LECTURE 3 – MODELS AND SPECIFICATIONS

Master of Applied Computing

COMP-8117 : Advanced Software Engineering Topics

Dr. Aznam Yacoub – aznam.yacoub@uwindsor.ca
School of Computer Science

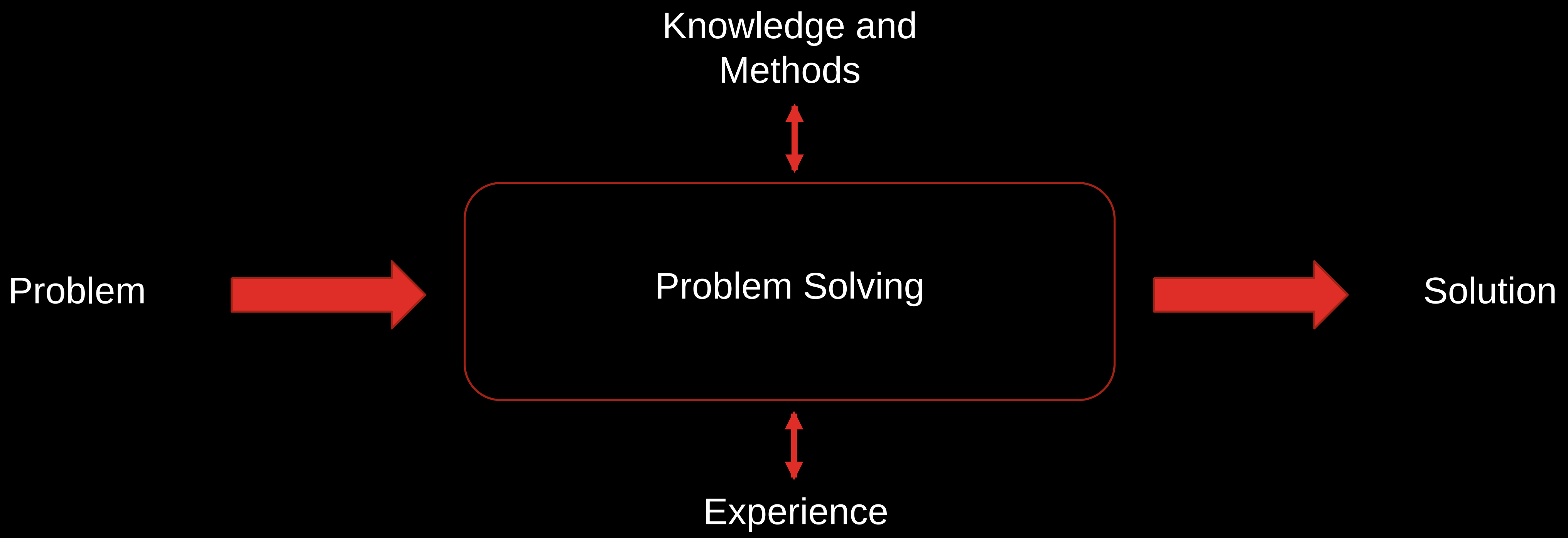
SCHEDULE

- Introduction
- Models and Formalisms
- From gathering requirements to specifications
- Conclusion

INTRODUCTION

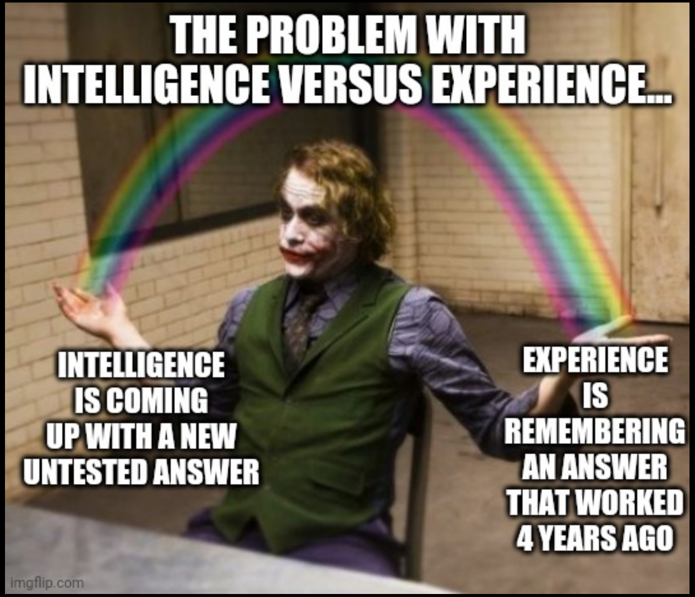
- To build a software, two questions to be answered:
  - What is the problem => Requirements / Specifications
  - How solve the problem => Specifications / Design

INTRODUCTION



INTRODUCTION

- Problem solving = Decision-Problem Process
- Problem solving = Scientific Approach
- Involves 2 elements
  - Intelligence
  - Experience


THE PROBLEM WITH
INTELLIGENCE VERSUS EXPERIENCE...
INTELLIGENCE IS COMING UP WITH A NEW UNTESTED ANSWER
EXPERIENCE IS REMEMBERING AN ANSWER THAT WORKED 4 YEARS AGO
imgflip.com

INTRODUCTION

- Steps for problem solving
  - Formalize the problem => Elicitation
  - Analyze the problem => Elicitation
  - Look for existing solutions
  - Evaluate the solutions
  - Describe the most suitable solution

INTRODUCTION

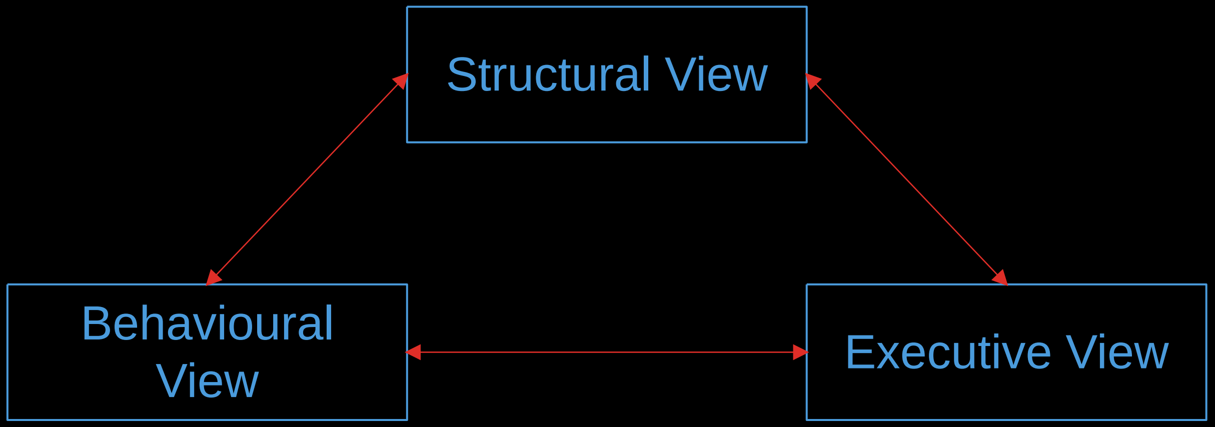
- These suitable solutions are **Models**.
- Rules leading to these models are **Methods**.
- The set of all the sequences of rules allowing to solve a problem is a **Methodology.**

INTRODUCTION

- By definition, possible to have a model of everything, including a model of methods.
- Keep in mind the goals
  - Understand what you do
  - Explain what you do
  - Manage what you do

SYSTEM 3-VIEW MODEL

- Especially, Models help you to gather and specify systems & software. Be careful : models are a description of a partial system.



SYSTEM 3-VIEW MODEL

- Structural view (static) : Describes the components (libraries, modules, piece of software or hardware…) and relationships between these components. From a logical point of view = relationships between conceptual objects.

- Behavioural view (dynamic) : Describes how the components evolve in time.

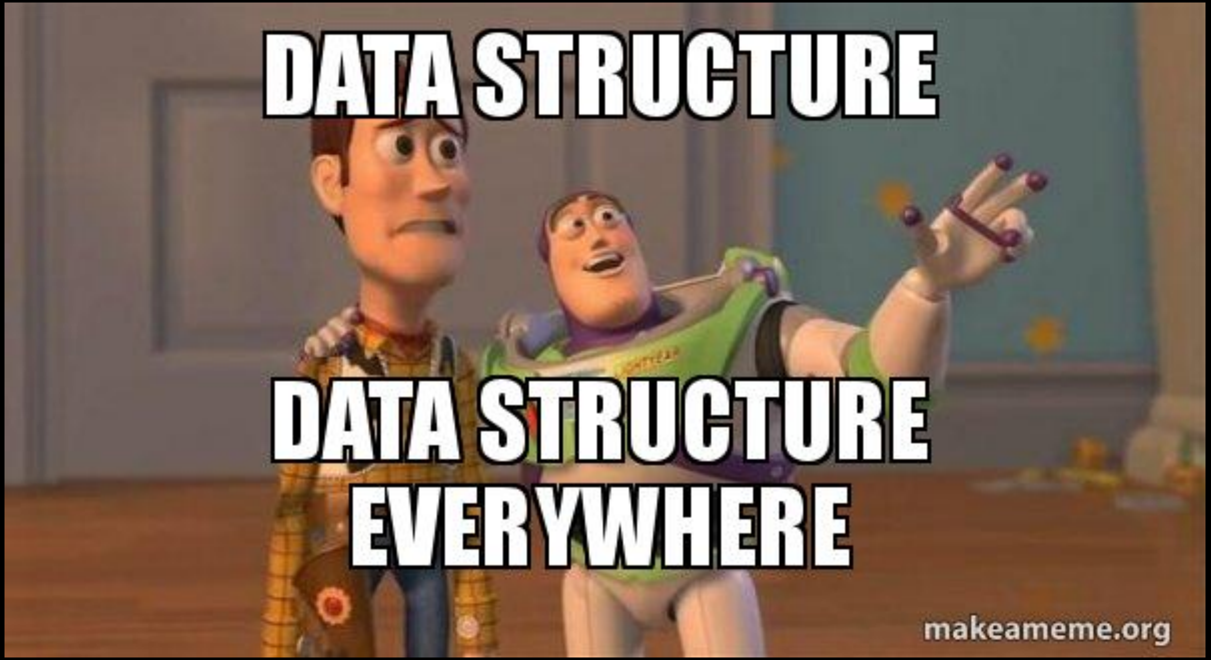
- Executive : Describes the physical distribution of the components (how they are deployed).

SYSTEM 3-VIEW MODEL

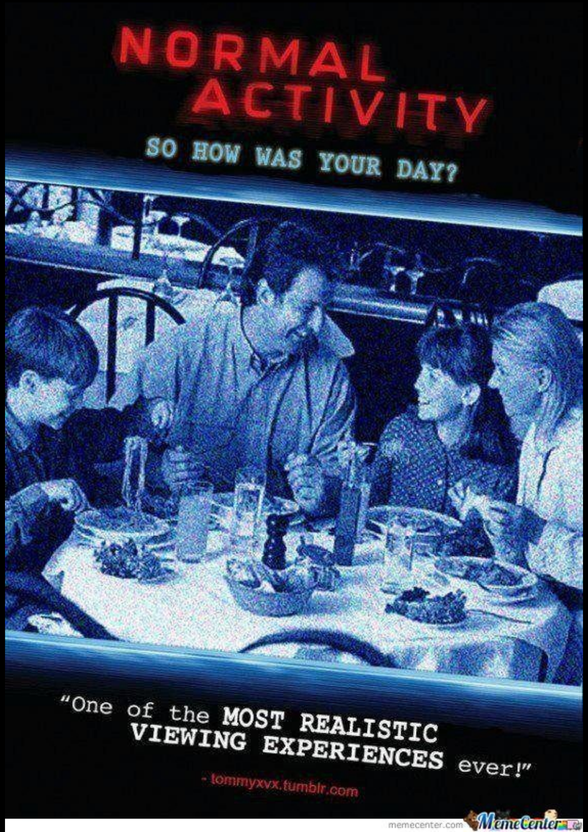
- All the existing models for specifications and design try to define these 3 views.

- 3 spaces when you model a software:
  - Data space = define structure of the data (from a information system perspective)
  - Activity space = define structure of the functions (from a computing system perspective)
  - State space = define bheaviour of the system (from a controlling system perspective)

DATA… IS WHAT YOU SEE



ACTIVITY… IS WHAT YOU DO


NORMAL ACTIVITY
SO HOW WAS YOUR DAY?
"One of the MOST REALISTIC VIEWING EXPERIENCES ever!"
- tommyxvx.tumblr.com
memecenter.com MemeCenter

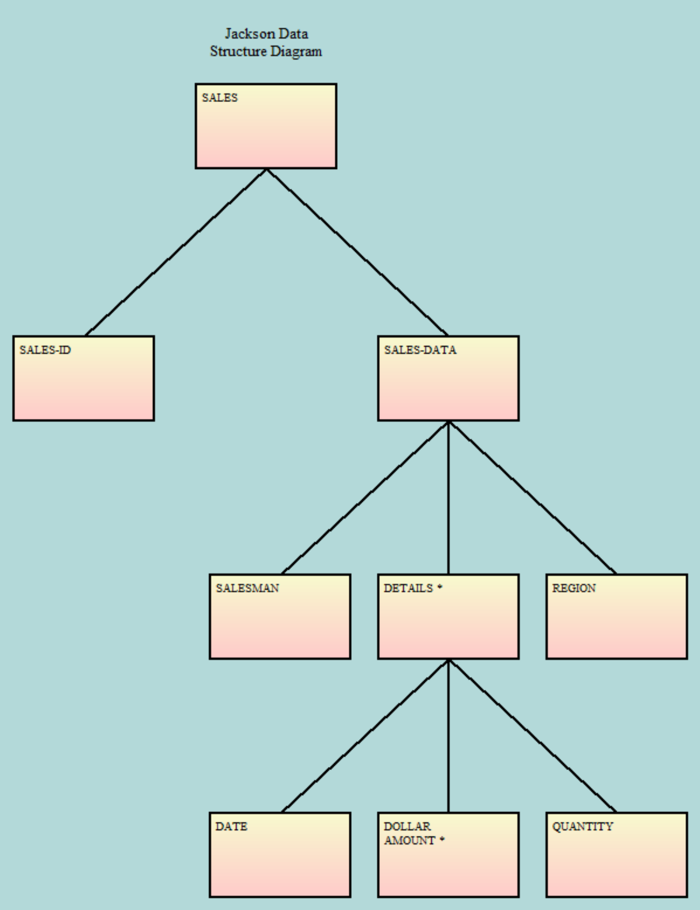
OK NOW… STATE IS HOW YOU FEEL

THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.

DATA SPECIFICATIONS

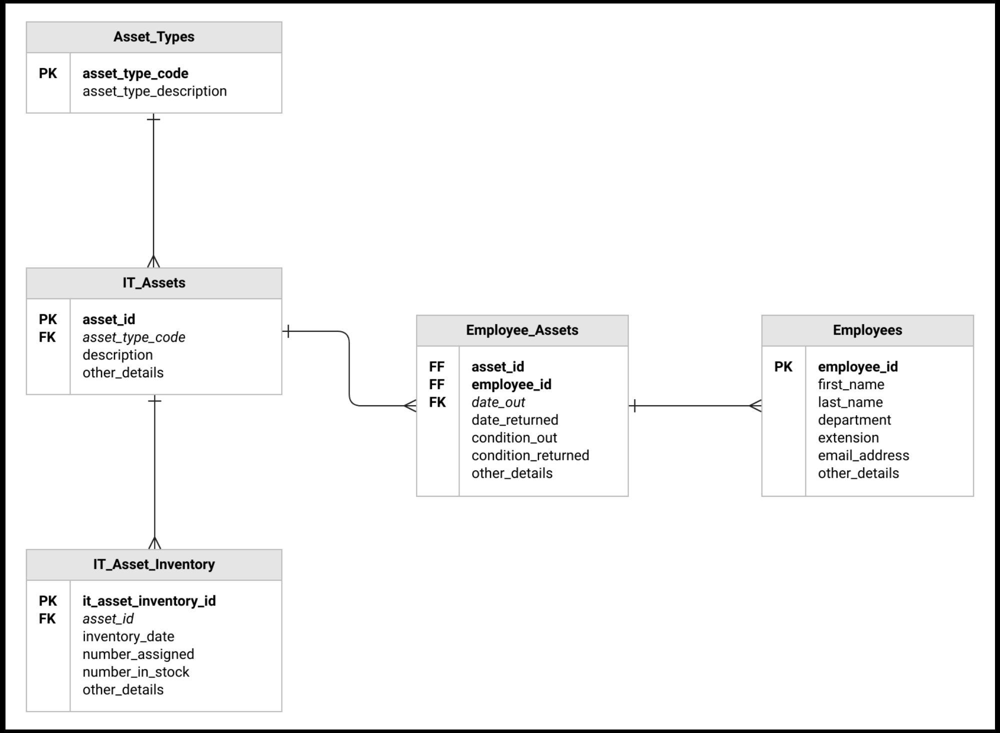
- There is a lot of representation (formalism) to represent data structure.

- Example : Jackson Data Diagram



DATA SPECIFICATIONS

- There is a lot of representation (formalism) to represent data structure.
- Example : Entity Relationship Diagram



Asset_Types

PK	**asset_type_code** asset_type_description

IT_Assets

PK FK	**asset_id** *asset_type_code* description other_details

Employee_Assets

FF FF FK	**asset_id** **employee_id** *date_out* date_returned condition_out condition_returned other_details

Employees

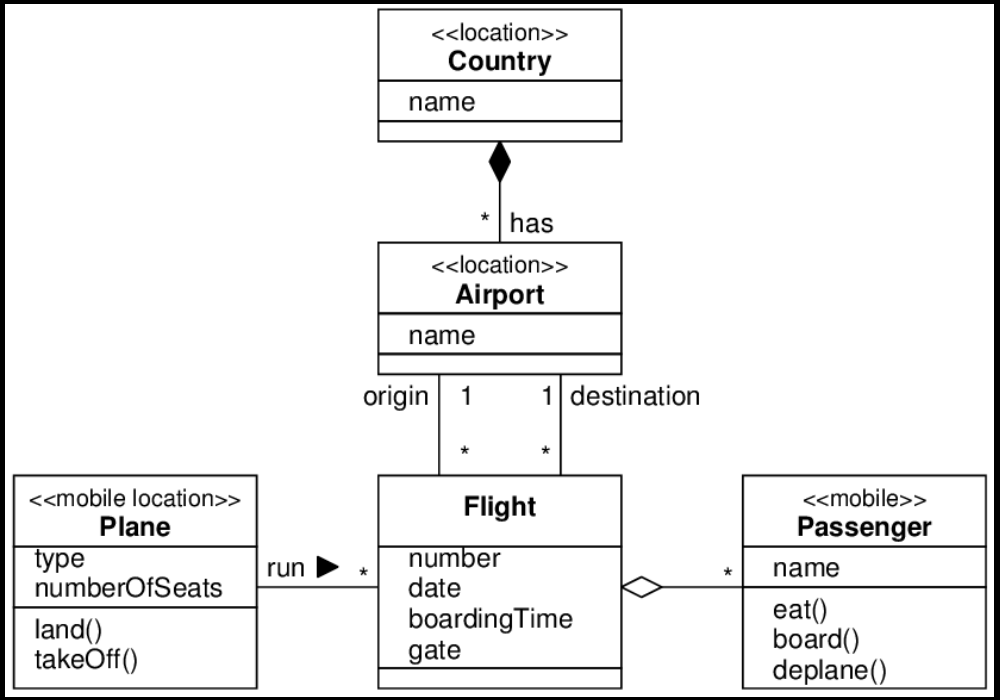
PK	**employee_id** first_name last_name department extension email_address other_details

IT_Asset_Inventory

PK FK	**it_asset_inventory_id** *asset_id* inventory_date number_assigned number_in_stock other_details

DATA SPECIFICATIONS

- There is a lot of representation (formalism) to represent data structure.
- Example : Class Diagram



FUNCTIONS SPECIFICATIONS

- There is a lot of representation (formalism) to represent activities/functions.
- Example : SADT, Dataflow, UML Activities, Flowchart, etc.

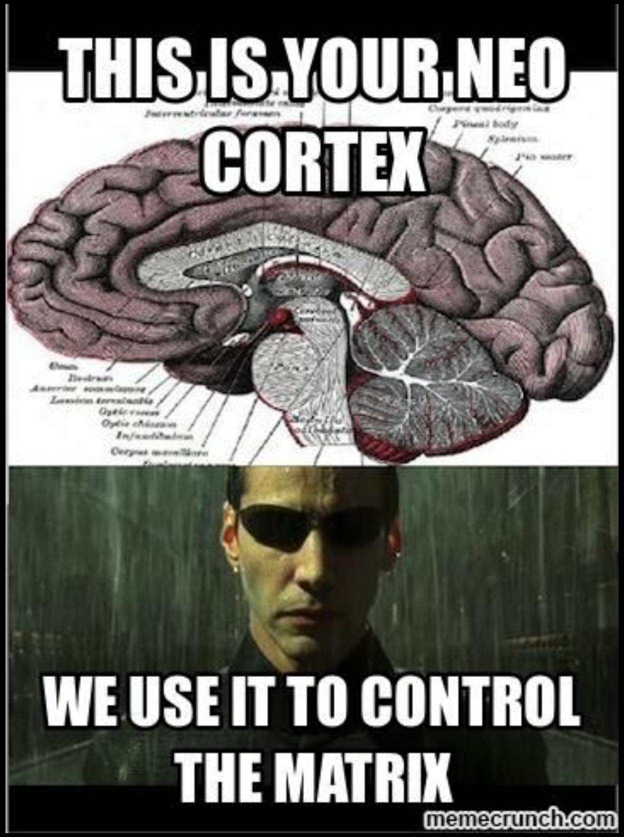
BEHAVIOURAL SPECIFICATIONS

- There is a lot of representation (formalism) to represent behaviours (states).
- Example : Mathematical equations, formal models, algorithms, pseudo-code, state machine, statechart, petri nets…

WHAT YOU HAVE TO UNDERSTAND

- When you do specifications, you model your software from several point of view => You’ll get several models
- Requirements models (specifications) => represents something concrete / tangible
- Architecture models (design) => represents something abstract / not tangible

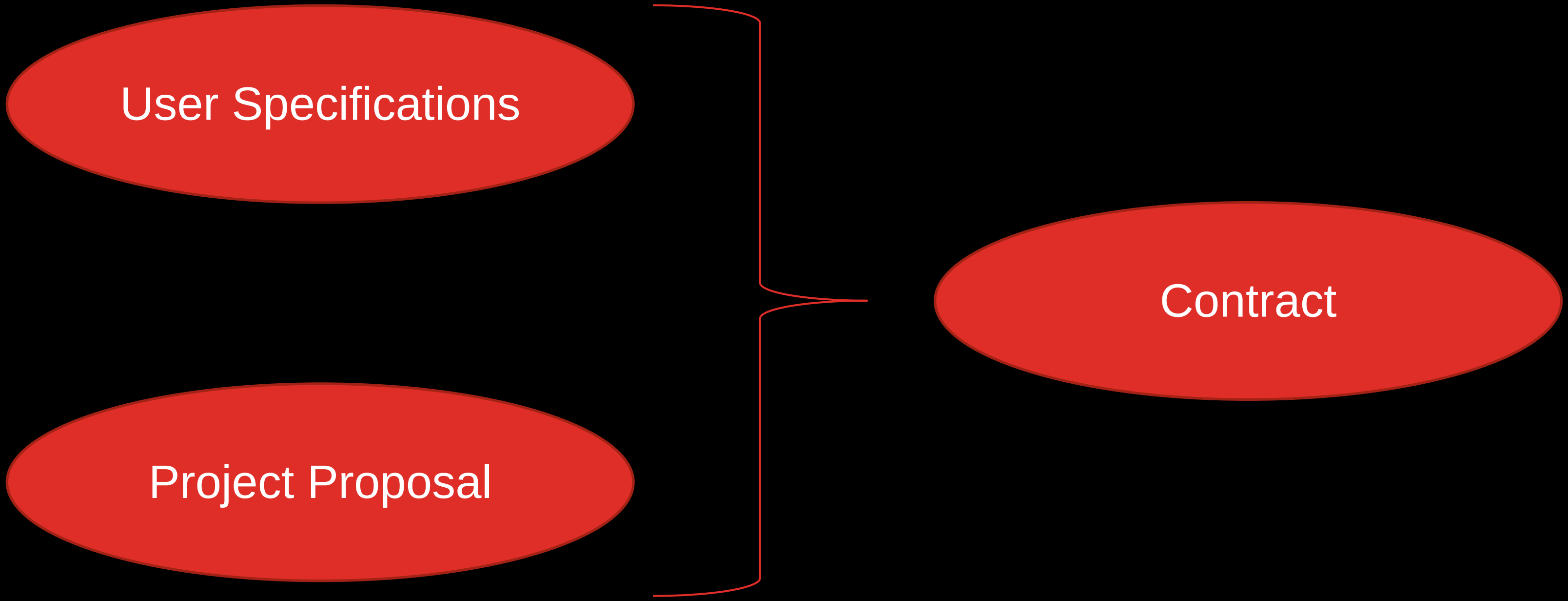
WHAT YOU HAVE TO UNDERSTAND



PROJECT PROPOSAL

- It depends on the project, but generally:
  - The SRS (In iterative methods, it’s an overview, not detailed)
  - The QAP (Quality Assurance Plan)
    - Description of the lifecycle
    - Description of the certification plan
    - Description of the proposal
    - Risk analysis
    - Description of the development team and roles
    - Communication processes
    - Review procedures
  - Cost and Market analysis

WHAT A SPECIFICATION ?

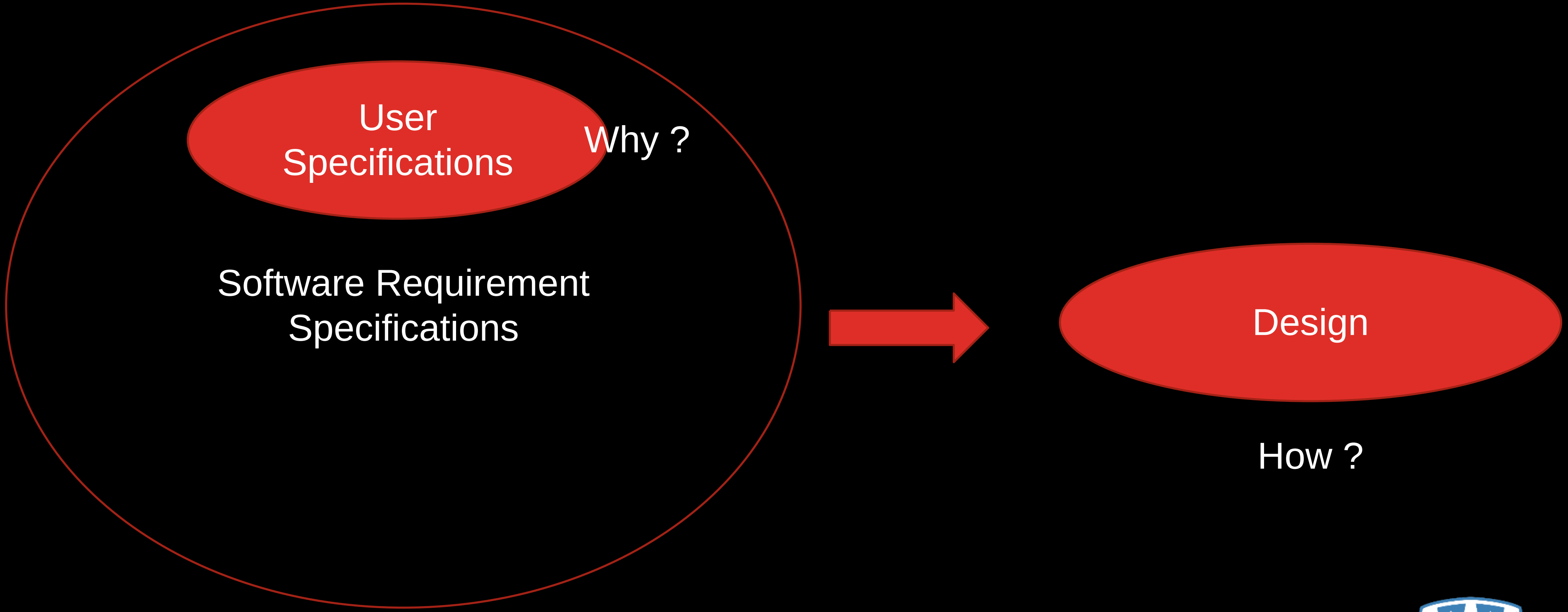


User Specifications

Project Proposal

Contract

WHAT A SPECIFICATION ?



User Specifications

Why ?

Software Requirement Specifications

Design

How ?

What ?

WHAT A SPECIFICATION ?

- The three documents describe the project, but not from the same point of view and not with the same level of details.
- User Specifications (User Stories) <= Requirement gathering
- Software Specification Requirements <= Requirement analysis

A project proposal answers the user specifications. SRS contains US.

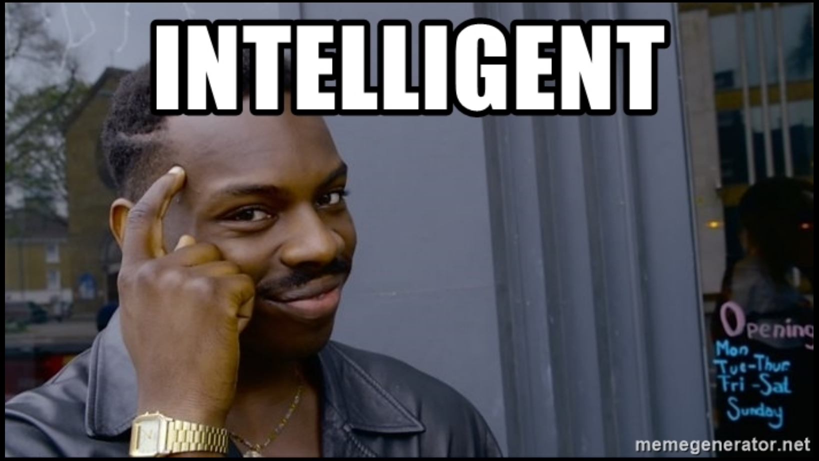
THREE PERSPECTIVES

- Complete specifications describes external charateristics according to three view. What are these views ?

THREE PERSPECTIVES

- Complete specifications describes external charateristics according to three view. What are these views ?

- Ok, it was just to see if you were still alive.



THREE PERSPECTIVES

- Structural:
  - Data (What) => Describe the problem, flow of informations, structure of the information
  - Functions (How) => Describe the function required to solve the problem
- Behavioural:
  - Behaviours / State (When) => Describe the events and reactions to these events

4 PHASES OF SPECIFICATION

- Model the environment
  - Use schemes
  - Describe the objects which the software will interact with (An object can be from the real world or from the application world)

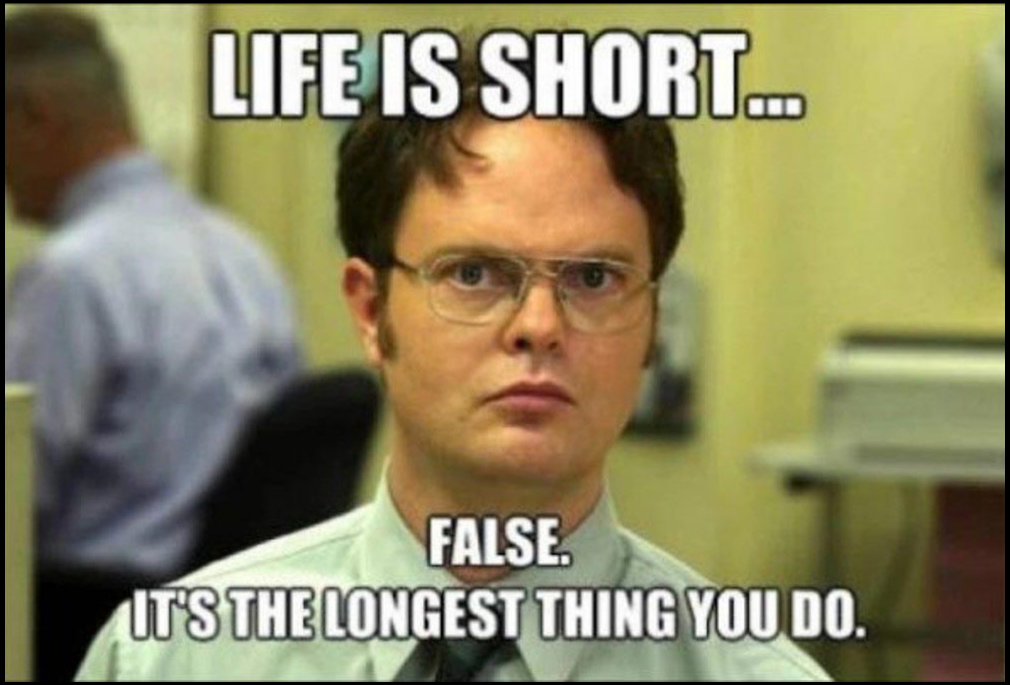
  - If you do an elevator controller, a possible object is an elevator
  - If you do a chat messenger, a possible object is a window

4 PHASES OF SPECIFICATION

- Model the environment
    - For each object, decompose it in subobjects and group them by categories (or types)
    - Example:

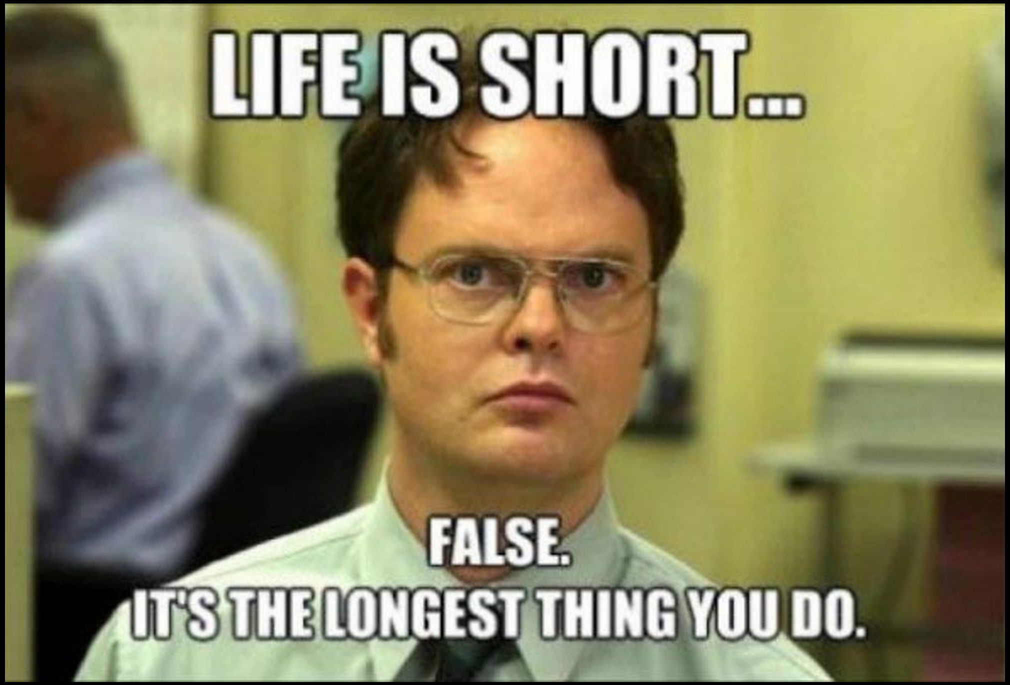
4 PHASES OF SPECIFICATION

This is a meme. →



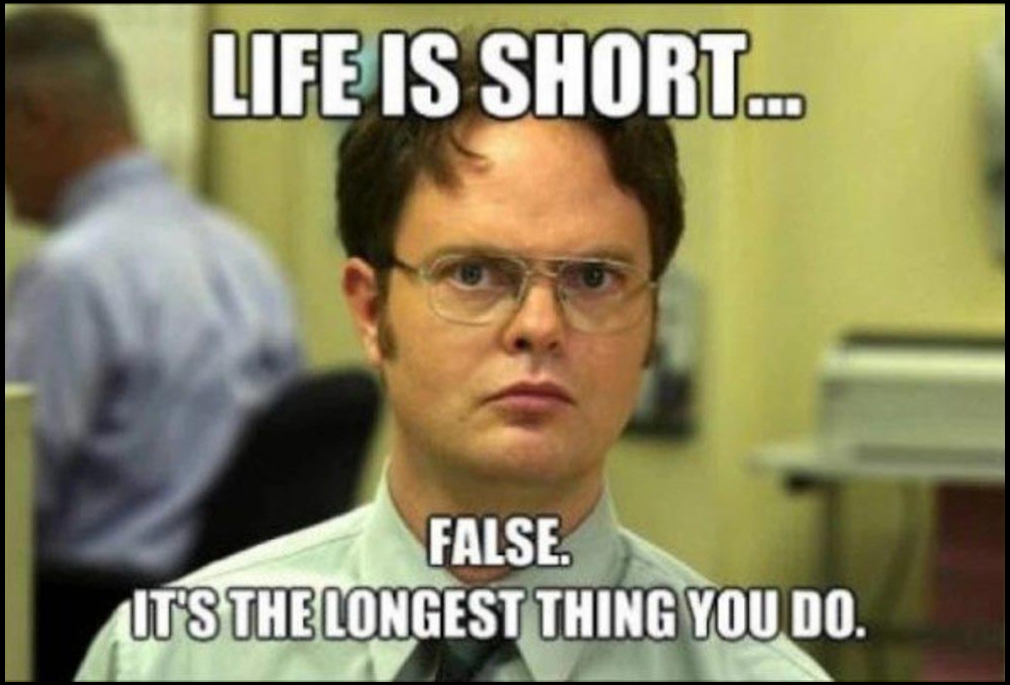
4 PHASES OF SPECIFICATION

This is also a picture.



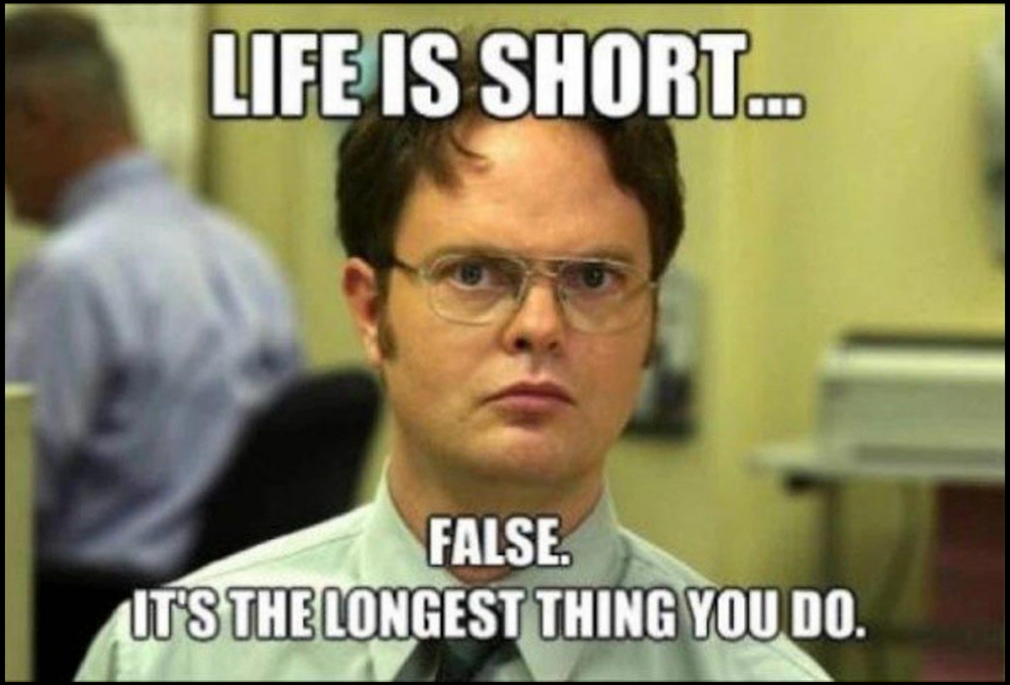
4 PHASES OF SPECIFICATION

This is also a set of pixels.



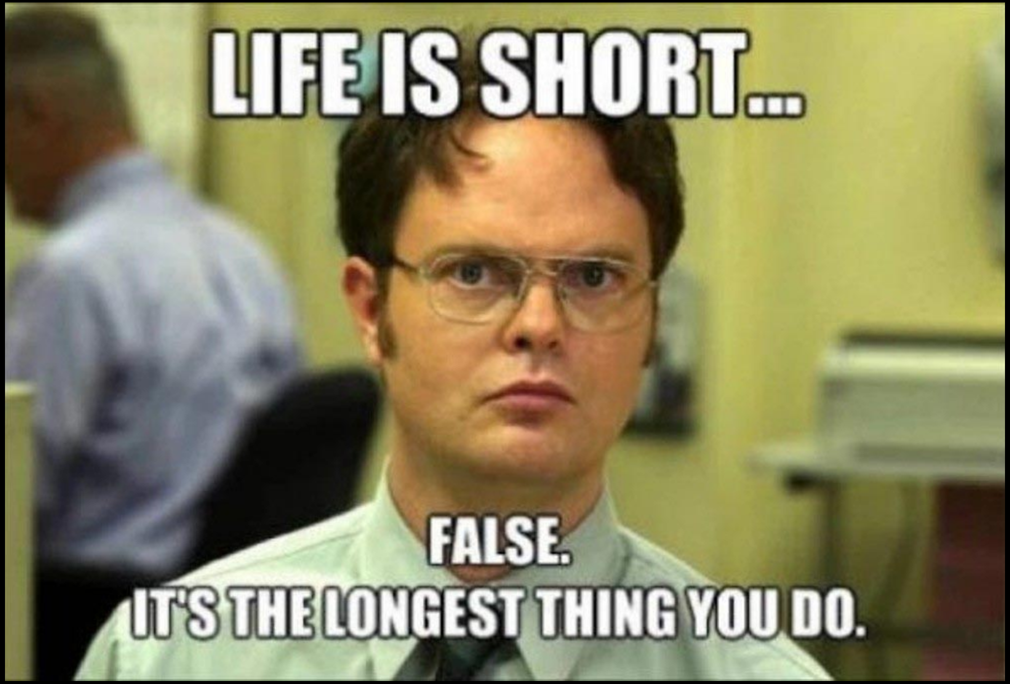
4 PHASES OF SPECIFICATION

Each pixel is a tuple of colors.



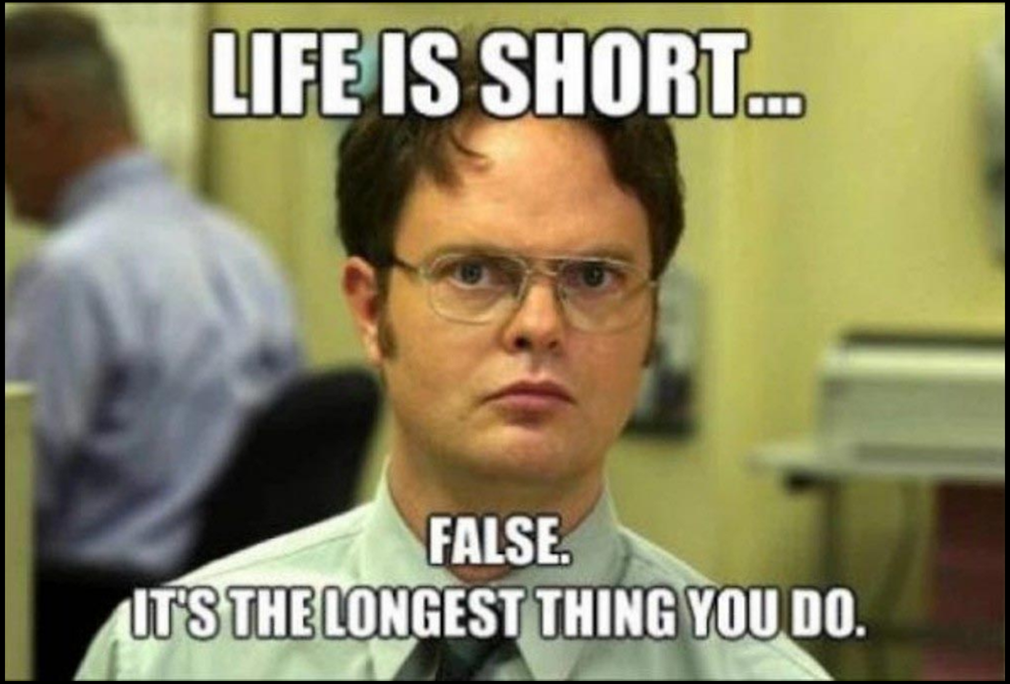
4 PHASES OF SPECIFICATION

Components of color are Red, Blue, Green



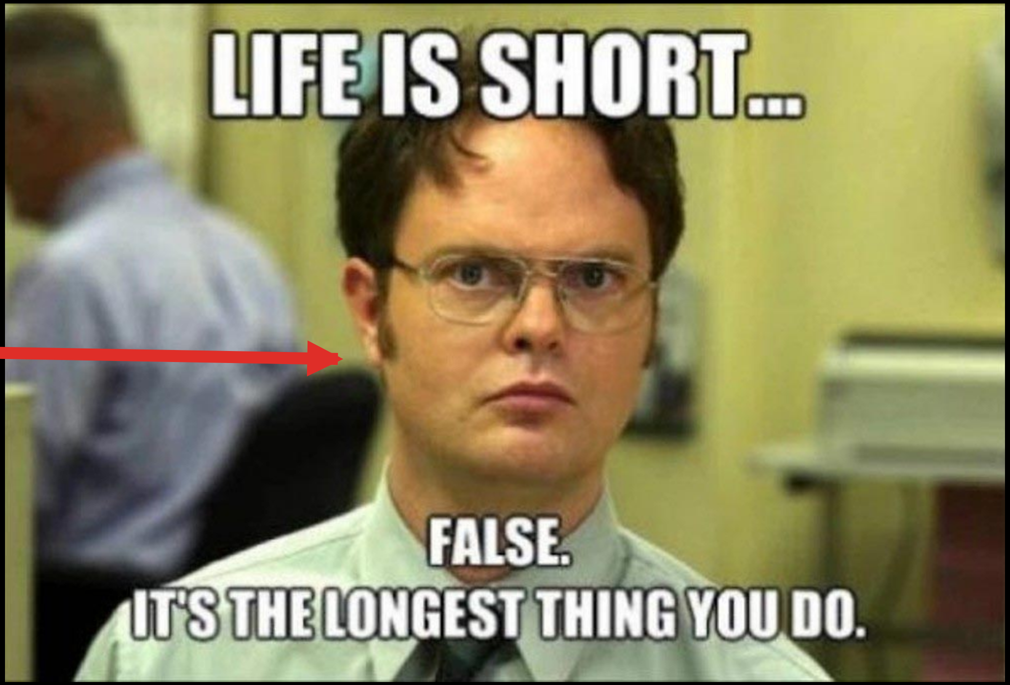
4 PHASES OF SPECIFICATION

This picture contains a Human.



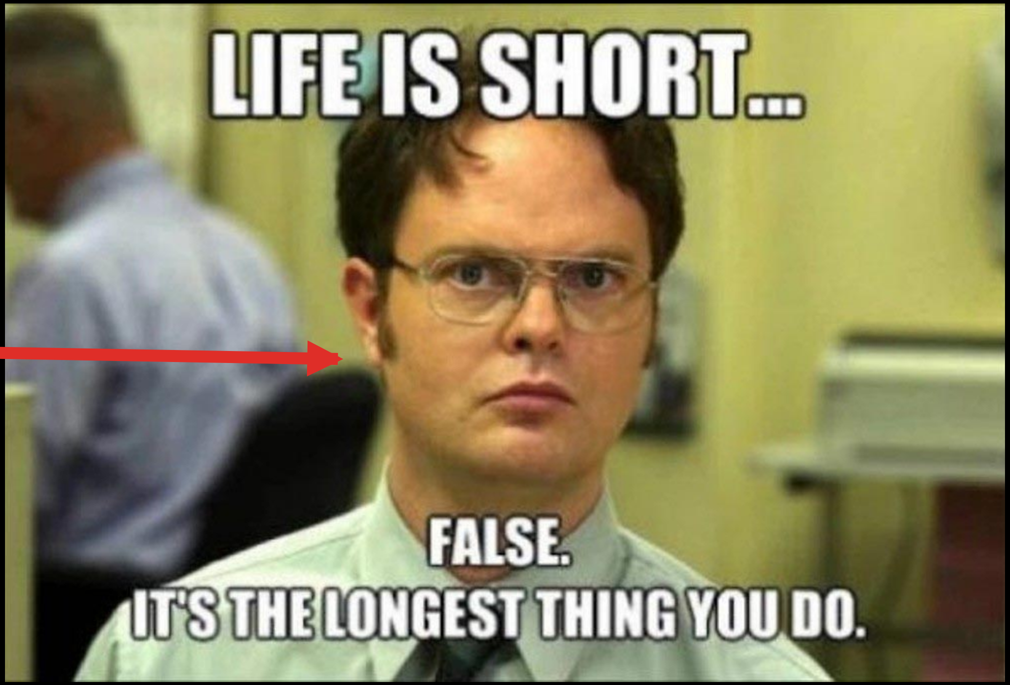
4 PHASES OF SPECIFICATION

This is a Human



4 PHASES OF SPECIFICATION

It’s also a character



4 PHASES OF SPECIFICATION

It’s also a character

4 PHASES OF SPECIFICATION

A character doesn't exist in reality

ME
SENDING MEMES
IS THIS A MEME?
made with AI on imgflip.com

4 PHASES OF SPECIFICATION

But it’s still an abstract object

4 PHASES OF SPECIFICATION

- Model the environment
  - 3 categories of characteristics (called also attributes)
  - Data (everything is a data)
  - Event (data which triggers the system and implies a change of state)
  - Action (operations on the system or by the system)

4 PHASES OF SPECIFICATION

- Define the input/output functions
  - For each data your software will manipulate through a function, describe the output.
  - You have external functions (features)
  - You have internal functions (functionnalities performed by the system but hidden for the users)

  - In this step, you describe only what you can observe from an external point of view. You don’t describe the internal steps !

4 PHASES OF SPECIFICATION

- Define the input/output functions

This is a input/output functions.

4 PHASES OF SPECIFICATION

- Define the input/output functions

You don’t see it ?

4 PHASES OF SPECIFICATION

- Define the input/output functions

Normal, it’s a black box.

4 PHASES OF SPECIFICATION

- Define the input/output functions

Now you see it.

Strawberry → Banana

4 PHASES OF SPECIFICATION

- Define the sequence of events
    - Now, you describe the different steps of the internal operations of the external functions.
    - At this step, your job is to describe generic algorithm / subfunctions (but still from an external point of view).
    - You don’t describe the internal computational operations.

4 PHASES OF SPECIFICATION

- Define the sequence of events
  - Example :

This a Jedi. A Jedi has a lightsaber. The lightsaber can be on/off position. When activated, a plasma is heated. This plasma goes out and stabilized, emitting a light thanks to a cristal. This cristal deviates the light and gives the color of the saber.

I'm still describing what I observe, not how I implement it.

4 PHASES OF SPECIFICATION

- Define the user guide.
  - Write a kind of procedures of uses and installation as if it was the final user guide.
  - This user guide should confirms you that the user will be able to use the software as intended.

SPECIFICATION IN AGILE METHODS

- A good approach of specification in agile methods:
  - Don’t specify everything at the beginning
  - Specify when you need and get a feedback

- Use the FURPS+ checklist
  - Functional : features, capabilities, security.
  - Usability : human factors, help, documentation.
  - Reliability : frequency of failure, recoverability, predictability
  - Performance : response times, throughput, accuracy, availability, resource usage
  - Supportability : adaptability, maintainability, internationalization, configurability

SPECIFICATION IN AGILE METHODS

Use the FURPS+ checklist

- Functional : what do I want ?
- Usability : who will use ?
- Reliability : what is acceptable failure ?
- Performance : what level of performance ?
- Supportability : what about maintenance ? How is it easy to test ?

- + = Design constraints (dev team knows about how you would like the system to be designed ?), Implementation (which standards ?), Interface, Physical, Operations, Packaging, Legal

SPECIFICATION IN AGILE METHODS

- We call these requirements:
    - Quality attributes
    - Quality requirements
- Two categories of requirements:
    - Functional (Behavioural)
    - Non-functional
        - Architectural (= Technical)
        - Business
        - User
        - Quality of service
        - Implementation (= Transition/Deployment)
        - Regulatory

SPECIFICATION IN AGILE METHODS

- In UP, requirements gathering is done during Inception. Analysis (Specification) is done during Elaboration.
- ALL the specifications is NOT done during Inception. Especially, you have to answers:
  - Vision and business case : Describe high level goals
  - Use cases : Functional requirements (less than 10% detailed)
  - Non-functional requirements
  - Glossary : Data and terminology
  - Risks and Management Plan
  - Prototypes
  - Iteration plan (because Inception is iterative)
  - Phase plan and Development case : describe the lifecycle

SPECIFICATION IN AGILE METHODS

- Inception may comes during project proposal. In Inception, you have still time to kill the project.
- Questions:
  - What is the vision ?
  - Is it feasible ?
  - Should we buy or build ?
  - Exact Cost ?

SPECIFICATION IN AGILE METHODS

- Goal of inception is to have a better idea.
- It should be brief.
- Question: Do you think you'll need an inception phase i your project ?
- https://www.youtube.com/watch?v=B0CY5bKgoeY

SPECIFICATION IN AGILE METHODS

- Elaboration in UP – Specify and develop the core architecture.
- We define at these steps most of the requirements and specifications.
  - Requirement gathering is more seriously performed than in Inception.
  - Requirement analysis & specification is performed for the core architecture.

- Difference between Inception and Elaboration
  - Inception gives you an idea of the project
  - Elaboration gives you an idea of the architecture

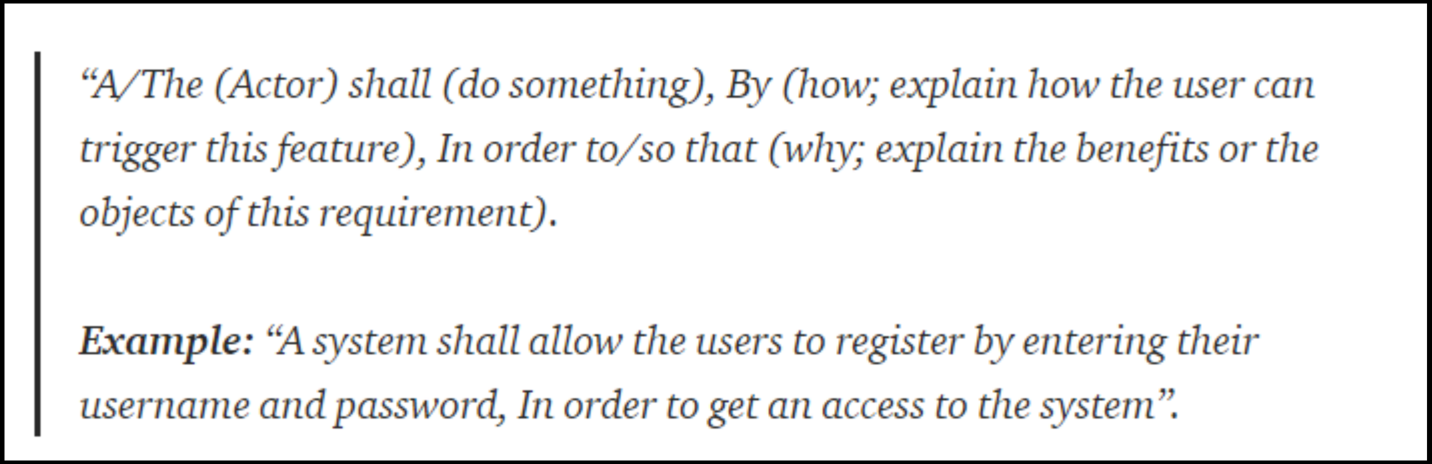
SPECIFICATION IN AGILE METHODS

- Construction in UP – Specify and develop the secondary features.
- We refine the specifications for the remaining not implemented features.

- Whatever the step, requirements & specifications follow a systematic, manageable, verifiable walktrough.
  - Top-Down : Decompose all the elements structurally
  - Bottom-up : Start from small components and describe their structural composition
  - Relationship : Describe the interactions between components
  - Functional : List all features, and for each features describe the components involved. Recursively do the same for all components.
  - Etc.

NATURAL LANGUAGE SPECIFICATION

- No predefined format => create your own format when you define procedures.
- Just use plain text.
- Example (From Omar Elgabry’s blog) :



> *“A/The (Actor) shall (do something), By (how; explain how the user can trigger this feature), In order to/so that (why; explain the benefits or the objects of this requirement).*
>
> ***Example:*** *“A system shall allow the users to register by entering their username and password, In order to get an access to the system”.*

MATHEMATICAL SPECIFICATION

- Use mathematical objects and domains to describe your object.
- Example:
  - My system is composed by a set of users U = { Boy, Girl }
  - My feature « connect » defined as followed:

    Connect(u) = u’ where (u,u’) in U x U connects to friend

STRUCTURED SPECIFICATION

- Use structured form.

Number - Title	
Description	
Inputs	
Expected Output	
Source	
Possible errors	
Requires	
Pre-condition	
Post-condition	
Sequence of actions	

FORMAL SPECIFICATION

- Use formalisms or (semi-)formalized diagrams
- Example:
  - Algebric specification, Z specification, B specification
  - UML Diagrams* (Use Case, Activity, Sequence, Object, Class, Entity-Relationship, etc.)
  - SADT..

FORMAL SPECIFICATION

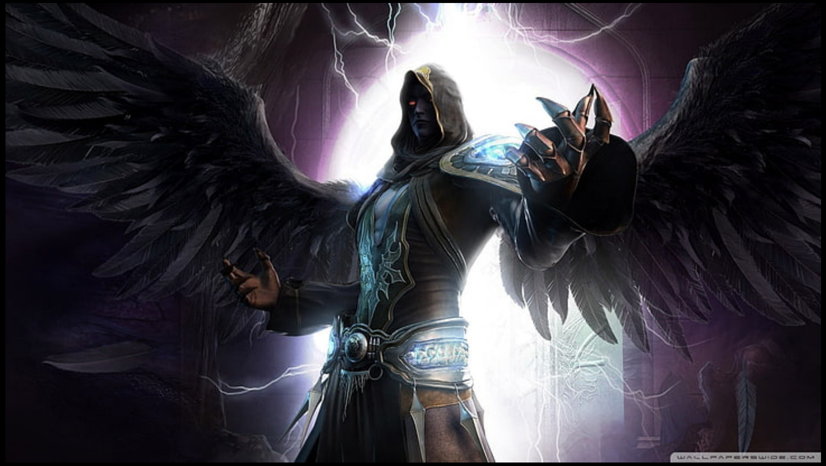
- Be careful with UML: in companies, people associate UML diagrams to design and often make confusion between high-level/simplified design and specifications (because both translates the user domain).

- High recommandation: keep UML diagrams for design, and use the other formalisms for specification (except if you cannot capture a relation because of the paradigm; example: you cannot use SADT if you’re in object-oriented paradigm).

SPECIFICATION – PRACTICAL CASE

- Specification of human:

  Describe what you see on the picture



SPECIFICATION – PRACTICAL CASE

- A human has legs, arms, head, body 🡐 subcomponents

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age 🡸 attributes

How to make the difference between components and attributes ?

Actually an attribute can become a component <= depends on the level of abstraction

SPECIFICATION – PRACTICAL CASE

- A human can move, sleep, eat, drink 🡐 operations / functionnalities

- What is a move ? He can walk/run/jump <= Refinement

- What exactly a walk ?

SPECIFICATION – PRACTICAL CASE

- A human can move, sleep, eat, drink 🡸 operations / functionnalities
- What is a move ? He can walk/run/jump <= Refinement
- What exactly a walk ? It's when the human goes from a ***position A*** to ***B***.

SPECIFICATION – PRACTICAL CASE

- A human can move, sleep, eat, drink 🡐 operations / functionnalities
- What is a move ? He can walk/run/jump <= Refinement
- What exactly a walk ? It's when the human goes from a ***position A*** to ***B***.

Discovery of two new attributes

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age, position ← attributes

- Do I need one or two position attributes to represent the move ? => Design

- *Optional specification / design constraint: position is couple (x,y) of real numbers*

SPECIFICATION – PRACTICAL CASE

- Walk is a function defined as:
  - If (x,y) is the position of my human, and v = (vx, vy) a vector which represents its speed, then

  walk(x,y) = (x+vx, y+vy)

  Discovery of a new attributes : speed

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age, position, speed 🡐 attributes

- Speed is a vector. What is a vector ?

- Three possibilities:
    - 1. A vector is defined by the domain, no need to specify it.

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age, position, speed 🡐 attributes
- Speed is a vector. What is a vector ?
- Three possibilities:
  - 2. A vector is not defined by the domain, but it's not my problem (because I don't think about design, but from the user perspective)
  - User here = final user or designer

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age, position, speed 🡐 attributes

- Speed is a vector. What is a vector ?

- Three possibilities:
  - 2. If it's not a problem related to my software, I won't go further in the decomposition.

  - For instance, if the vector has the common mathematical definition, I don't define it again.

SPECIFICATION – PRACTICAL CASE

- A human has a name, gender, age, position, speed 🡰 attributes

- Speed is a vector. What is a vector ?

- Three possibilities:
    - 3. It’s not defined by the domain, but the meaning of a vector cannot be captured or understood by the user => I define what is a vector.

SPECIFICATION – PRACTICAL CASE

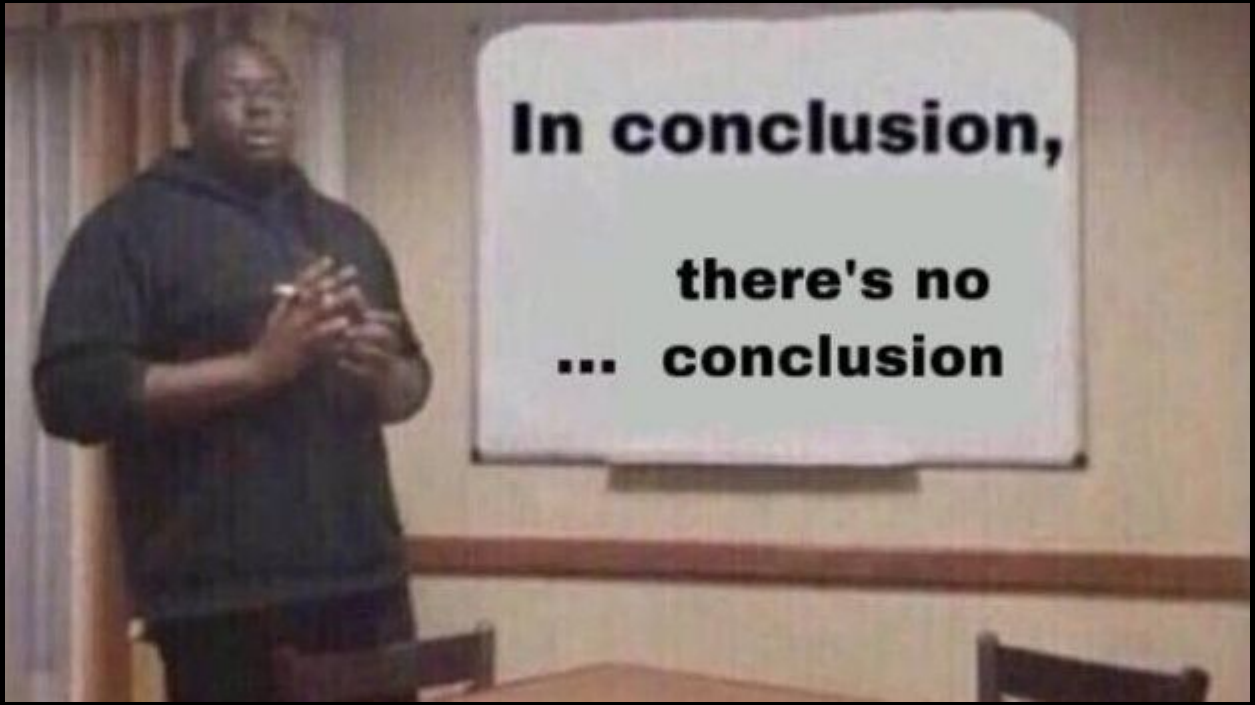
- I continue this specification process for all the functions, all the components, and all the attributes.

SPECIFICATION – PRACTICAL CASE

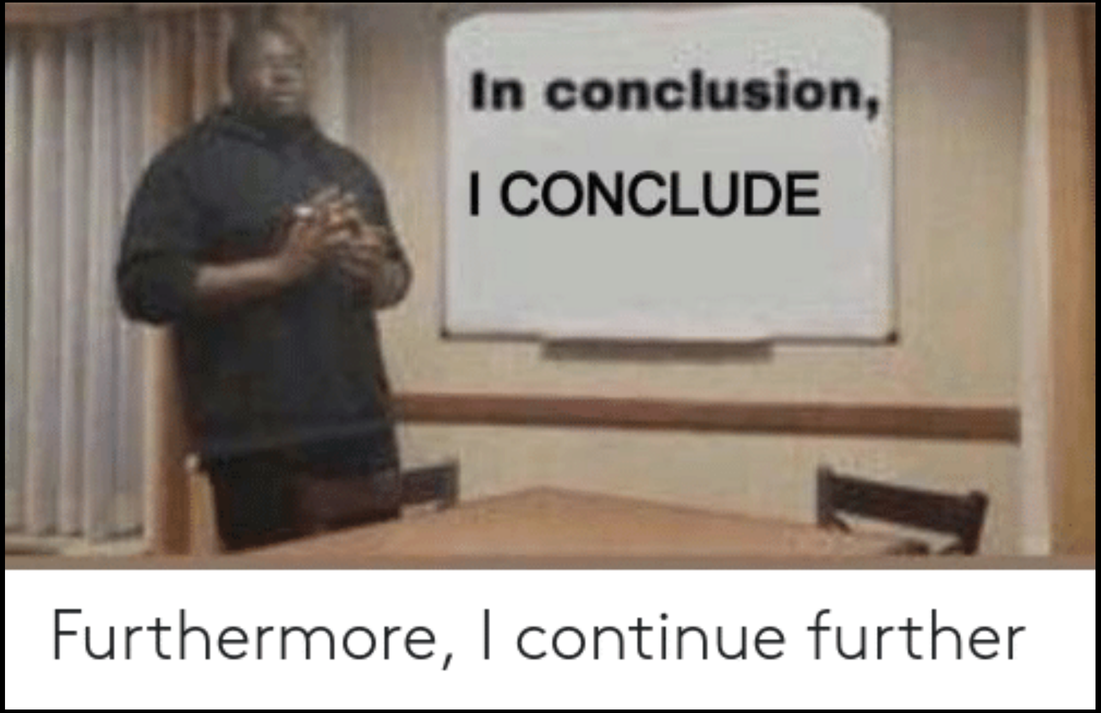
- I continue this specification process for all the functions, all the components, and all the attributes.

CONCLUSION

CONCLUSION



CONCLUSION



- How you do specification (complete or not) depends on your domain, and your methodology
- In waterfall : you specified everything at the beginning
- In iterative : you specify only what you need
- Specification should be understandable by your customers and your designers

CONCLUSION

- Specifications is a really important part of the work, including in agile.
- The difference is the quantity of specifications.
- Specifications help you to capture the user needs precisely, and the designer needs. It constraints also design. It helps you to define the problems (what and why), and make emerge a possible solution (but not the how).

NOW, IS YOUR VISION OF SE STARTING TO CHANGE ?

REFERENCES

This lecture is based on:

- COMP-8117 (Winter 2020) – Dr. Ziad Kobti
- Software Engineering (Fall 2020) – Dr. Amine Hamri, Dr. Aznam Yacoub
- Software Engineering – Ian Sommerville