requirements, I/O device capability, among others.

Importance of non-functional requirements

- Often apply to the system as a whole rather than individual features or functions.
- Non-functional requirements may be more critical than functional requirements.
  - If they are not met, the system may be useless.

Examples of non-functional requirements

- The property management system shall be available to all property agents during normal working hours (Mon–Fri, 0830–1730).
- Downtime within normal working hours shall not exceed five seconds in any one day.
- Customers must authenticate themselves using their drivers license.
- After receiving an inspection request, the system must acknowledge with a response e-mail within 10 seconds.
- Process requirements e.g. mandating a particular IDE, programming language or development method.
  - In some cases, client wants you to recommend if they are not sure

Non-functional requirements implementation

- Non-functional requirements normally affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, “design” the system to minimize communications between components.
- A single non-functional requirement, such as a security requirement, may generate several related functional requirements that define system services that are required.

FLUX Code:

QJN9BW

https://flux.qa/QJN9BW

FLUX : QJN9BW

Multiple Choice

Which one of these is a non-functional requirement?

A unread emails will be marked with an exclamation sign

B user wants to register herself to the application

C data backup must be performed 2 every week

D All of the above

E None of the above

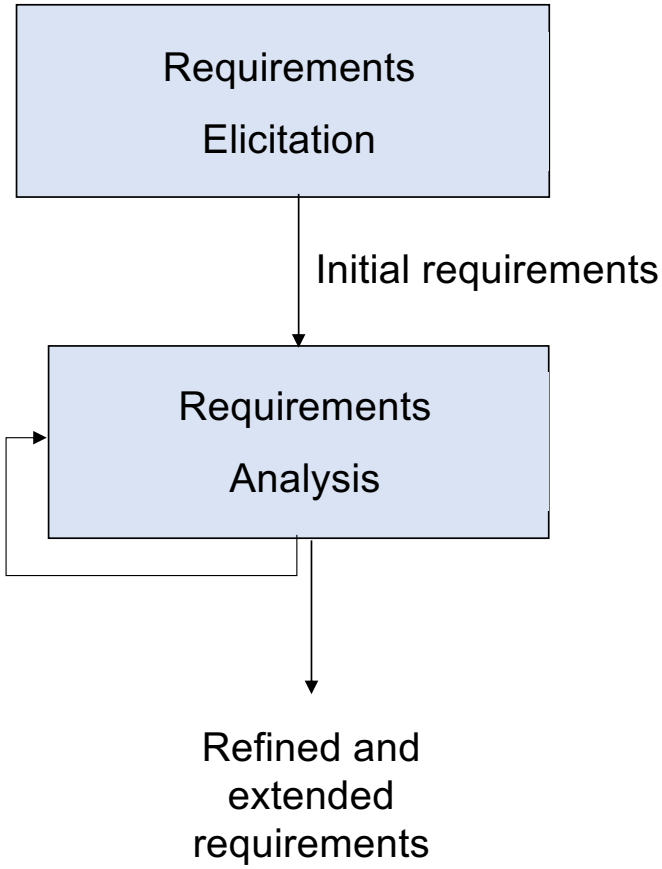
Requirements engineering (RE) process

- The process used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.

In practice, RE is an iterative activity in which these processes are interleaved.

Requirements elicitation and analysis

- Discovering the client's requirements
  - Requirements *elicitation* (or *requirements gathering*)
  - Methods include interviews and surveys
- Refining and extending the initial requirements
  - Requirements *analysis*
- Involves technical staff working with customers/clients to find out more about the application domain, the services that the system should provide and the system's operational constraints.
- May involve stakeholders - end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.



Determining what the client needs

- Needs skill to elicit the appropriate information from the client
- The client is the only source of this information
- The solution:
  - Obtain initial information from the client
  - Use this initial information as input to the software engineering process
  - Follow the steps to determine the client's real needs

Steps to determining what the client needs

First, gain an understanding of the *application domain* (or *domain*, for short)

- The specific environment in which the target product is to operate

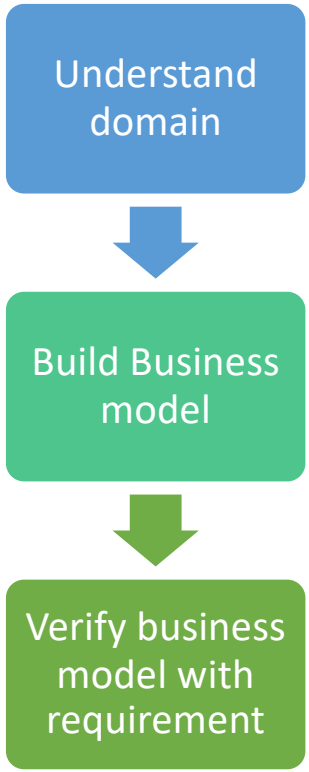
Second, build a business model

- Model the client's business processes

Third, use the business model to determine the client's requirements

Is that it??

- NO - Iterate the above steps



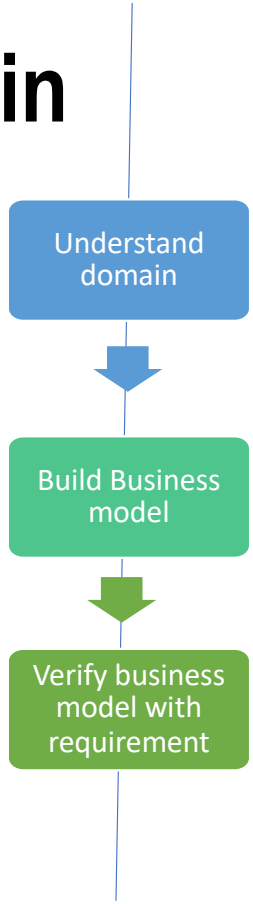
Understanding the domain

Every member of the development team must become fully familiar with the application domain

- Correct terminology is essential

Construct a glossary (Wordlist)

- A list of technical words used in the domain, and their meanings



A *business model* is a description of the business processes of an organization that generate revenue

Before we move on…

What must the product be able to do?

- Misconception is that we fulfill the client's full expectation
- Instead, we as the software engineer should help the client realize what they need in a more technical & controllable way

every finished project be like..

COMFORT BREAK
Any questions?

"Hello! Relax, take a seat, we'll begin the requirements elicitation session shortly."

MONASH University

REQUIREMENTS GATHERING

YOU CAN USE A RANGE OF TECHNIQUES FOR REQUIREMENTS ELICITATION, INCLUDING INTERVIEWS AND SURVEYS.

MONASH University

Requirements gathering techniques

- To construct a system, you'll need to gather information (of course)
- There's a variety of approaches:
  - Interviews
  - Questionnaire
  - Existing Documentation and system(s)
  - Workshops
  - Domain Experts
  - Direct Observation
  - 'Riding the trucks' - get to know the business
  - Prototyping

Photo by Markus Spiske from Pexels

Key aspects of interviewing

- Interview ***all*** stakeholders
  (A stakeholder – any person or organization who is affected by the system in some way and so who has a legitimate interest)
- Describe **business mission**
- Determine how **success is measured**
- Be aware of the **value chain**
- Don't forget the **IT department**…

Interviews

There are two types of questions

- **Close-ended** questions require a specific answer (e.g. yes/no)
- **Open-ended** questions are posed to encourage the person being interviewed to speak out (detailed explanation)

There are two types of interviews

- In a **structured** interview, specific pre-planned questions are asked, frequently close-ended
- In an **unstructured** interview, questions are posed in response to the answers received, frequently open-ended. May start with 1 or 2 pre-planned but then follow-up questions relate to answers given

Photo by Christina @ wocintechchat.com on Unsplash

Advantages and disadvantages of interviews

Advantages

Disadvantages

Advantages and disadvantages of interviews

Advantages

- Get a **first-hand impression** of business requirements
- **Create bonds** (Connections, opinions)
- High **response** rate
- Ambiguities can be **clarified** and incomplete answers **followed up**
- Can be recorded.

Disadvantages

- **Time** to **cover** all relevant **people** (Time Consuming)
- What people **say** is **not** always what they **really do**

Questionnaire

- Online, paper based, e-mail.
- **Advantage**: Cover a wide range of people. Their opinions are important
- **Disadvantages**:
  - **Low** response rate, **not many** people are willing to respond.
  - Questions may **not** be **well understood**
- Questions must be **carefully** prepared to be effective

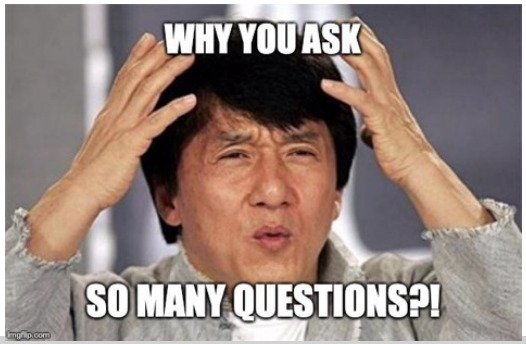


Photo by RODNAE Productions from Pexels

Existing documents and systems

- An important source of information
- Advantage: Great for background briefing before undertaking one of the other approaches
- Disadvantage: Out of date, may not match reality
- Types of documents:
  - Forms
  - Reports
  - Screenshots
  - Workflow manuals
  - Strategy Documents

Workshops

- Sometimes referred to as JAD sessions:
  - Joint Application Development
- Get all stakeholders in a room for a couple of days and facilitate discussion.
  - Build the model with everyone there.
  - People bounce ideas of each other.
  - No delay in confirming or denying aspects of the model
- Requires a conference room, whiteboard, etc., facilitator and a scribe.
- Nowadays can be run on Zoom

Workshops

- Can be very difficult to organise, particularly if senior people are involved. Requires taking people off regular work for a week or so.
- Once the initial session is complete, **all notes** must be **written up** and a **working document** produced.
- This is then **distributed** to all involved for **comment** and **feedback**.
- May even run a **subsequent session** to refine and develop the model.
- **Labour intensive**, but yields **great results**.

Direct observations

Don’t just rely on interviews and workshops to understand business processes

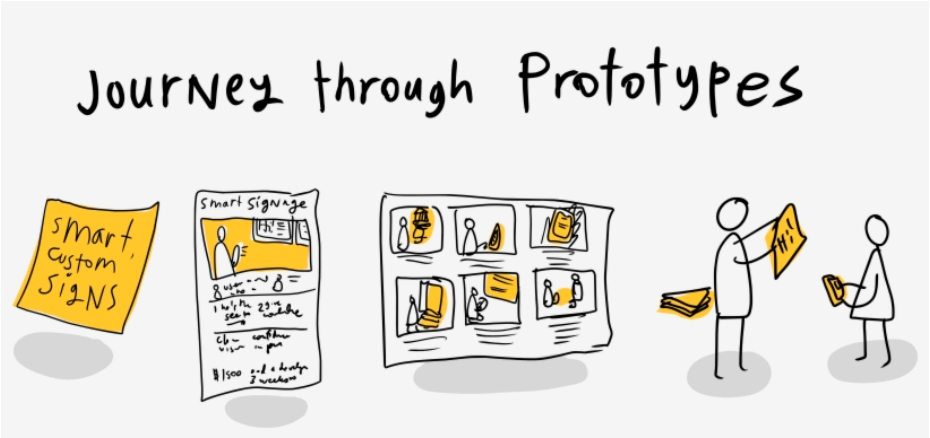
Often a significant difference between what is documented and what happens

Get out and see the business processes in action

- Talk to frontline staff, watch what they do
- See how variations to the ‘official’ way of doing things occurs in different places
- See how people interpret and use the same data in different ways
- Look at how people use systems
- Understand how data is created and used

Prototyping

- Provides the **basis** for early, rapid iterative cycles – works great in **agile**
- Generally focuses on the **user interface** of the application software
- Remember to manage **client expectations**!
  - Prototype is **not the real system**
  - Clients will wonder why it takes another **6 months** after they saw something that looks like it works before anything real gets delivered
- More on prototypes later…



FLUX : QJN9BW

Multiple Choice

Which of following requirements gathering technique will fit best for Agile methodology?

A Direct Observation

B Survey

C Client Interview

D Prototype

Requirements specification (Document)

- The process of **writing** down the user and system **requirements** in a requirements document.

- **User requirements** must be understandable by end-users and customers who do **not** have a **technical** background.

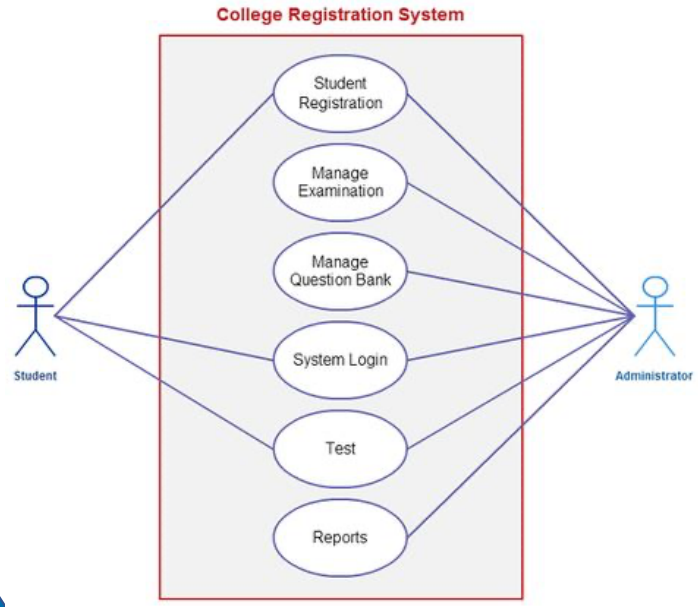
- **System requirements** are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

REQUIREMENTS GATHERING

MONASH University

USE CASE DIAGRAMS

TO DESCRIBE AND DOCUMENT ALL INTERACTIONS WITH THE SYSTEM, WE USE *USE CASE DIAGRAMS*.

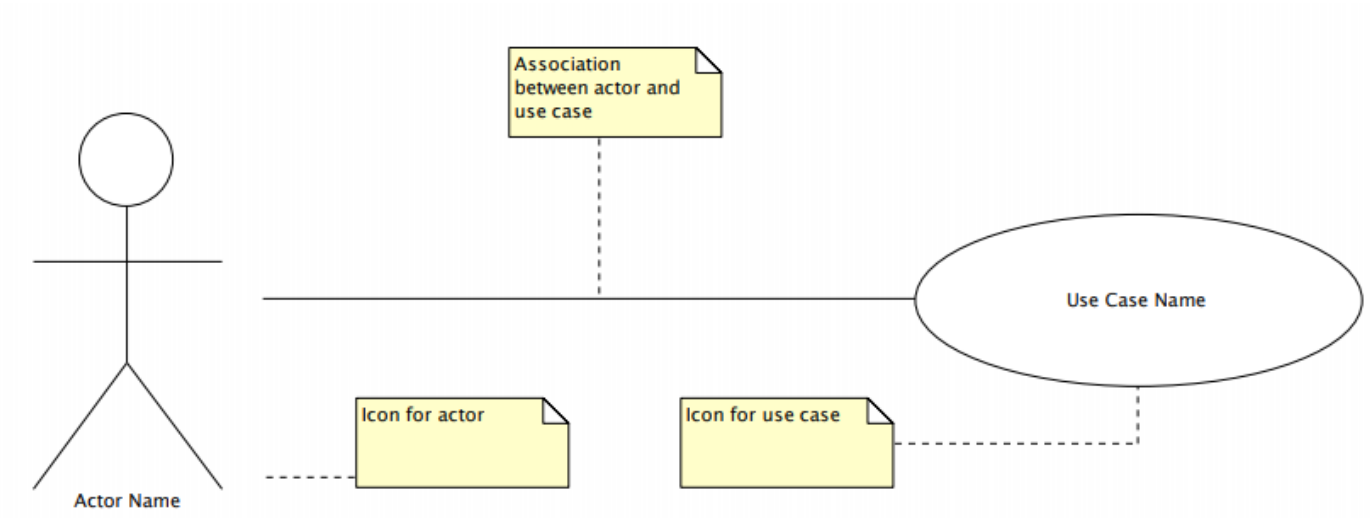


MONASH University

Use case diagram

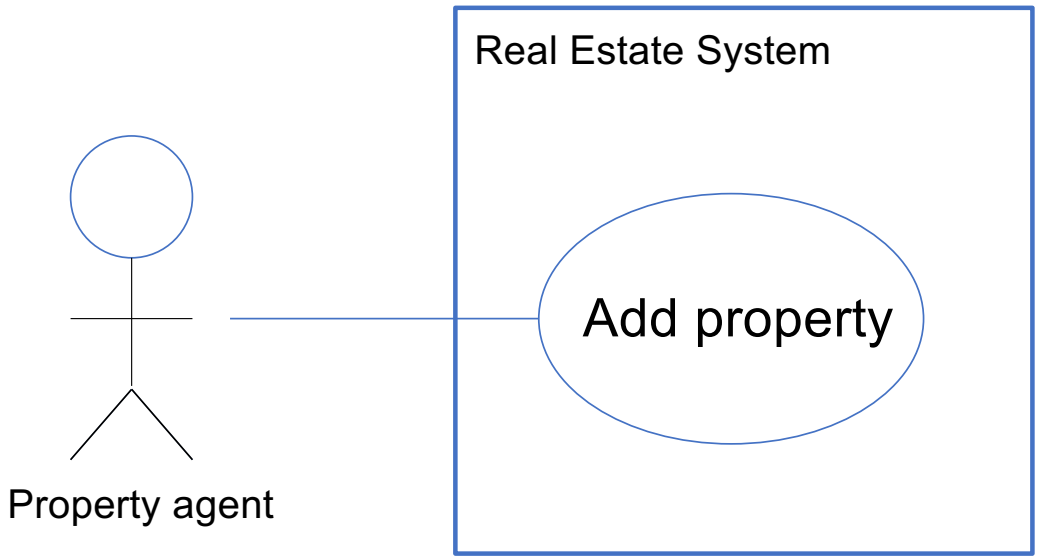
A use case models an interaction between the software product itself and the users *(actors)* of that software product

- Tool for formalizing your understanding of requirements.
- Captures some user-visible function or behaviour.
- Relationship(s) between actors and use cases

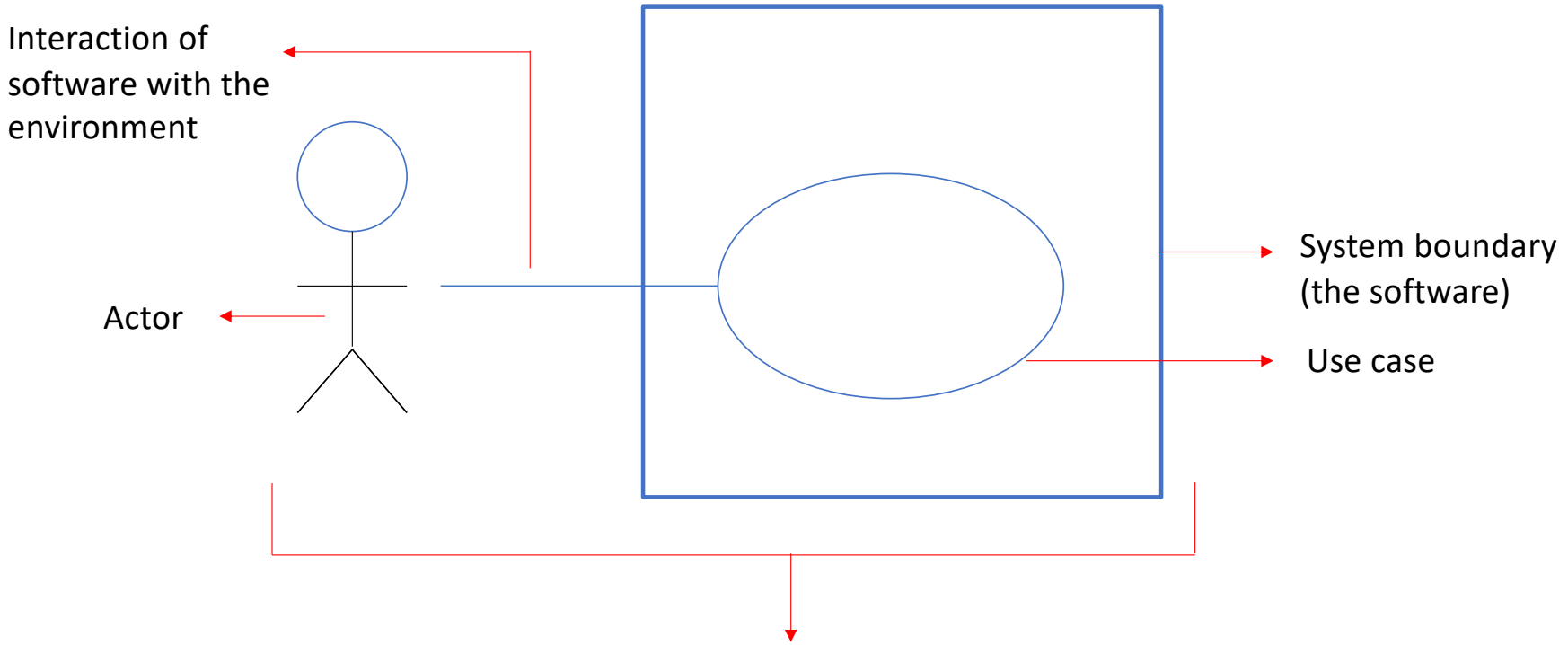


Use case diagram: example

A property agent add a property to the system



Use case diagram: syntax



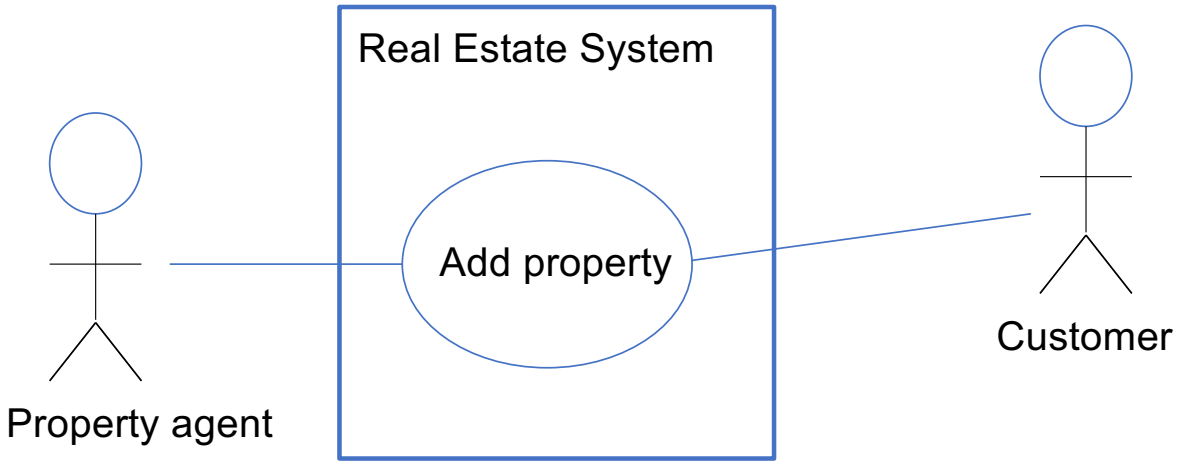
Use case diagram: Actor

- An actor is a member of the world outside the software product
- It is usually easy to identify an actor
  - An actor is a frequent user of the software product
  - Most of the software have more than one type of actor
- In general, an actor plays a role with regards to the software product. An actor:
  - As an initiator;
    - is generally an actor who initiates/start the use case
    - is placed on the left-side of the use case diagram
  - As a participant;
    - is generally an actor who participates in the use case
    - placed on the right-side of the use case diagram
    - one actor can participant in multiple use cases
  - As someone who plays a critical part in the use case
  - As someone who plays more than one role. E.g. a staff can be a lecturer and a TA

Example is in next page

Example of Initiator and participant

- Customer must go to the real estate agent's office with proof of ownership (of the house/apartment) to let the real estate agent add the property to the system for sale.
- Real estate agent is the initiator and customer is the participant



Use case diagram: …of note

An actor need not be human

- Note: this real estate system can never be an actor. Other systems can.

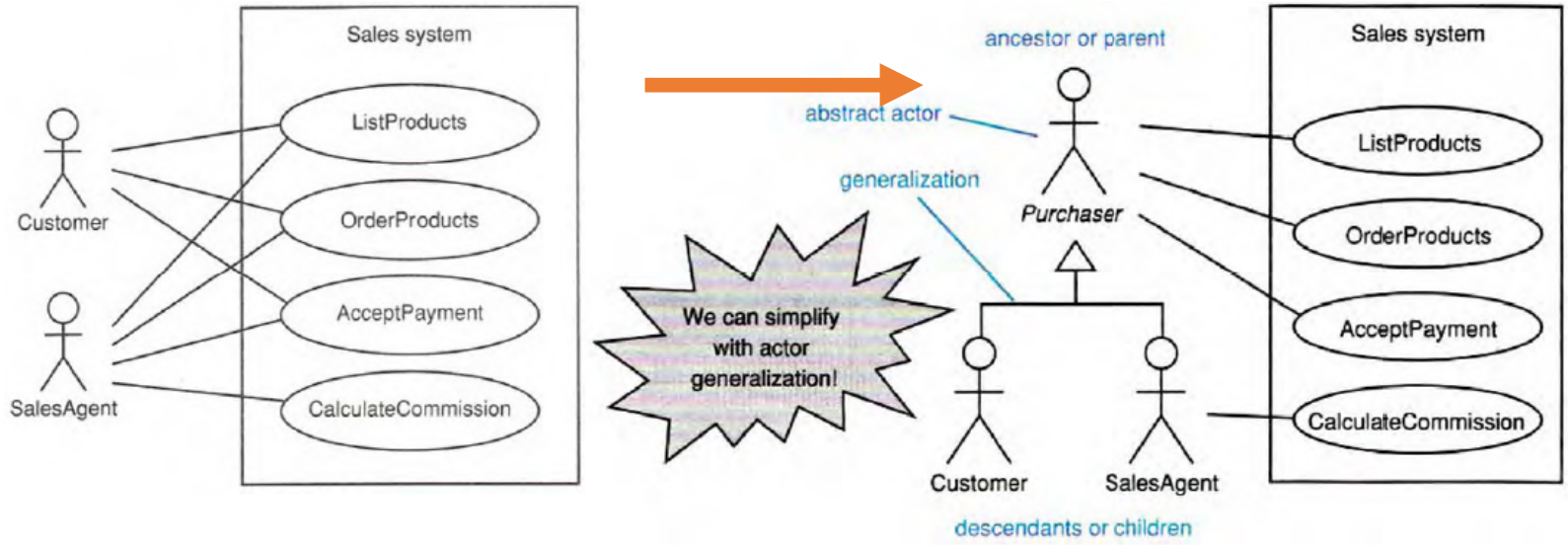
Example: An e-commerce information system must interact with the credit card company information system

- The credit card company information system is an actor from the viewpoint of the e-commerce information system
  - The credit card company is a participant in the e-commerce system use case diagram.
- The e-commerce information system is an actor from the viewpoint of the credit card company information system
  - The e-commerce system is a participant in the credit card system use case diagram.

Overlapping actors (Actor Generalization)

Alternatively:

- If two actors communicate with the same set of use cases in the same way, we can express this as a generalization to another (possibly abstract) actor.
- Generalization (**Inheritance**/type)

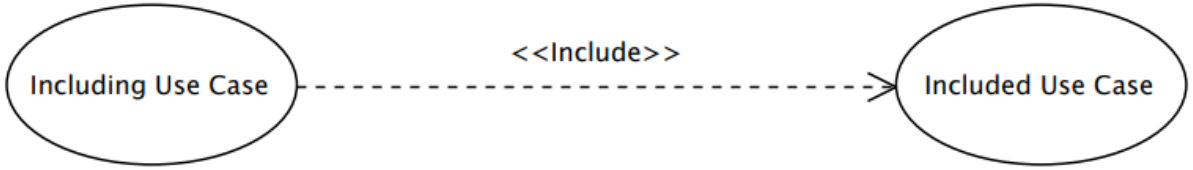


Use case diagram: Inclusion and extension

Inclusion:

- <<Include>>
- The inclusion are compulsory part of the use case.
- An including use case never stands alone. It always includes the included use case

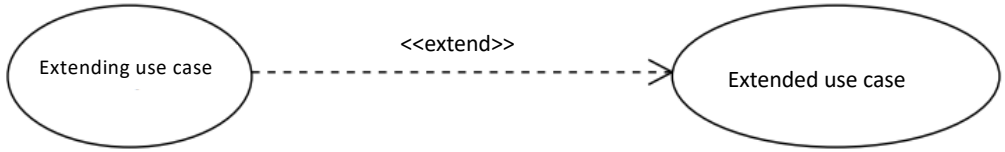
*Including use case: The "main" use case, such as transfer funds
*Included use case: The "sub-use" cases, such as confirm transfer
If you still have difficulty recognising, think about Category/ Subcategory



Extension:

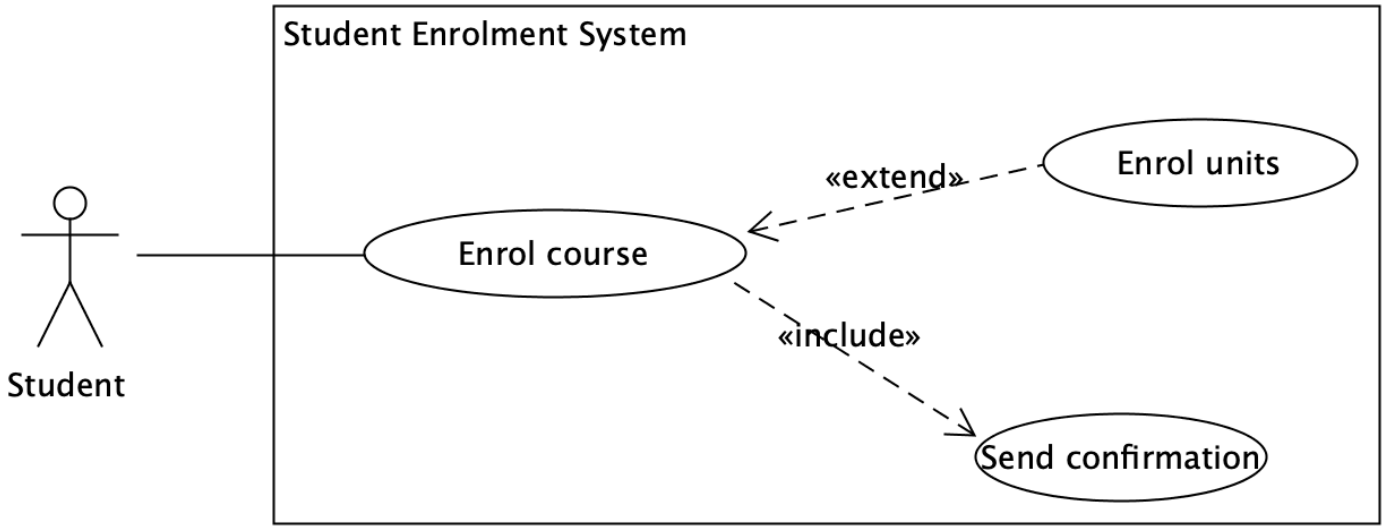
- <<Extend>> is used to separate optional behaviour from mandatory behaviour.
- The extended use case may stand alone, but, under certain conditions, it may be extended by another use case.

*Extending use case: The "main" use case, such as transfer funds
*Extended use case: The "sub-use" cases, such as print receipt



Example: inclusion and extension

- When a student is enrolled, a confirmation is required to be sent to the student
- When a student is enrolled, a student can choose to enroll to the units
- How to find what to include or extend?
  - Check your client's requirements



FLUX : QJN9BW

Multiple Choice

In a use case diagram, what does extend implies?

A it's supplementary (optional) directed relationship

B one of main use cases of the system

C reveals the details about the system that are typically unnecessary

D all of the above

Final Notes

A use case diagram is NOT:

- **A flowchart**!
- A description of the use case.
- A series of steps to define the use case

Use case diagrams should be **simple**!

Iterate and Increment until you get it right!

Note:
Please follow the use case diagram UML syntax described here in these slides. The reference book Satzinger et al. uses a different syntax.

If you are interested, another good example of UML diagramming can be found in the following resource:

Stephen R. Schach, "Object-oriented software engineering" 1st ed, McGraw Hill, Chapter 10, pg 297

COMFORT BREAK

Any questions?

MONASH University

USER STORIES & ACCEPTANCE CRITERIA

IN AGILE, WE WRITE USER STORIES AND ACCEPTANCE CRITERIA TO UNDERSTAND THE DETAILED REQUIREMENTS OF THE CLIENT.

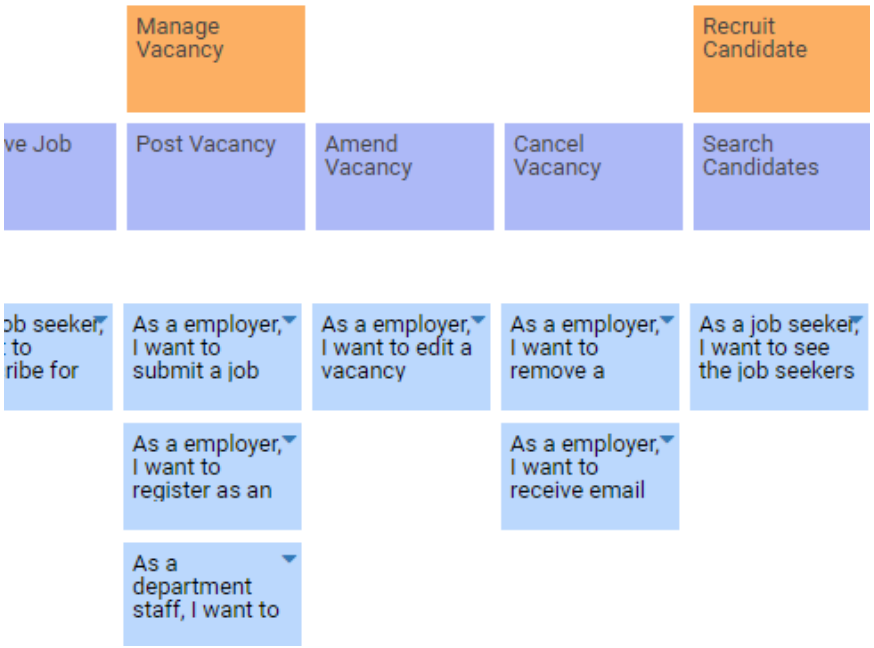
MONASH University

User stories

A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.

The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer.

Source: [Atlassian Agile Coach]



Source: *visual-paradigm.com*

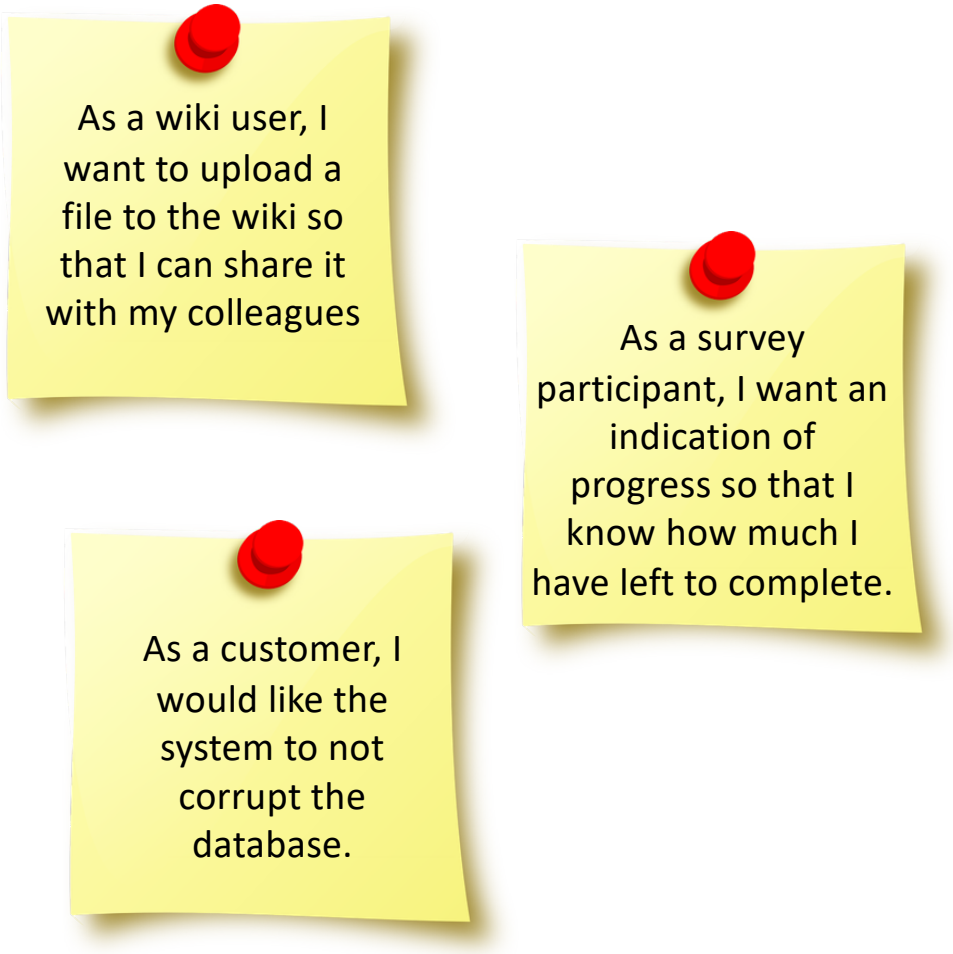
User stories

User stories are succinct descriptions of actions a user would want to do with a system.

They are written with the following structure:

As a <*user*>, I want to <*goal*> so that <*benefit*>

- As a (**who** wants to accomplish something) [user]
- I want to (**what** they want to achieve) [goal]
- So that (**why** they want to achieve the goal) [benefit]
  - Benefit is optional, but is it?



FLUX : QJN9BW

Multiple Choice

As a mail user I don’t want to see multiple scrollbars in text area when composing mail. Is this a well-written user story?

A Yes

B No

C not sure

FLUX : QJN9BW

Multiple Choice

As a user, I want to be able to make emails of different colour depending on the content of the email so that I can see which emails are about complaints and things that I can reply later and are not so important. Is this user story small? Discuss your answer.

A Yes

B No

C not sure

Acceptance criteria

- Acceptance criteria define the boundaries of a user story and are used to confirm when the software is working as intended, i.e., when the story is completed.
- In simple words, the criteria to test that the user story is implemented as expected.
- For a smart home app, one of the user stories could be: “As a homeowner, I want to be able to dim or brighten individual light globes so that I can create my ideal lighting.”
- The acceptance criteria for this user story might be:
    - The user will select a globe and then see this view;
    - This view should use a slider control so that the user has the full range of brightness levels available to them;
    - When opening this view, the initial slider position should reflect the current brightness of the selected globe;
    - The time for a light globe to respond to a change made by the user should be no greater than 1 second.

Acceptance criteria- More Examples*

Example 2.

- As a Customer I want to to check the balance of my bank account so that I can perform transactions.
- Acceptance Criteria
  - Customer logged in before checking balance.
  - Balance for all accounts is displayed.

Example 3.

- As a Customer I want to transfer money from my account to another bank account so that I don't have to go to the bank for making such transactions.
- Acceptance Criteria
  - Customer logged in before transferring amount.
  - System check the receivers account number and validate it prior to performing the transactions.
  - If Ok the local account balance is updated and displayed.
  - System update both accounts concurrently.

* Adopted from: http://groups.umd.umich.edu/cis/course.des/cis375/active/class5/User-Stories-ATM.pdf

To sum up…

- When using agile methodology, teams write user stories and their acceptance criteria
- When using unified process (and most of other models), teams draw use case diagram and write use case description for each use case.
- To write a user story:
  - Use the structure provided
  - Confirm the user story conforms with INVEST principle
  - Write its acceptance criteria

Notes: User stories

- User stories are written when a team is using agile methodology for developing the system.
- User stories should be very short and written from the perspective of the client—they are often required to fit on a post-it note.
- These user stories should be written in the client's language with terminology appropriate to their business.

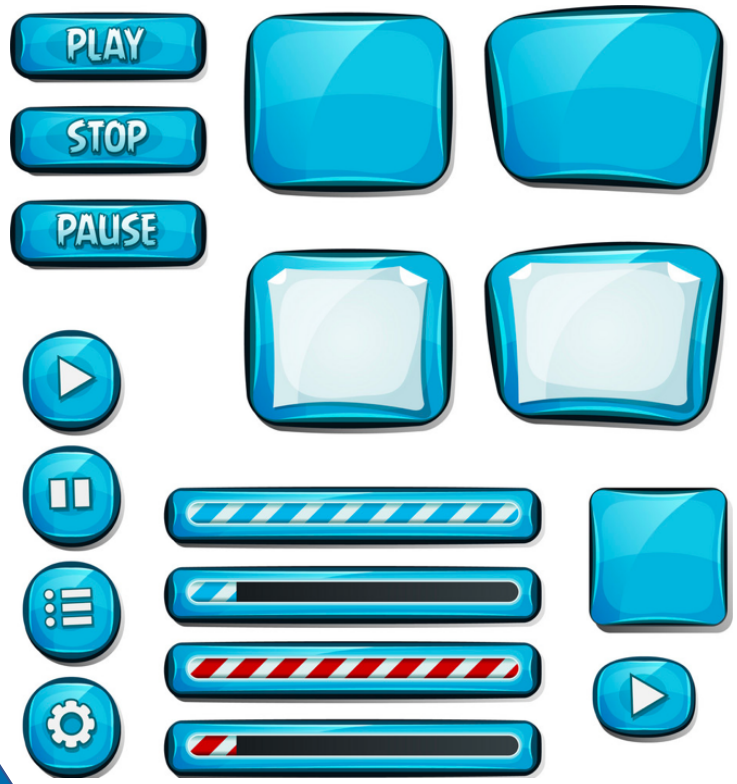
- It is important that the client understands the user stories since Agile methodologies typically involve the client repeatedly choosing user stories to be implemented next.
- For each user story, include an estimation of time required to implement the desired features or size of the user story or a story point related to the level of difficulty or priority.
- Since they are phrased in terms of need, it is usually easy to write tests for them.

Notes: User stories (contd.)

- A handy guide to the production of good user stories is using the INVEST mnemonic:
  - **Independent**: User stories should be self-contained. (Can develop the user story independently by ignore other user stories)
  - **Negotiable**: User stories should be able to be easily revised or replaced, such as with a different user story that satisfies the same outcome.
  - **Valuable**: A user story must be beneficial to end users.
  - **Estimable**: A user story should correspond to functionality for which implementation effort can be estimated.
  - **Small**: User stories must be small enough to be considered and scheduled independently.
  - **Testable**: The user story must provide enough info to verify that it has been implemented satisfactorily. Some tasks or requirements may not be concerned with user-centric functionality. You can either phrase these in terms of the user, write them as developer stories, or just track them internally.

USABILITY DESIGN PRINCIPLES

USABILITY DESIGN PRINCIPLES HELPS US UNDERSTAND THE BETTER WAY TO DESIGN THE USER-INTERFACE OF A SOFTWARE



MONASH University

Before we begin…

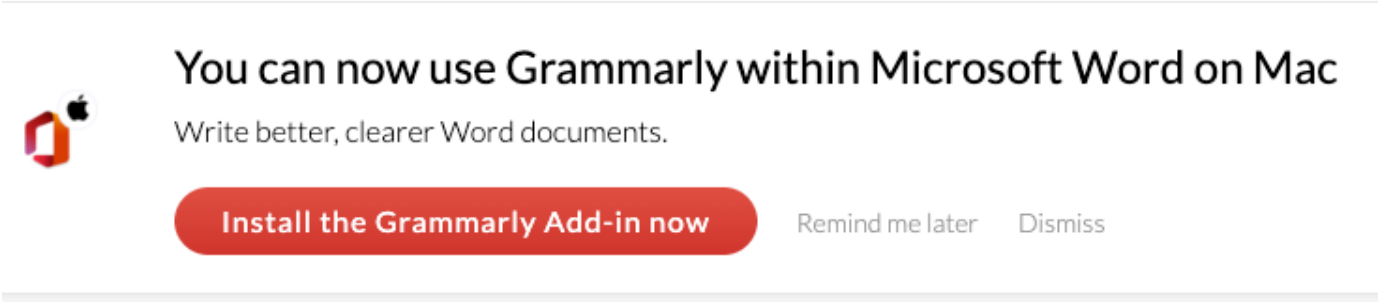
- A prototype is an early sample of a product built to test a concept or process or to act as a thing to be replicated or learned from.
- In the industry, prototypes are mostly the coded screens or wireframes that shows how the whole or most of the software will look like and how the flow of events will occur.
- Types of prototype:
    - **Paper-based**: sketching the screens on the paper. You would imitate how to transit to the next screen
    - **Wireframe**: stitch together paper-based screens with links (arrows) that transition between each screen
    - **Visual design**: using any software to draw the screens. Very accurate and can receive concrete feedback
    - **Coded**: create the real thing. Code the screens
- There are usability design principles or rules that can be followed to design the screens better.
    - Donald Norman's design principle or Ben Shneiderman's 8 golden rules.

Donald Norman's design principles

- **Visibility**
- **Affordance**
- **Constraints**
- **Cognitive aids**
- **Transfer effects**
- **Natural mapping**

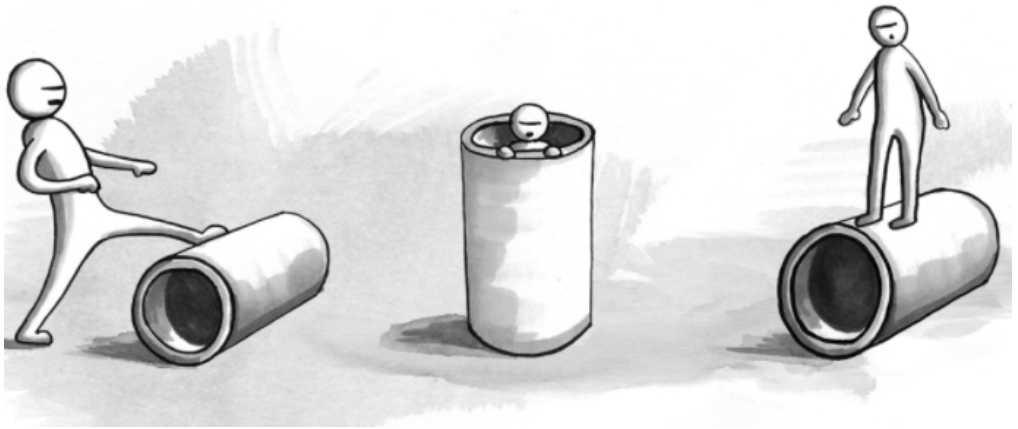
Donald Norman's design principles

(1) Visibility: relevant objects should be in view and obvious to recognise



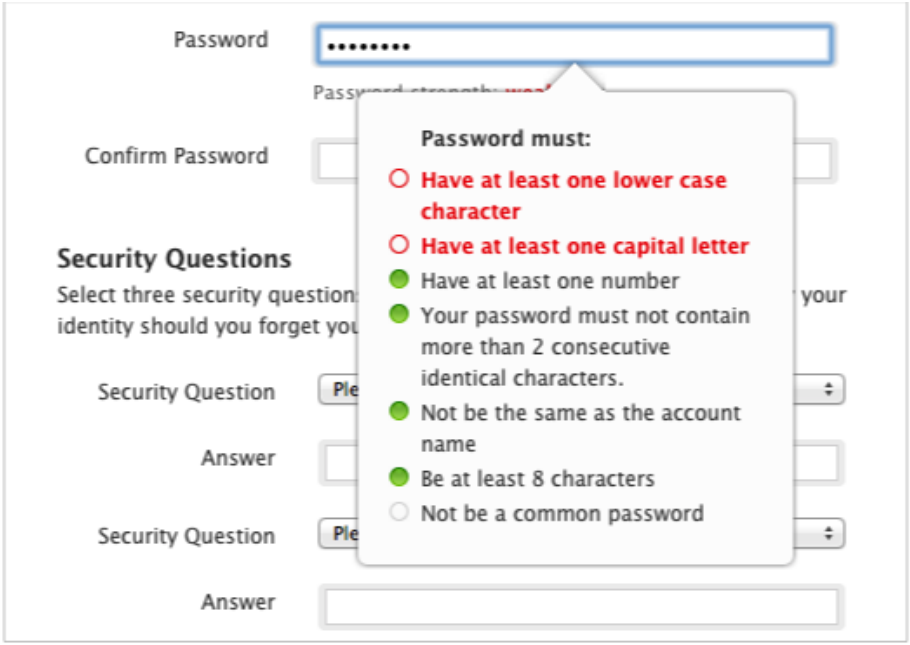
Donald Norman's design principles

(2) Affordance: the appearance of an object should indicate how it should be used



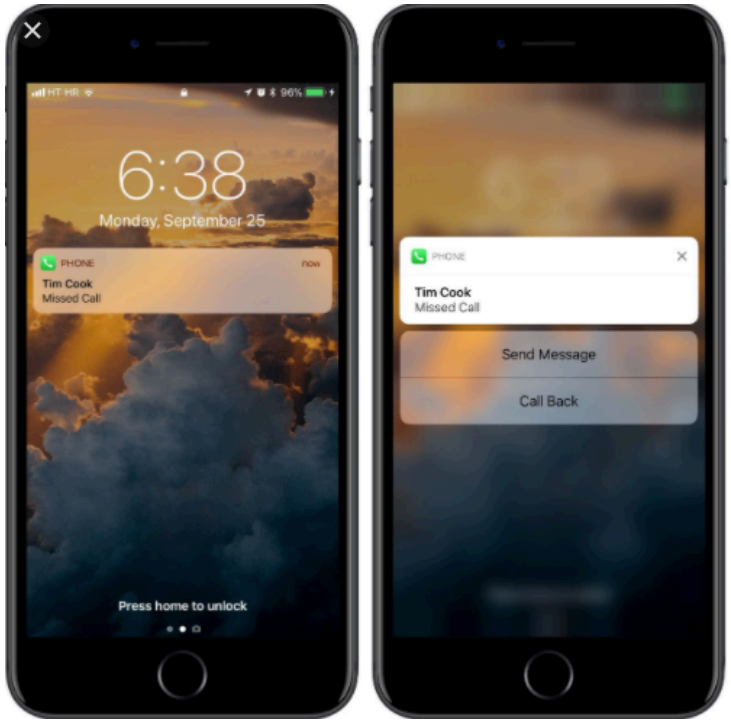
Donald Norman's design principles

(3) Constraints: limiting the possible actions of an object, to prevent user making errors



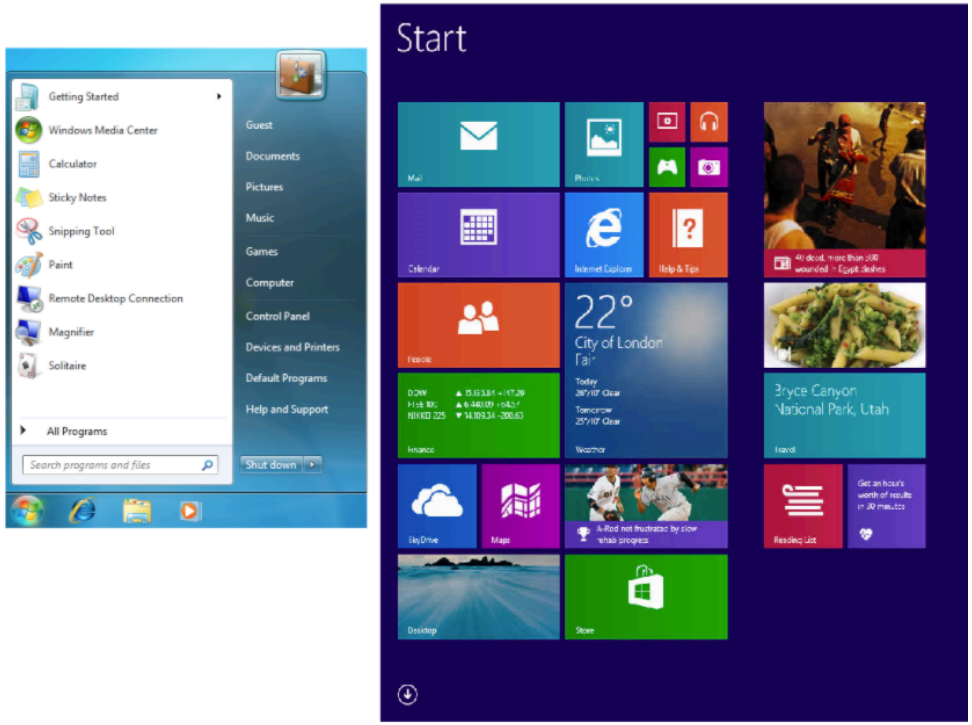
Donald Norman's design principles

(4) Cognitive aids/ Feedback: External representations intended to gain our attention



Donald Norman's design principles

(5) Consistency/ Transfer effects: transferring learning and expectations of similar objects/interfaces to the current task



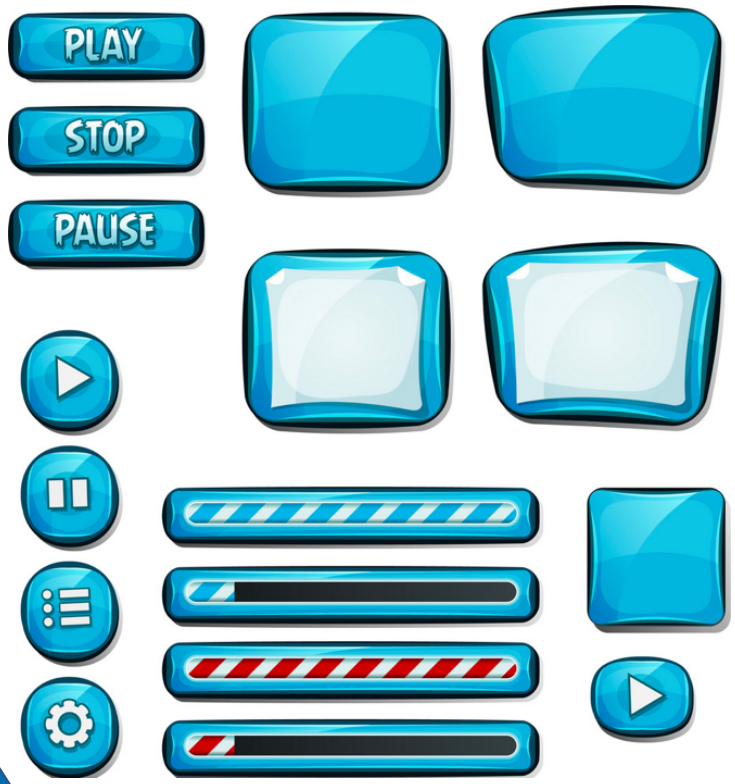
Donald Norman's design principles

(6) Natural mapping: laying out screens to better represent their function

Ben Shneiderman's 8 golden rules

- **Strive for consistency**: consistent user-interfaces
- **Cater to universal usability**: for all range of users (age, disability, etc)
- **Offer informative feedback**: give user feedback on their actions (green, tick, sound, highlighting, etc)
- **Design dialogue to yield closure**: feedback or warnings at the end of the action
- **Prevent errors**: detect error, let the user undo their mistake
- **Permit easy reversal of actions**: offer an easy way out where possible (change of mind, unsubscribe)
- **Support internal locus of control**: give the user a sense of control. Let the user initiate, and control actions
- **Reduce short-term memory load**: Don't make navigation and tasks excessively complex—use meaningful mnemonics, icons, and abbreviations or "hint"

USABILITY DESIGN PRINCIPLES



MONASH University

References

- Stephen R. Schach, “Object-oriented software engineering” 1st ed, McGraw Hill, Chapter 10
- https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33
- https://www.enginess.io/insights/6-principles-design-la-donald-norman
- https://www.cs.umd.edu/users/ben/goldenrules.html
- https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces
- https://en.wikipedia.org/wiki/INVEST_%28mnemonic%29
- https://www.atlassian.com/blog/agile/agile-design-prototype
- “Object-oriented Design course”, Raman Ramsin, Sharif University of Technology, Iran,http://sharif.edu/~ramsin/index_files/undergradcourse_OOD.htm
- http://groups.umd.umich.edu/cis/course.des/cis375/active/class5/User-Stories-ATM.pdf


Thank you!