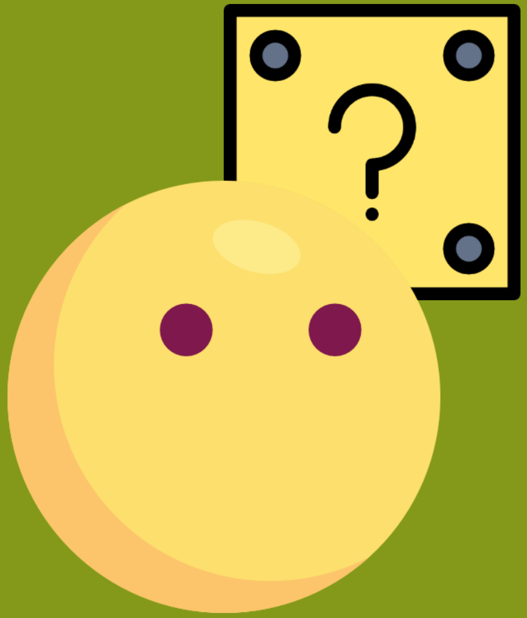
SOFTWARE ENGINEERING

Chapter 2: Software Requirements

MOTIVATION...

The correct management of a Project is all about **leveraging your resources, working around the problem, doing what’s right** and the most important thing here is to make sure that **everything is controlled!**

What are software requirements?



Requirements

- Descriptions of the services that a system must provide and the constraints under which they must operate.
- They are divided into levels and each of those levels has different complexity layers.

WAIT A MINUTE!

Let’s bring some cases to the table!

Cases!

1) Pet shop!

- The pet shop ‘my first pet’ is interested in an information system that allows them to manage their business.

2) Coca Cola...

- Coca needs to display his products with an interactive advertisement strategy!

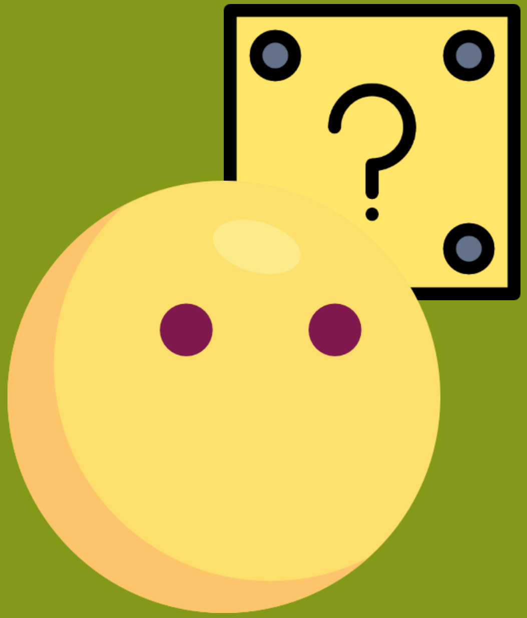
3) CLEI

- The scientific conference CLEI needs to provide information about the event in real time

4) PC Gamer Magazine

- PC Gamer magazine is interested in the newest development trends of AI, they want to recommend games based on user’s data.

Could you define those levels?



Requirements levels PT I

- User requirements: Are all the services that the system will provide

- System requirements: Detailed functions, services and constraints

So… let’s give an example

As user requirements and system requirements are not always that easy to differenciate from each other

User Requirements

- The app ‘my first pet’ should allow users to buy a new pet and deliver it to the door of the customer’s house.

- The system of ‘my first pet’ should allow users to control bills, pays and all the financial stuff.

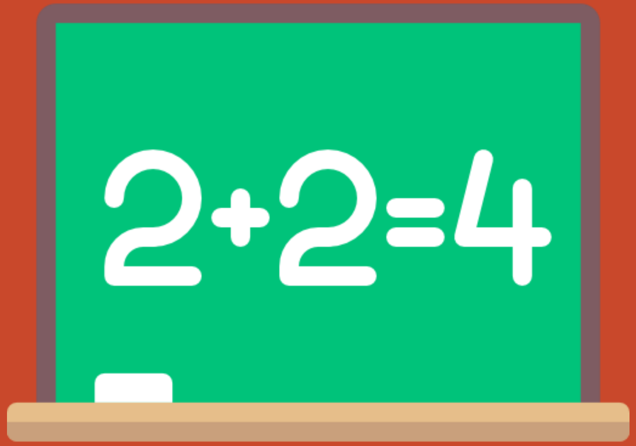
User vs system requirements

- The system will provide a form asking for customer's information such as name, ID, age, address, and a photography.

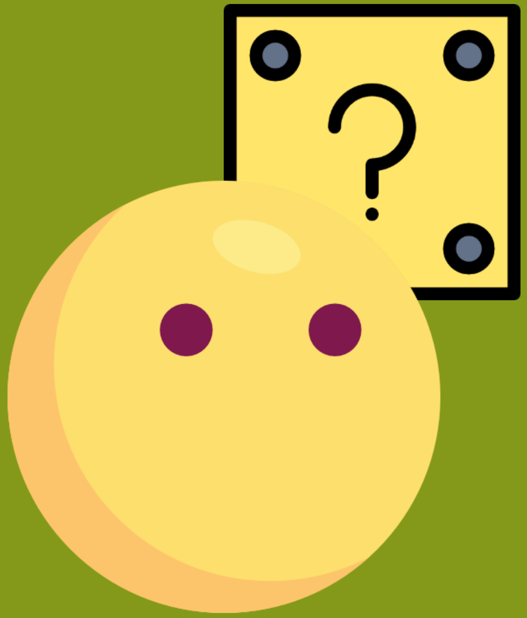
- The information is sent to the server and it'll be indexed by user ID. Another member of the shop will recieve the information and validate it by using *ONIX*, after validation is completed the system will provide the checkout to the customer.

Time for some board class...

Debate excercise!



Is there any other type of requirements?



Domain specific requirements

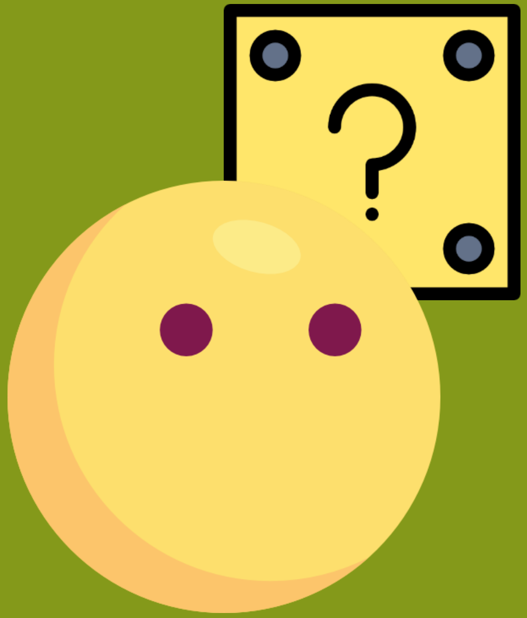
- Actually yes... but depends on the domain, this Domain Specific requirements are mostly constraints.

- They model, prioritize data or can be as specific as math formulas.

Some examples

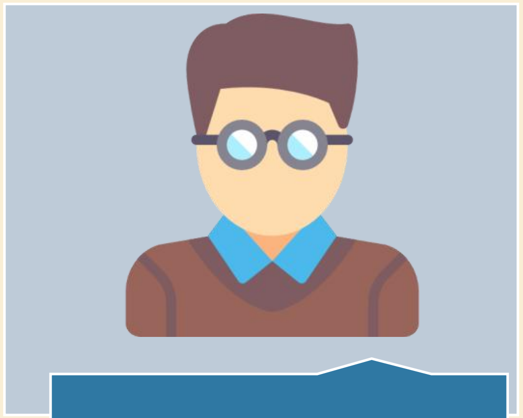
- The CV of every researcher involved in the CLEI Event must display publications ranked with the SCIMago standard.
- Games should be categorized in genres [a,b,c...] and subgenres [e,f,g...] in order to collect that data for the intelligent system.

Is there anyone else involved ?



Stakeholders

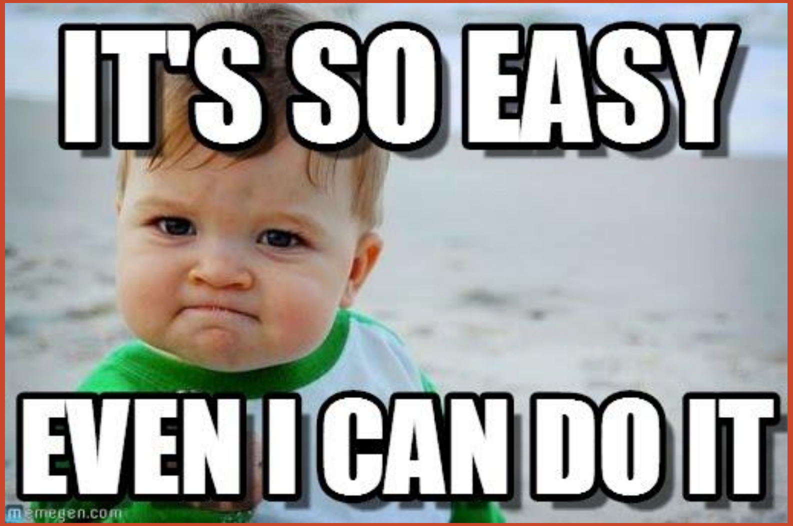
- The answer is yes!, all of those persons are called stakeholders


Final user

Administrators (Superusers)

Client

Development team



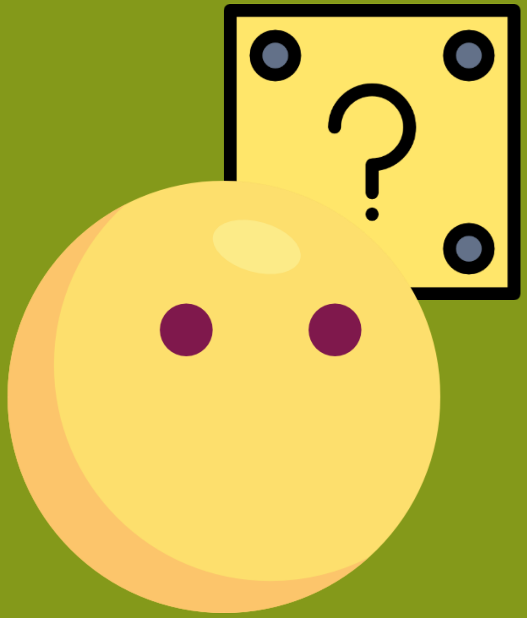
It doesn’t look too difficult right?

Some Stakeholders for the CLEI Project!

- Development team
- Client
- Chairs
- Administrators
- Researchers
- Students
- Teachers
- The university itself
- The scientific entity of the country

And we can have more like… research pairs, partner universities

Are there more deffinitions?



Functional requirements

- These are the core of the system:

Statements of services

System Outputs!

System behavior

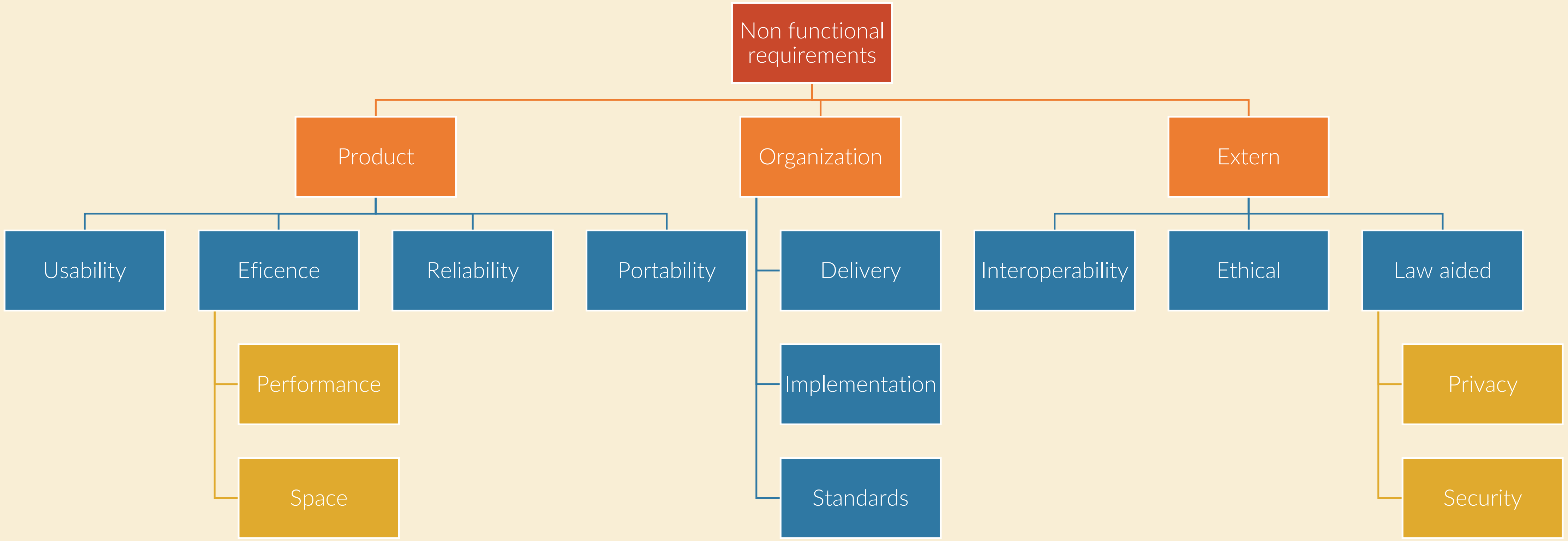
What to avoid!

Non-Functional requirements

- These are mostly constraints:

  - Standards
  - Law
  - Time
  - Development process
  - Individual features

These are all the non-functional requirements

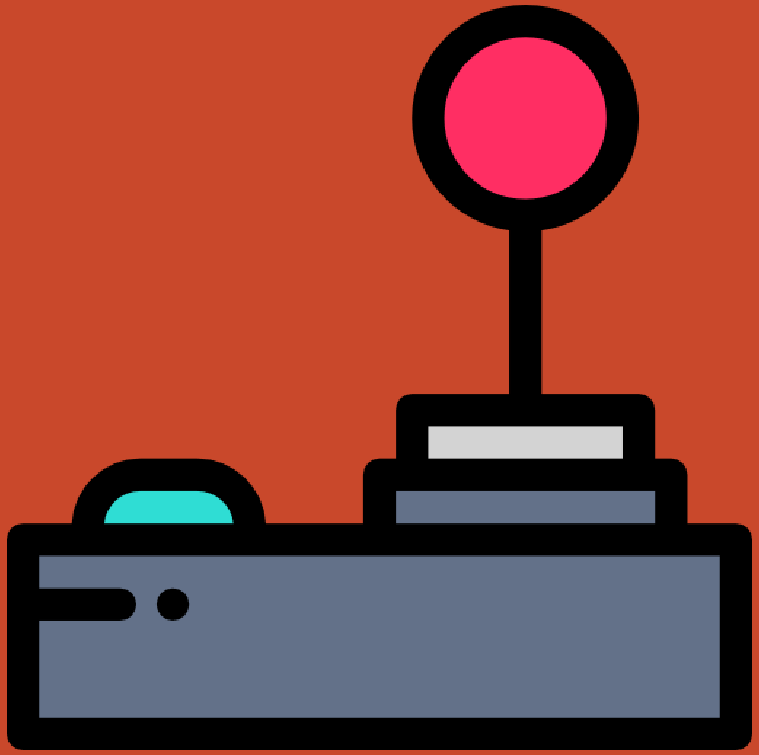


Metrics for requirements PT I

Property	**Measure**
Speed	# Processed transactions User / Event response time Screen refresh time Timeloads
Size	Kbytes Number of RAM chips
Ease of use	Training time # of Help screens
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability

Metrics for requirements PT II

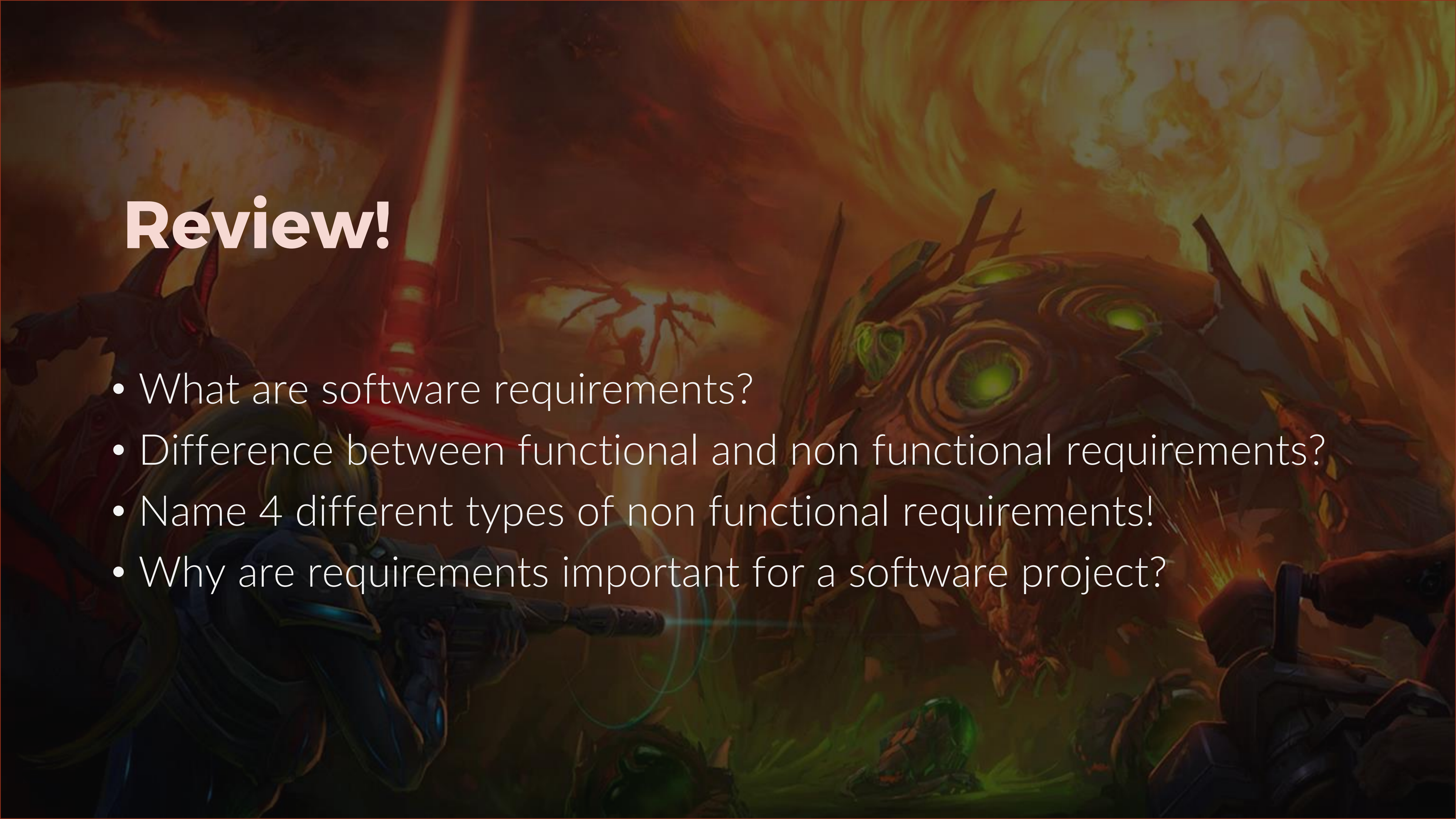
Property	**Measure**
Robustnes	% of Events causing failure Time to restart after failure Probability of data corruption under failure
Portability	% Target system dependencies Number of target systems



Excercise!

Let’s put the understanding of those requirements to the test!

One Shot Review



Review!

- What are software requirements?
- Difference between functional and non functional requirements?
- Name 4 different types of non functional requirements!
- Why are requirements important for a software project?

How am I supposed to get all of those requirements?

Elicitation!

Interviews

Meeting the stakeholders

Feasibility studies?

Interviews and Meetings

- Closed or Open
- Not always reliable
- Clients are not always available!
- *They need to point out needs and features!*

*The way you handle this, depends a lot on the methodology that has been chosen for the Project!

Let’s try them out...!

Again with the groups, the teacher will be the ‘client’

A

B

C

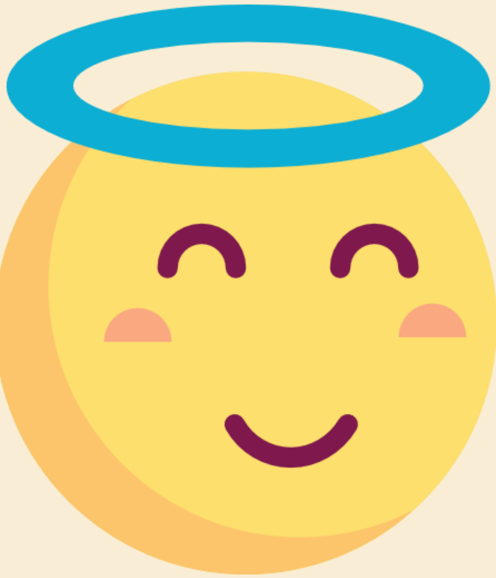
D

A

B

C

D



A

B

C

D

A

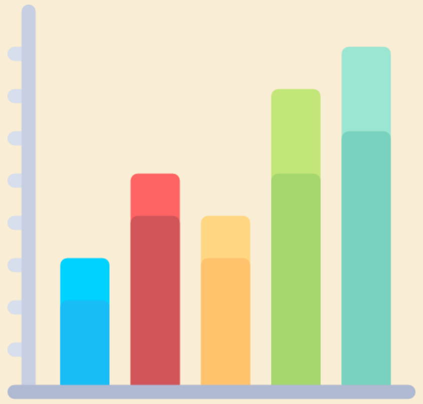
B

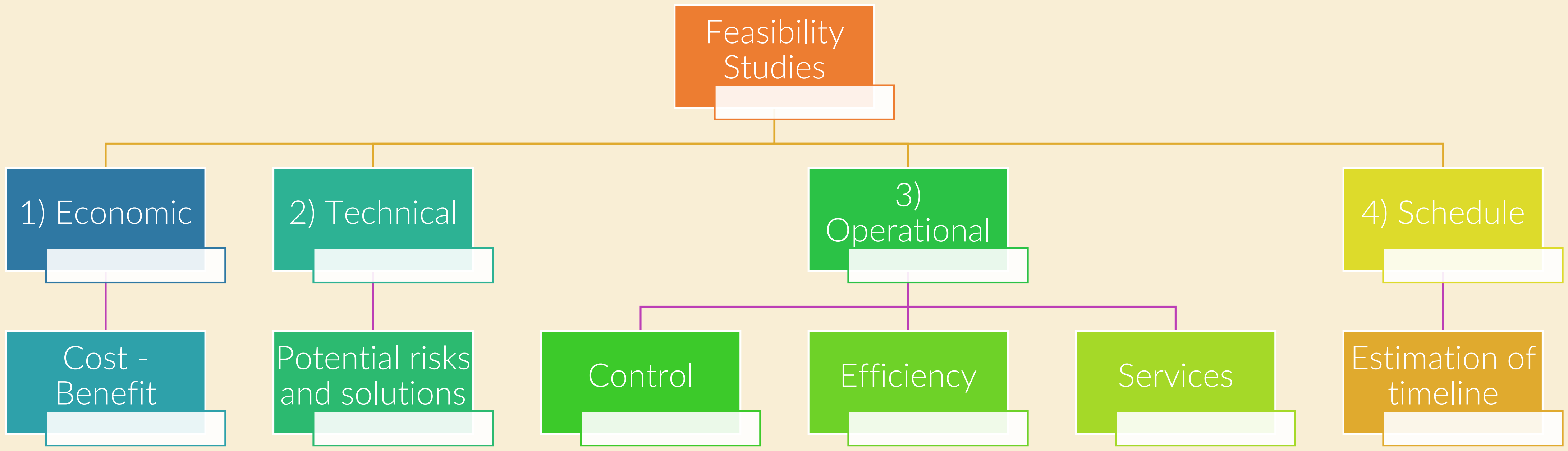
C

D

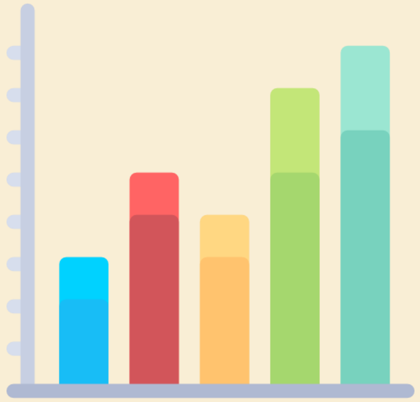
Moving on...!

Feasibility studies





Feasibility studies



1) Is it mandatory?

2) Does the Enterprise have any other software system?

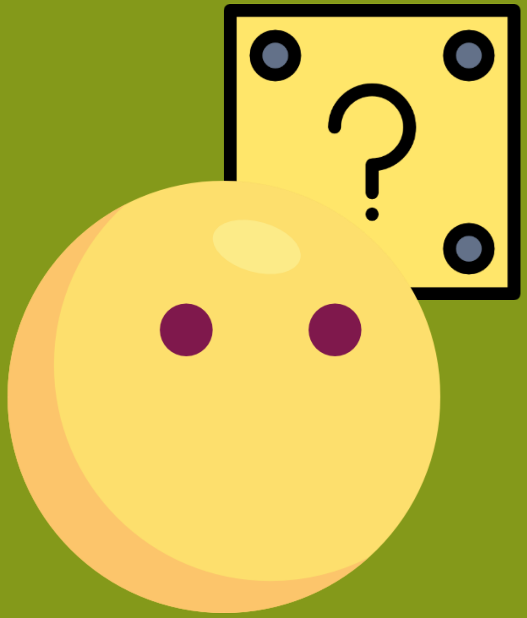
3) What would happend if the system is not built?

4) Can I build the system in that time with those specifications?

Remember that…

The client doesn't always know what he wants! And he/she assumes that the team has his same domain expertice!

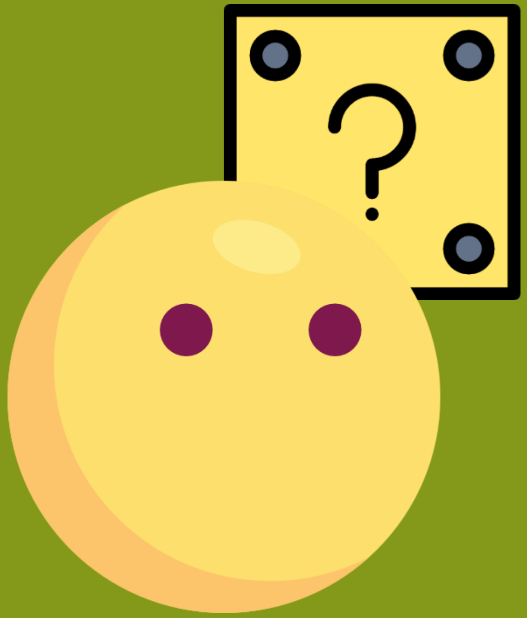
What Can go Wrong?



Traps

- High level concepts are not requirements!
- UI screens are also not requirements
- Requirements are just focused on functionality
- When Requirements are not verifiable
- Developer needs to ask too many questions

What should I do with all of this?

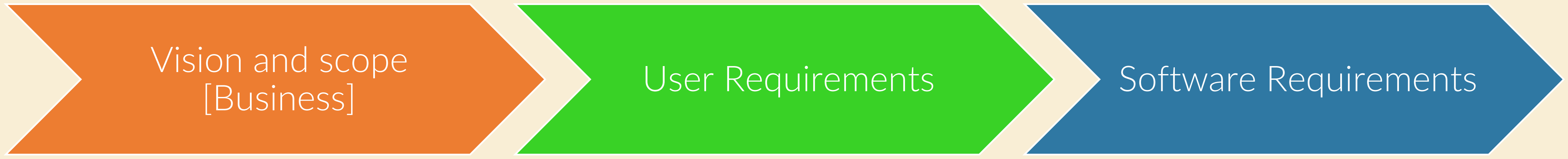


Requirements Engineering

Prioritizing layers…

Can come Handy at any Project! 😊

Prioritizing Layers!



- Use Stories

- Functional Requirements
  - System requirements
  - Business Rules (or domain)
- Quality Attributes
- Non Functional Requirements
- Constraints

The requirements document!

- Usually called SRS is the main scope for your Project, it’s built in this phases

1. Introduction
2. General description
3. Requirements Specification
    1. Natural language
    2. Use cases
4. Architecture
5. Design
6. Models
7. Evolution

1) Introduction

- Purpose of this document
- Project’s Limit
- Definitions, Abreviations, Vocabulary
- References
- Description of the rest of the document

2) General description

- Product perspective
- Product functions
- User's description
- General constraints
- Supositions and dependencies

3) Requirements Specification

- They can be done in natural language!
  [TIT, RAT, DESC, DEP]
- Remember that they need to be verifiable, and they need to be well classified.
- You can use notations of SM and QM (System requirement and Quality requirements) or put titles with the classification given in sommerville. [Slide 23]

Let’s put some examples!

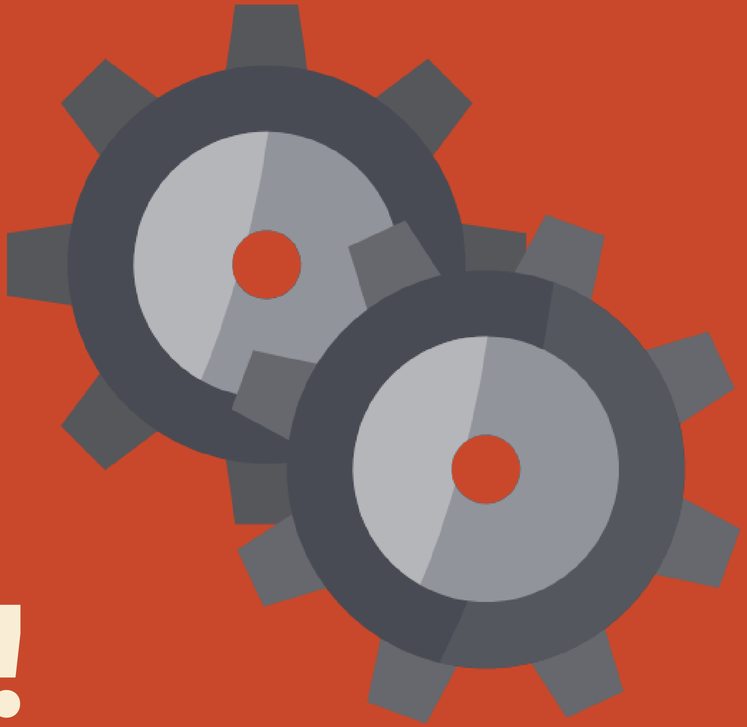
Of requirements in natural language

Natural Language Requirements [Examples]

- *TIT*: SM3 or FM3
- *RAT*: In order to collect data from the video
- *DESC*: The system should read the user’s location by using the 2D camera and recognize features such as speed, head’s location and arms location.
- *DEP*: SM3

Natural Language Requirements [Examples]

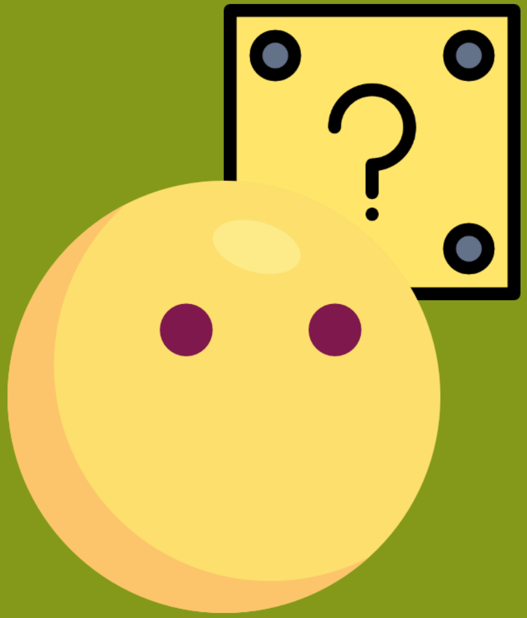
- *TIT*: QM3 or NFR3 [Implementation]
- *RAT*: In order to assure that all the components can blend together
- *DESC*: The implementation language must be C++, using the library OpenCV for video-processing and in order to assure an exportation for Unreal Engine
- *DEP*: QM2



Something else!

In that part of the document is where you're going to put use cases and user stories!

What are both of them?



Use cases

- Are a really useful tool for requirement’s specification!
- Descriptions of related domain processes
- This is like the ‘narrative’ version of the functionality of the system, it’s like writing stories!

Former definition!

Is an elicitation technique based on scenarios, they're a main feature of the UML that is now used to describe models of Object Oriented Systems.

That doesn’t tell too much!

Decomposing use cases

To make it clear let’s define some key elements like:
1) Scenarios, 2) Actors, 3) UML and finally again 4) Use cases!

1) Scenarios!

- Talking with real things about the product is more useful than long conversations!
- Scenarios are sketches of the interaction

1) Scenarios!

1) Expectations of the system and about the user when the interaction begins.

2) A description of the normal flow of events

3) A description of things that can go wrong and how to handle them

4) Simultaneous tasks

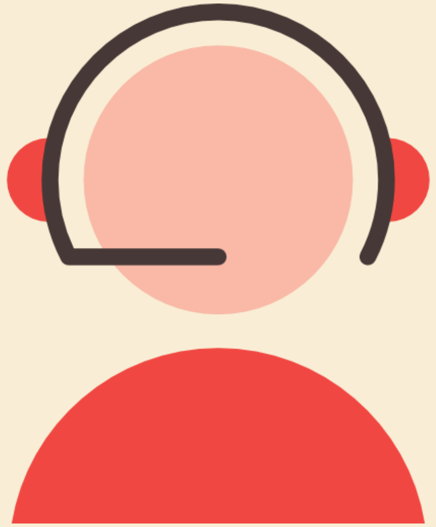
5) System's state (when it starts, when it ends)

1) Scenarios!

- On an interview think about!:

Hey, let’s try to put into paper how the interaction may work!

Okay…



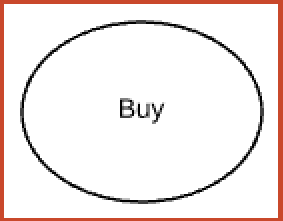
1) Scenarios!

Or just think about those big activites that your software will offer e.j:

1. Buy, *how?*
2. Sell, *how?*
3. Display Information, *how?*
4. Manage busines information!, *how?*

We actually don't care about those how's at this point!

You draw them like this!



Hello!, I’m an use case 🙂

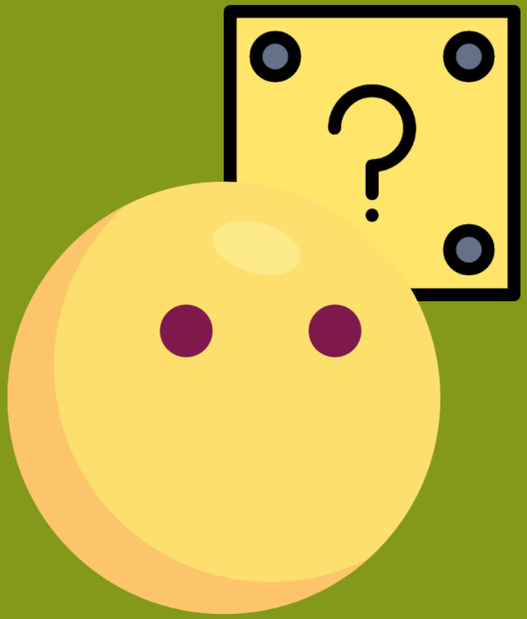
But you were talking about scenarios!

Pssht!

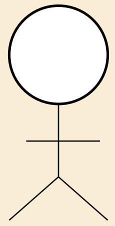
In a lot of cases, scenarios can be seen as the title of the use case!

Or a given use case can have multiple scenarios!

What about actors?

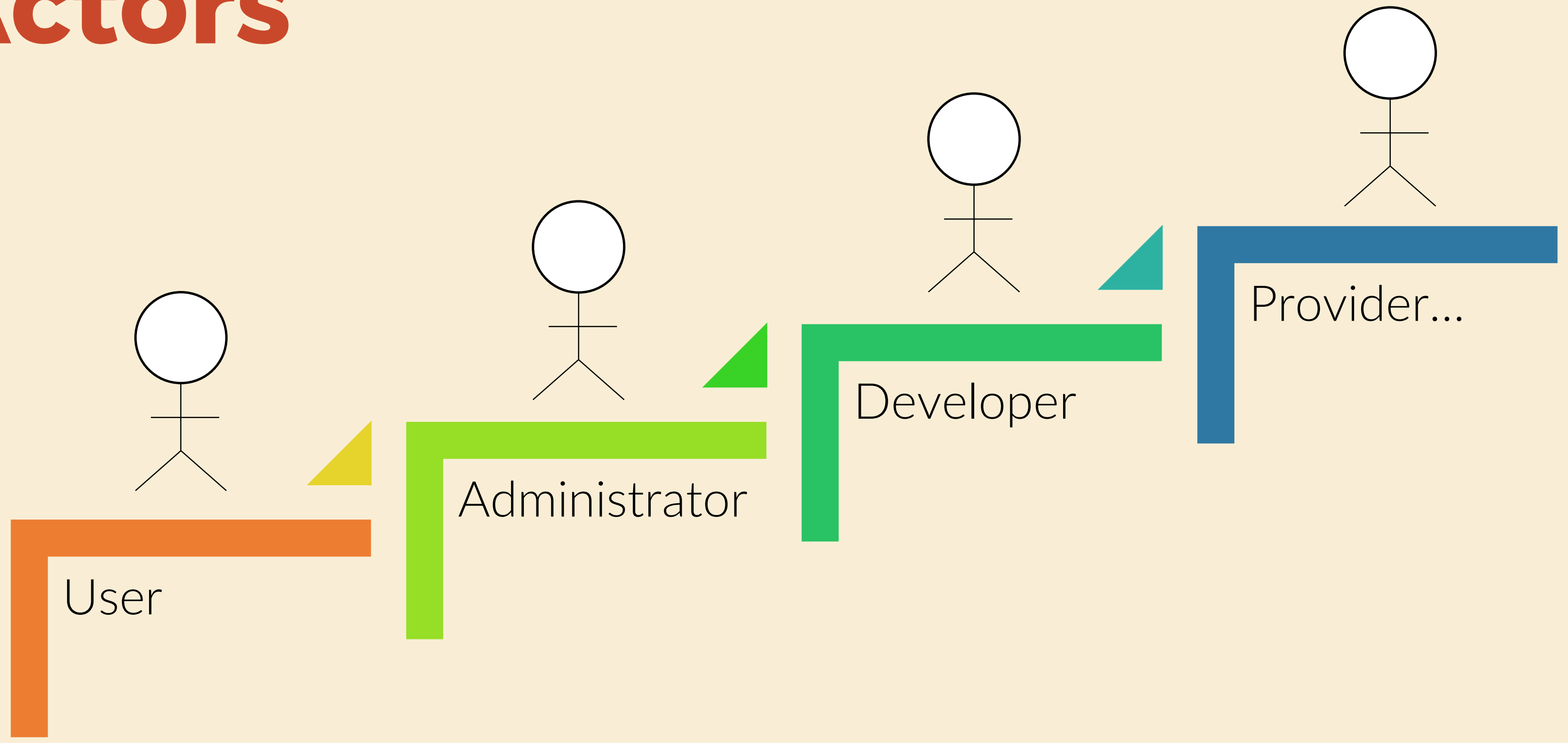


2) Actors

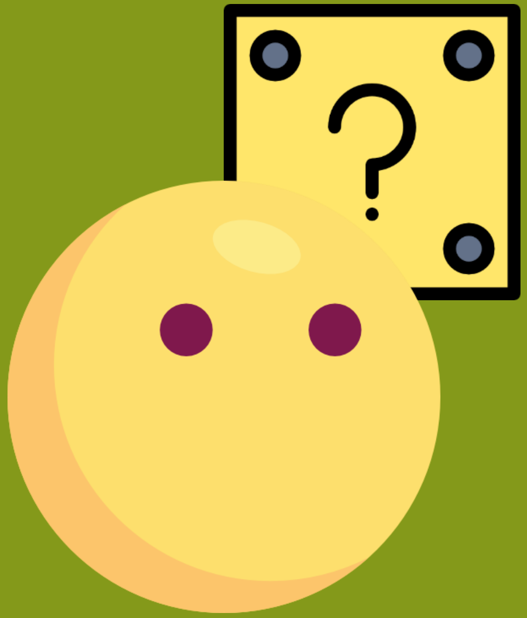


- They are the ones who interact with the system at a given scenario.
- They are the ones who ask for outputs of the system.

2) Actors



Why those drawings?



3) UML

- The reason is simple, and it’s the standard UML!
- It’s a language that all of us will learn to speak, as it’s easy to understand and gives clarity of every single software process like design and requirements engineering above all.

Former Deffinition

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [LARMAN]

4) Use Cases and actors and UML

- Remember that they represent the flow for each scenario.
- You can have an overview of each system and subsystem by using use case diagrams!

Example!

The Circus ‘Charlie’ wants a system that allows them to control every single aspect of their business.

Circus Charlie pt I

- They want to manage their animals: training, feeding, sleeping and overall health.
- They want to manage their business (incomes, outcomes).
- They want to manage their Schedule with ease.

Circus Charlie pt II

- Who will use this system? (Interactors)

Artists

Vets

Trainers

Admin (Data)

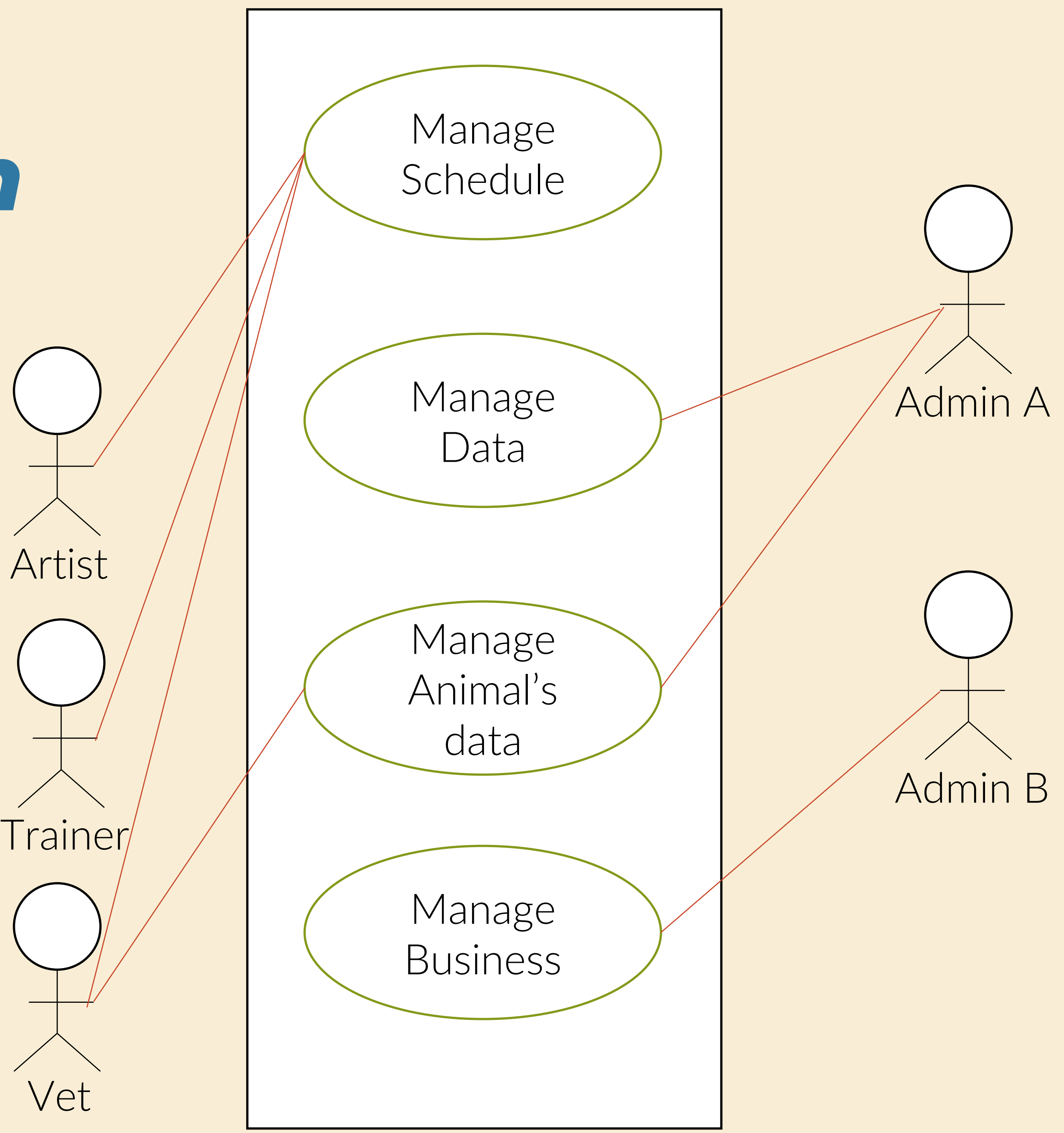
Admin ($)

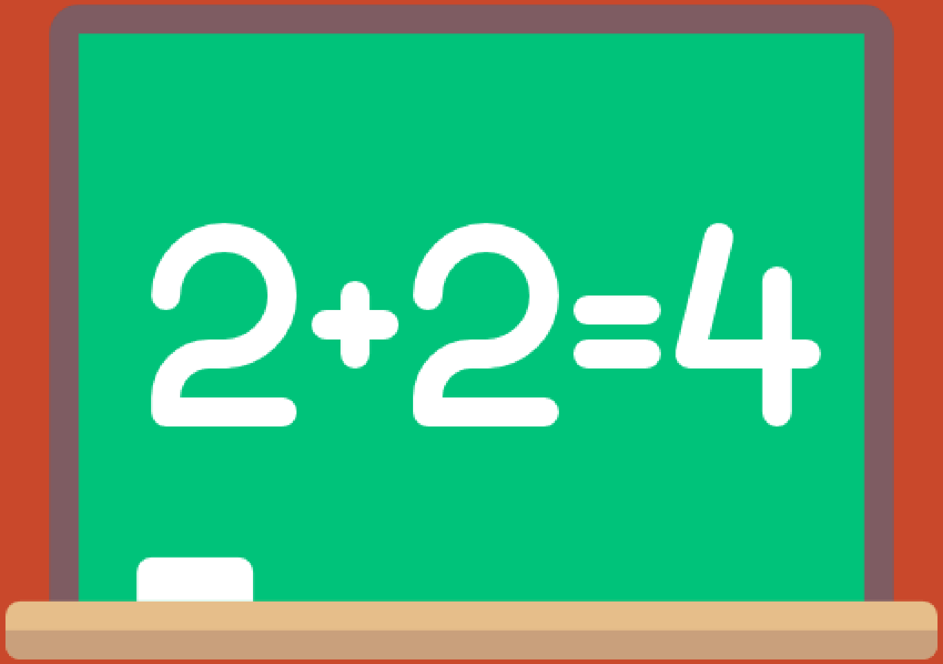
Domain Model
Use Case Diagram

Just remember:

- What activity can any of these actors do!

We don't want to consider unnecessary functionalities for the system!

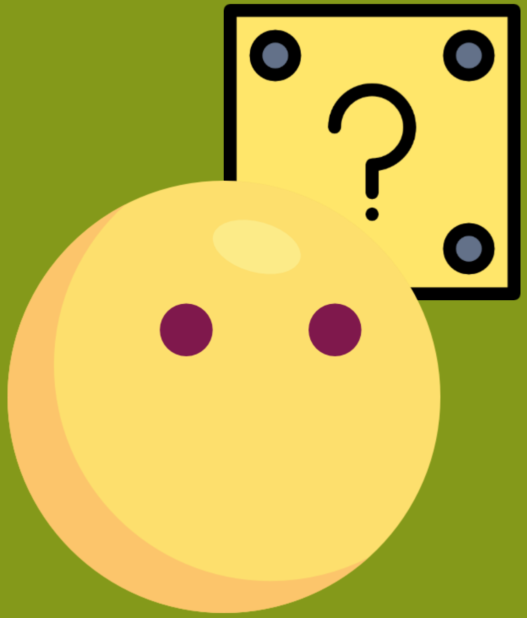




Let’s draw it in the board!

And consider every single element of the system

Will we return to use cases?



4) Use cases

- Fill this table!

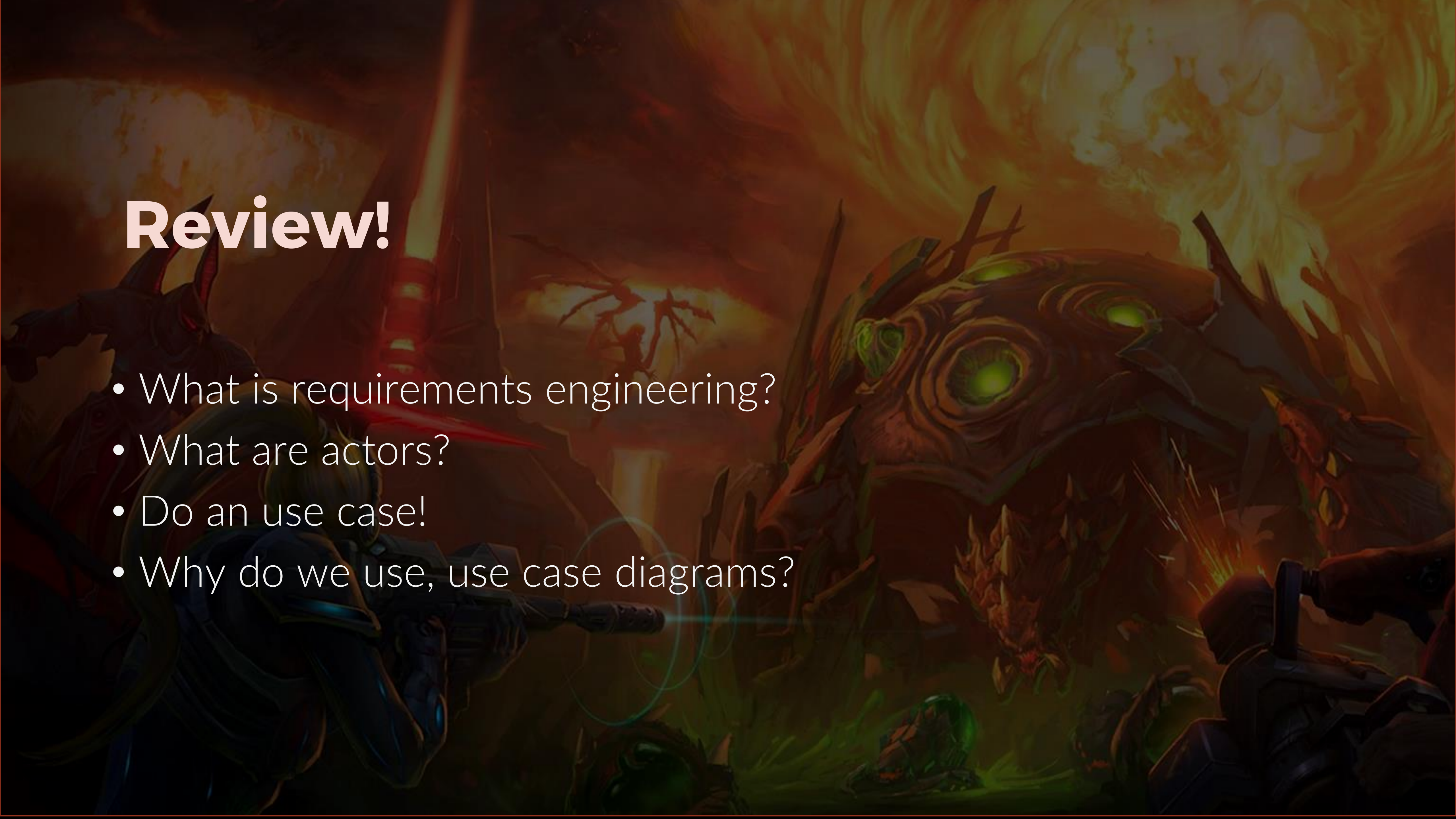
Use case title:
Description:
Actors:
Pre-Conditions
Post-Conditions
Normal Flow
Alternative Flow

Use case title: Add a new Animal
Description: This use case shows how to add an animal to the database
Actors: Database, Administrator
Pre-Conditions: Administrator must be logged in, Internet connection must be detected, Server turned on and services enabled.
Post-Conditions: A new animal is added to the database
Normal Flow: 1) The system asks to re enter the admin password 2) The user enters the admin password 3) The system asks for the permission or ID of the new animal 4) The user enters the ID 5) The system displays a form asking for, type (wild, domestic), name, age, health status, diet, type of act, description of the act and a tentative Schedule for feeding, Schedule for training and for healthcare. 6) The user enters all the requested data and accepts. 7) The system sends a request for the vet, the artist and the trainer to accept the Schedule, and it registers the new animal.
Alternative Flow: 2a) The password is wrong: Return to the normal flow step 1 and count +1 for the errors. 2b) The password is wrong 3 times: Break the use case. 3a) The characters for ID are wrong: Count the error +1 and return to the step 3, if it's wrong 3 times, throw an error and break the use case.

4) Use cases

- And from the example we can have a lot more of alternative flows.
  - What if there’s no internet while the system tries to send a request for the vet, the artist and the trainer?
  - What happend if the connection suddenly closes.
  - How to handle the wrong data error, can we try a generic use case for that?

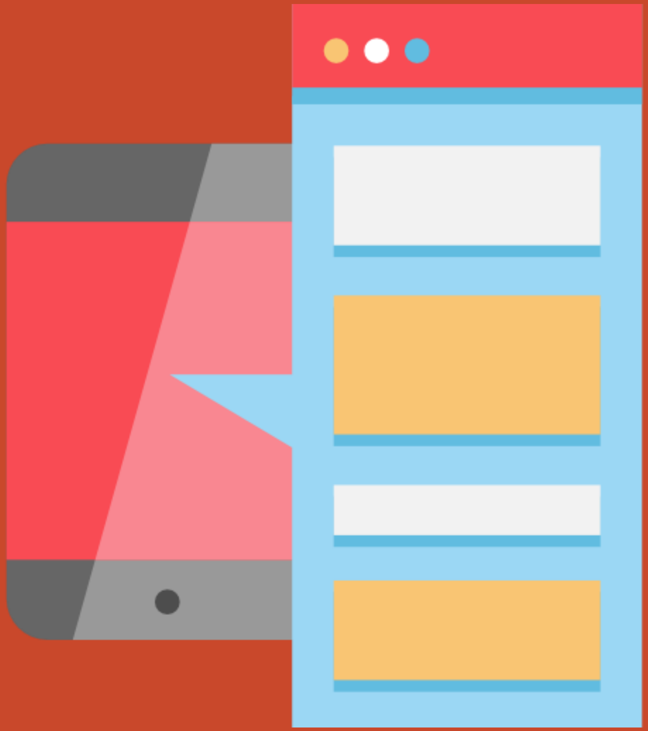
One Shot Review



Review!

- What is requirements engineering?
- What are actors?
- Do an use case!
- Why do we use, use case diagrams?

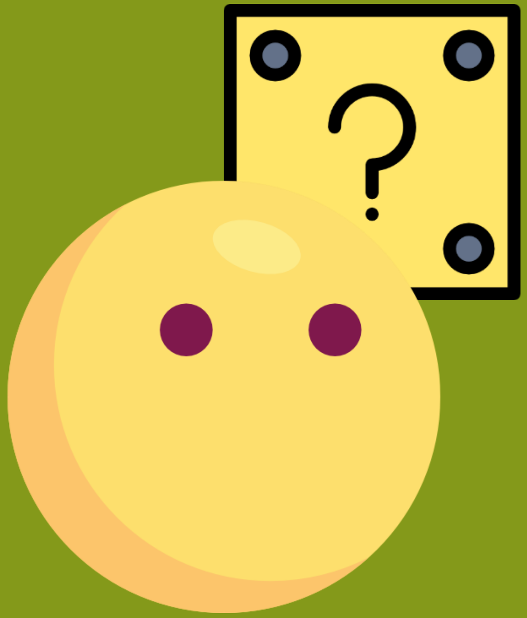
Will we ever finish this chapter?



Yep!

The only things that are left are sketchy techniques and some advice into all of this!

So... is there something easier than an use case?



User Stories

- They’re a lighter version for requirements specification
- Widely used in agile methodoligies
- They Tell a story of how anyone will use the product

User Stories

- They tell specific functionalities of what can be valuable for certain customer or stakeholders
- They're like a conversation… so… Get Creative! 😋

They’re actually post-its

So it’s fun to use them as a tool for specifying a Project and to find requirements

User Stories

Schedule feeding:
As the circus vet, I want to scheedule the feeding times of the animals!

→

Tracking Diet:
As the circus vet I want to track the diet of my animals by knowing what the eat in their meals

User Stories! - Comments

- Functionalities can start to appear with ease in the conversations
- Hidden functionalities can start to also appear, and elicitating functional requirements can be easier with this technique!

User Stories!

- Add acceptance criteria to your stories to make them verifiable and more specific!
- Try to use 3 to 5 criterion to make the stories more consistent

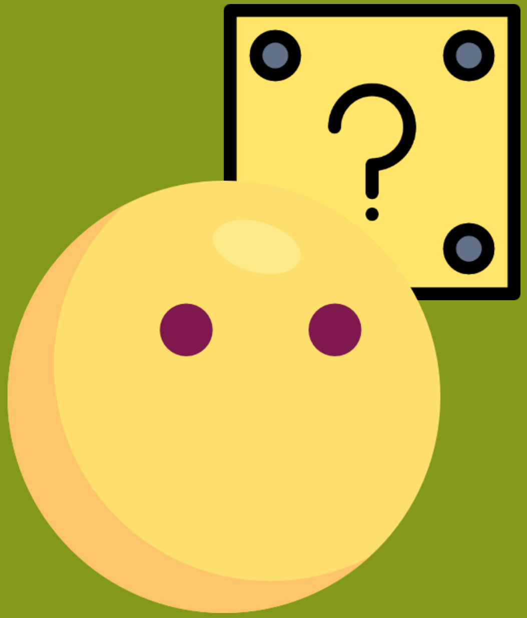
Acceptance Criteria

Schedule feeding:

- Date and hour must not have been scheduled by any other person
- The Animal and the plan must exist in the system
- System must be in an active state

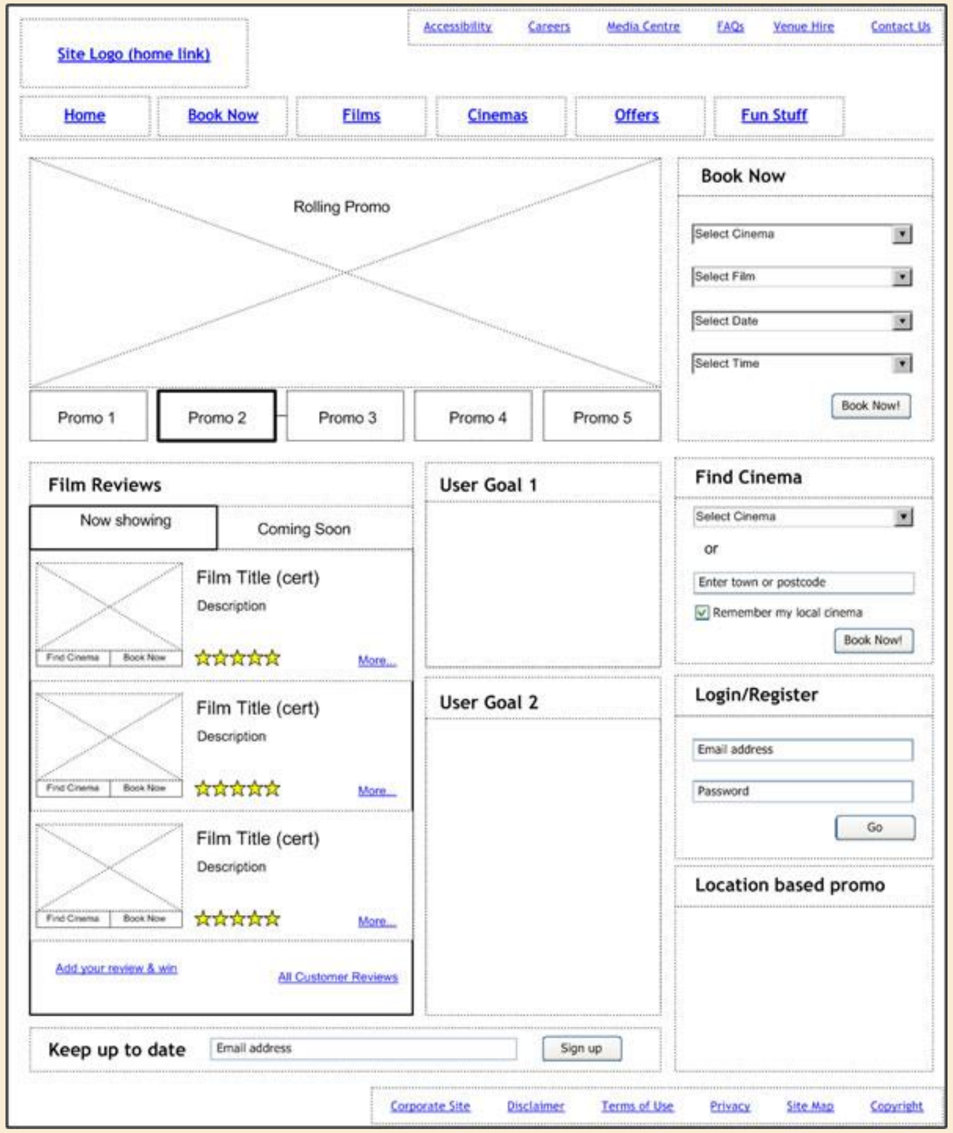
You can avoid unnecesary discusions by using this kind of technique, it's really Handy!

Something else?



Wireframes!

- They show how you're going to dispose the elements of the interface.
- They're really sketchy, but fast to do and not expensive (time and money)



Wireframes! *[Cost: $]*

- They can serve you to start the UI design, also to validate how many frames will any desired activity take.

- They're Handy to validate layouts, forms, composition…

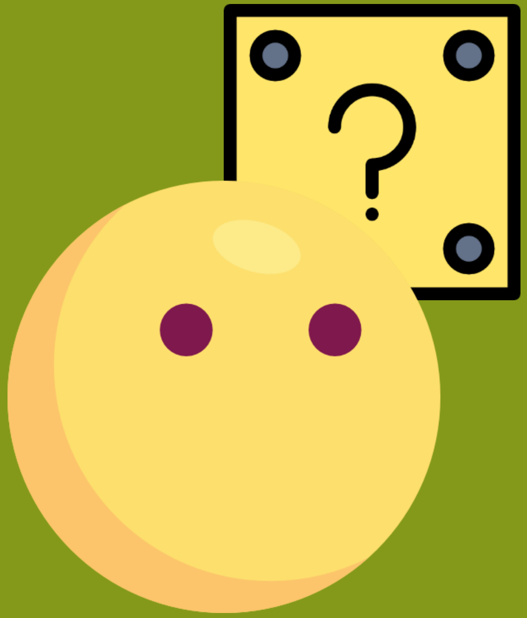
Mockups *[Cost: $$]*

- They're like the wireframes but they can:
- Include the content
- Include design and layout details

Prototypes *[Cost: $$$]*

- They’re Interactive!
- They can have functionalities and details implemented
- They can be used just vor validation purposes so… they’re an expensive tool!

What about tools?



Tools

- Actually there's not an standard for that, you can choose from paid Premium tools to free and opensource others.

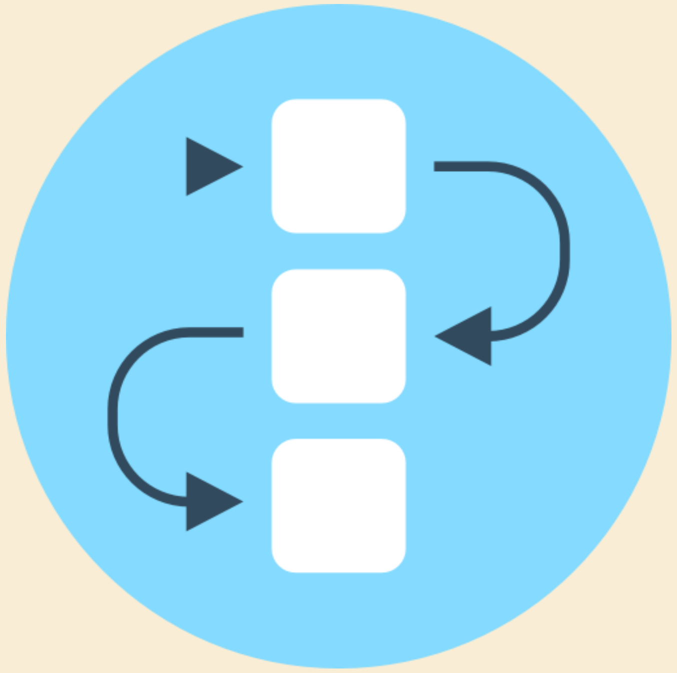
Like this: [Open Source]

https://speckyboy.com/10-completely-free-wireframing-and-mockup-tools/

Tools

I can list some here:

- Cacoo [Wireframes, UML]
- Lucidchart [Wireframes, UML]
- StarUML
- ArgoUML
- Visual Paradigm [$$]
- Sparx [$$]
- GenYModel[$$]



Let’s put all this chapter into practice!

Class excercise!

Let’s start again with projects, use everything don’t hesiatate!

References

- [SOMMERVILLE] Ian Sommervile. *Software Engineering 9th Edition*
- [SCHMIDT] Richard Schmidt. *Software Development Architecture-Driven Software Development*
- [WIEGERS] Karl Wiegers. Software Requirements 10 traps to avoid
- [CS2Ah] CS2 Software Engineering Lecture 2. Software Requirements
- [GeaGea et al] Software Requirements Specification, Amazing Lunch Indicator

This chapter has died!, review it!