PROGRAMMERING AV INBYGGDA SYSTEM

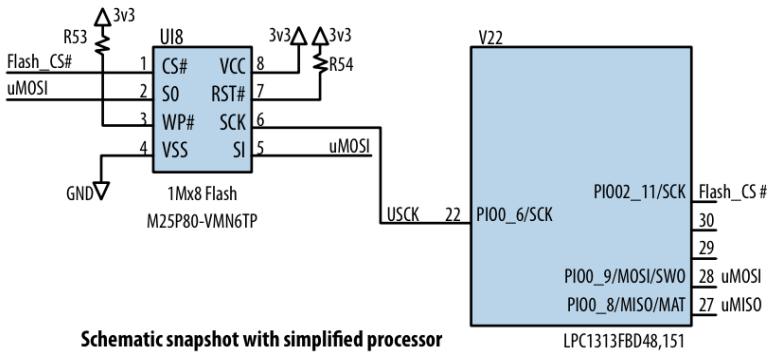
SW architectures

Dario Salvi, 2025

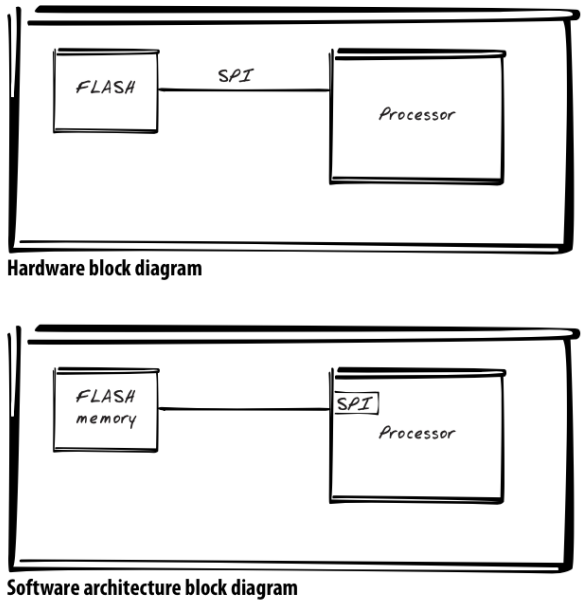
Materials

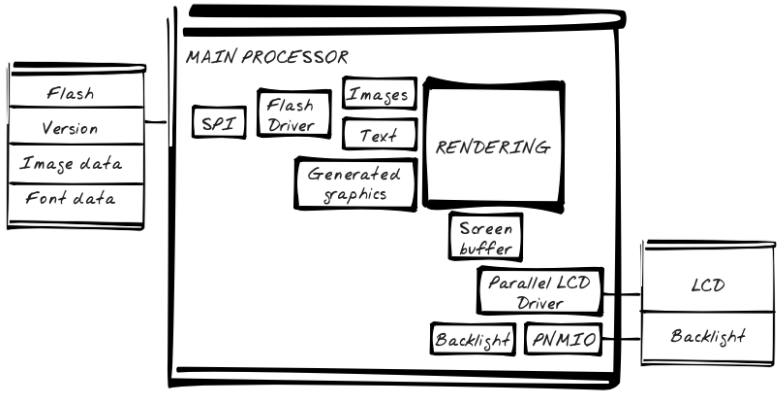
- Making embedded system: chapter 2, “Creating a System Architecture”
- The 4+1View Model of Architecture
- Applying 4+1 View Architecture with UML 2
- Online resources:
  - https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
  - https://martinfowler.com/bliki/PresentationDomainDataLayering.html
  - https://creately.com/blog/diagrams/uml-diagram-types-examples/

From hardware to software



Schematic snapshot with simplified processor





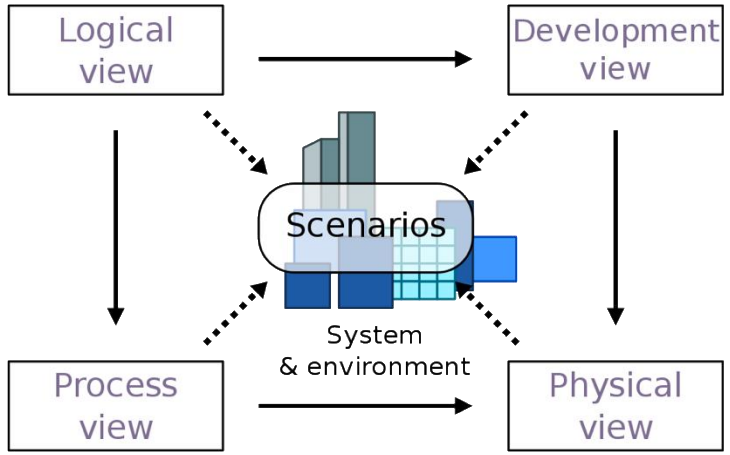
Hardware view

Basic software view

Complex software view

Software architecture

- *The architecture of a software system is a metaphor, analogous to the architecture of a building. It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.* (Wikipedia)

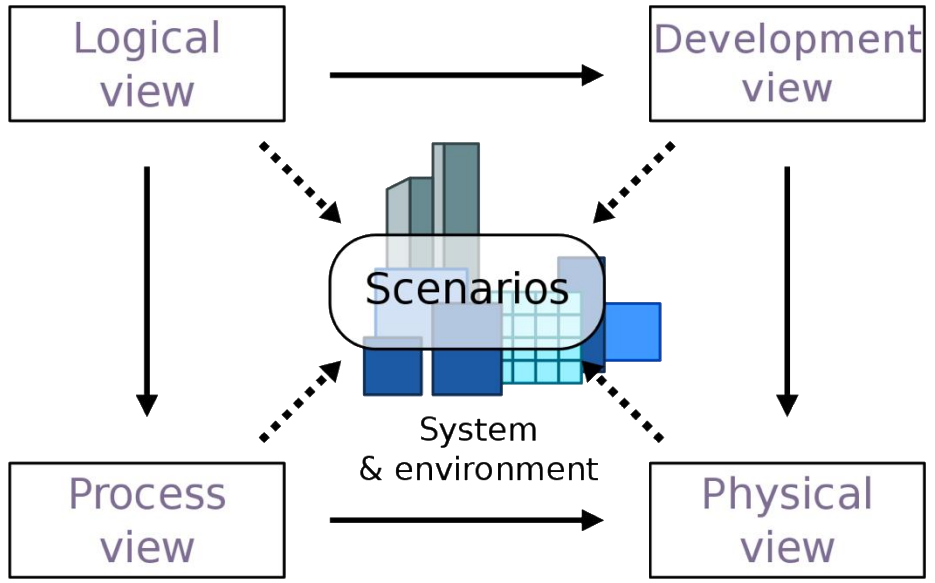


4+1 view model

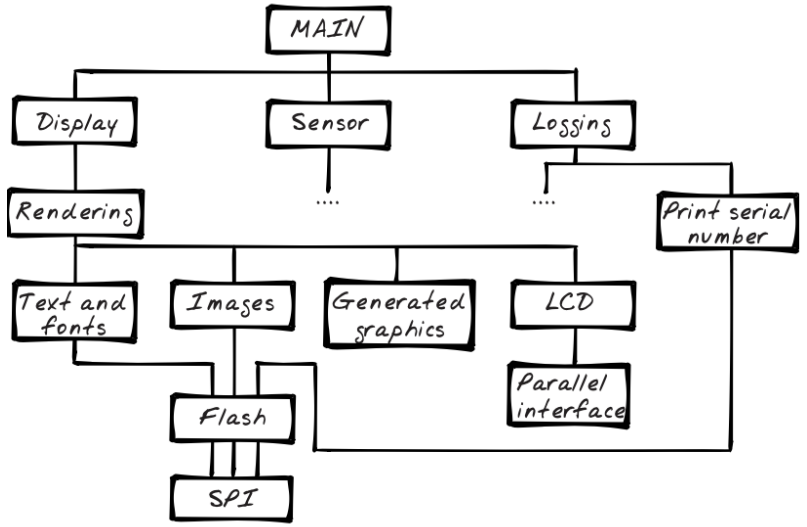
- It is composed of "**views**", each one may be match needs of different stakeholders

The 4 + 1 view model

- The **logical view** shows the key abstractions in the system as objects or object classes “code structure”.
- The **process view** shows how, at run-time, the system is composed of interacting processes “running code”
- The **implementation/development view** shows how the software is decomposed for development “who does what”
- The **physical/deployment view** shows the system hardware and how software processes are distributed across the processors in the system “things you can touch”
- All views are related using **use cases** or **scenarios**



Logical view



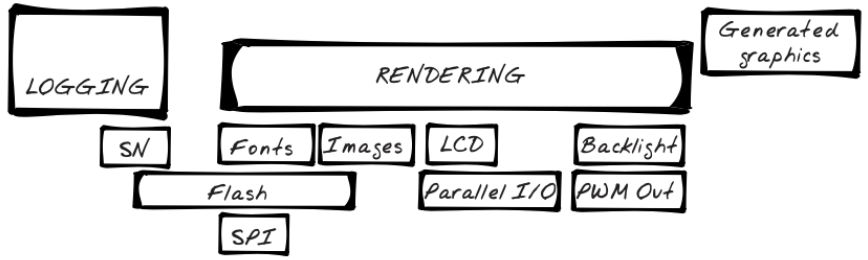
- The logical view is concerned with the **functionality** that the system provides to end-users.
- The logical view, shows the key abstractions in the system as **class diagrams** or **state diagrams**.
- Captures the structure of the system
  - Should exploit the principles of **abstraction** and **encapsulation** .
  - Also serves to identify common mechanisms and design elements across the various parts of the system.
- Object oriented programming helps here.
- Key view!
  - Experience says the design decisions done here will affect all other views.
  - Other views always relate to this view!

To think about

- Imagine you need to write the software that runs in a “smart fridge”.
- The fridge has a display that shows some information about temperature, humidity and how full (approximately) it is.
- It also connects to the Internet and sends/receive information through a website.

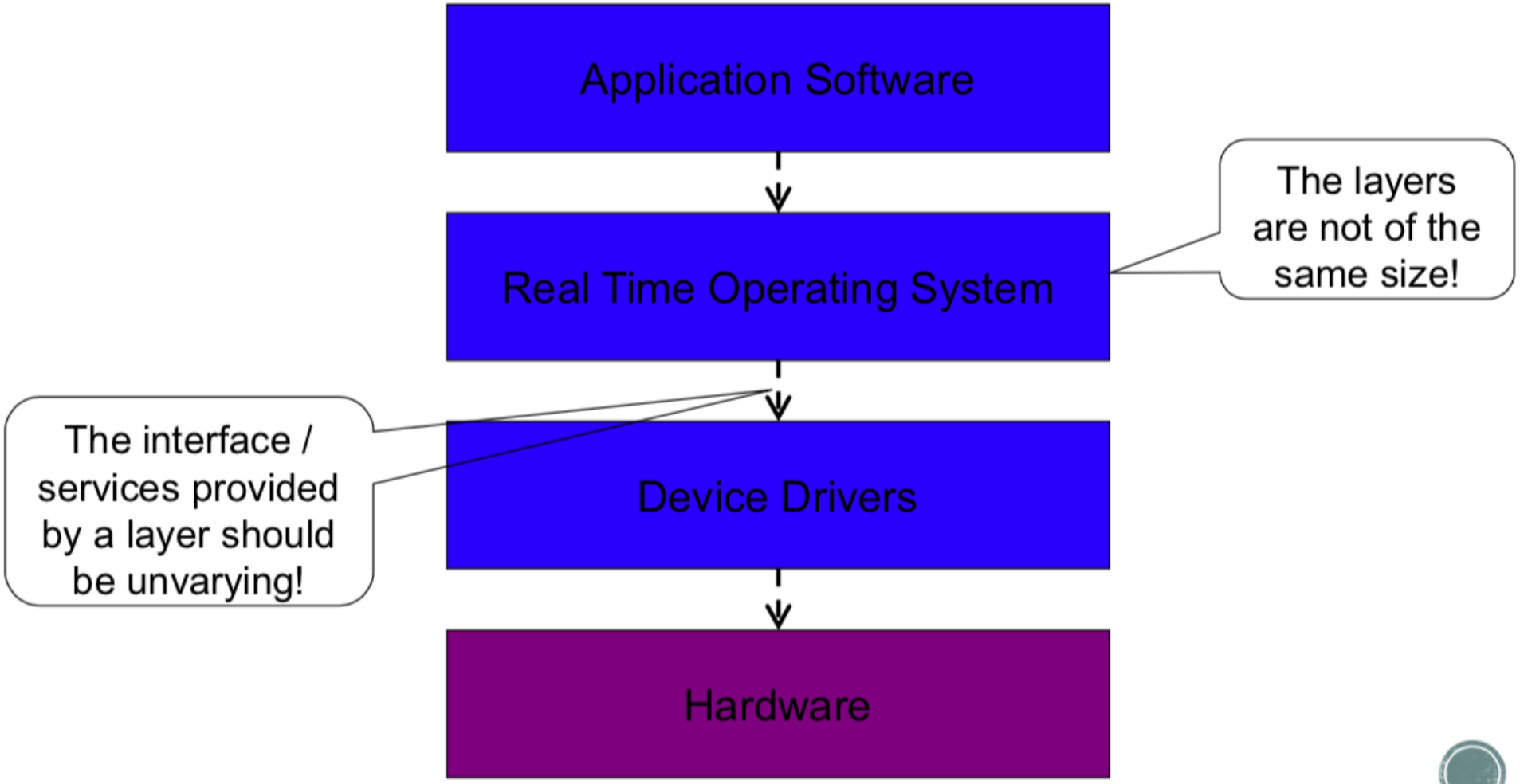
- What blocks (classes/modules) do you need?
  - Examples: sensors, user interface, parts of the UI (button, text, input), communication with server, lower protocols (HTTP, TCP, IP), configuration, RTOS?
- Draw them down on paper (or your laptop).

Layered logical view (n-tier)



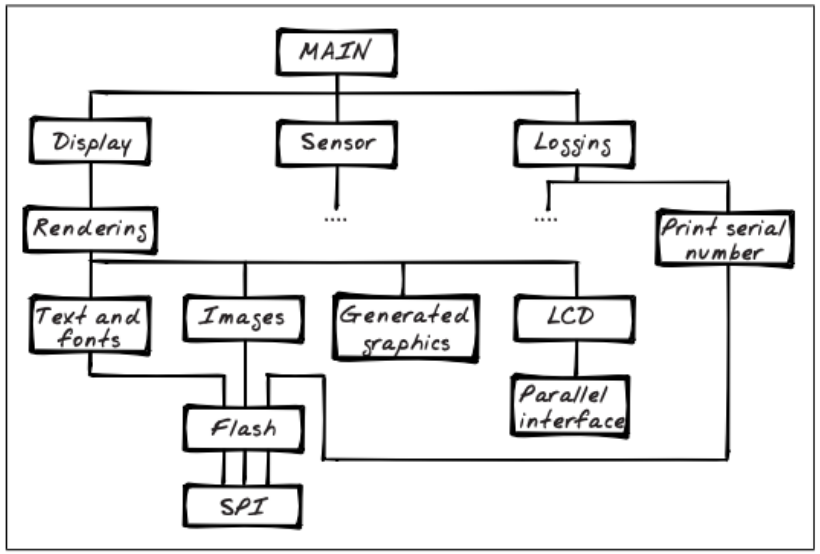
- A **layered architecture** is a very common style.
- Layers supports organising a system, i.e. *manages complexity.*
- Each layer provides a set of functionalities through a programming interface.
- **Layers can communicate only with adjacent layers**.
- Each layer abstracts underlying layers, so technologies can be changed and upper layers shouldn’t be affected (isolation).
- Some claim that large enough systems are *always layered*, to hide complexity.

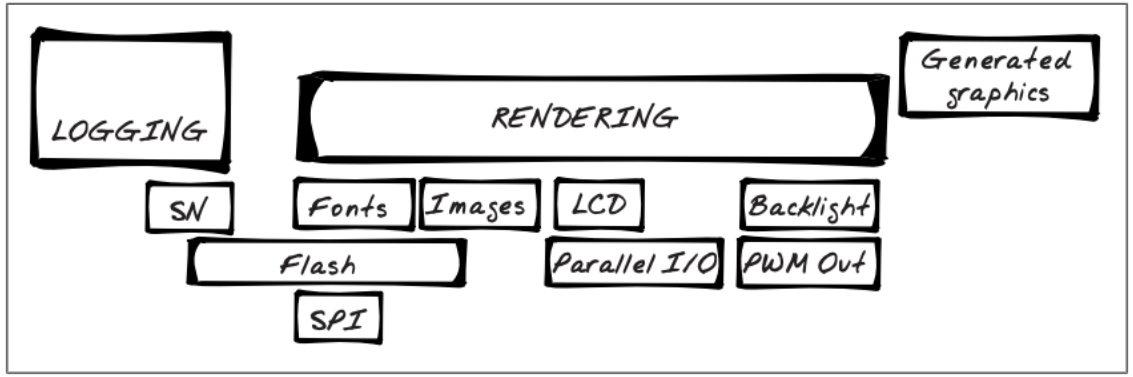
Layered view of embedded software



From blocks to layers

- Layers allow to visualise dependencies and reusable blocks easily.
- No need of spaghetti arrows!





Merging and splitting blocks

- Which modules have lots of interdependencies? Can they be broken apart and simplified? Or can the dependencies be grouped together?

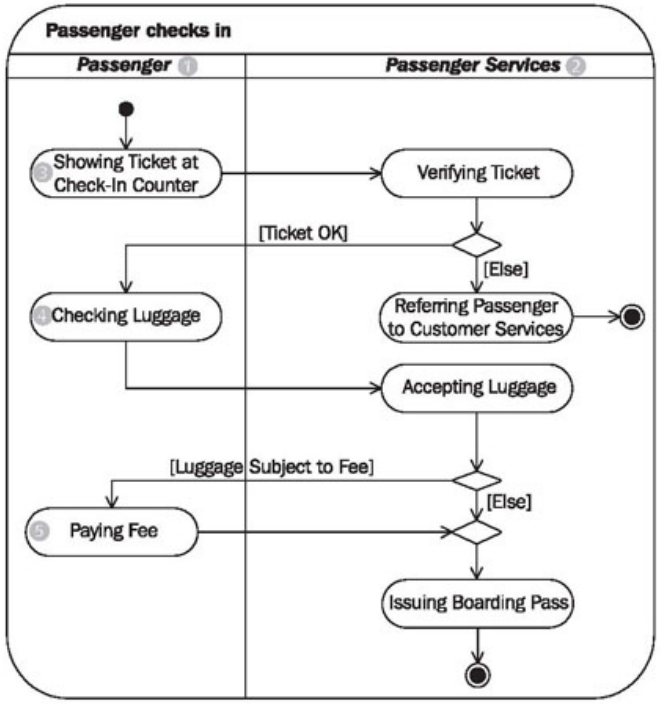
- In the hierarchy chart, look for objects that are *used only by one other object*. Are these both fixed? Or can one change independently of the other? (e.g. display and rendering)
- In the layered diagram, look for collections of *objects that are always used together*. Can these be grouped together in a higher level interface to manage the objects?
- Are blocks relatively of the same size? Is there any block that much bigger than others? Maybe that block needs to be split into sub-blocks?

To think about

- Take your diagram of your smart fridge and re-order the blocks.
- Can you spot any that could be merged because always used together?
- Can you spot any that has many incoming connections? Maybe it needs to be split?

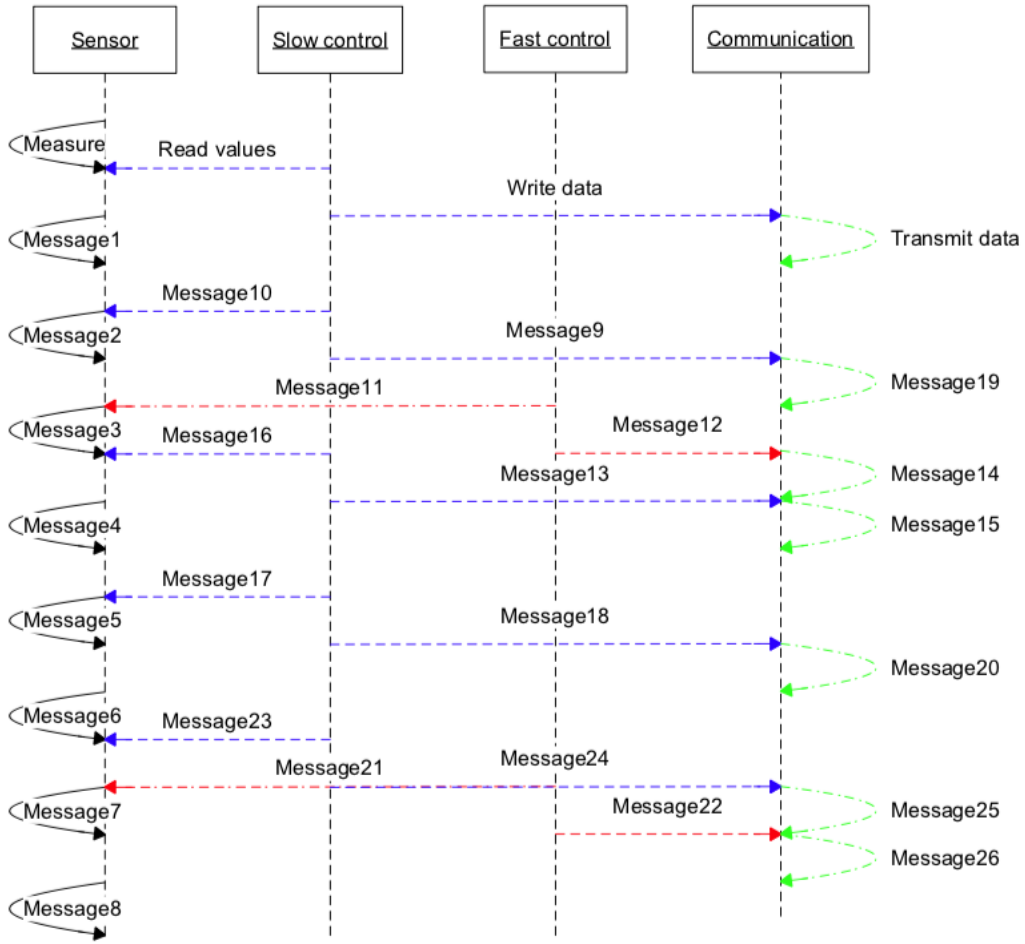
Process view

- Addresses concerns of **concurrency** and **distribution**, of system's **integrity**, of **fault-tolerance**, and how the main abstractions from the logical view fit within the process view.
- May address quality attributes, such as performance and availability.
- Typical states for processes/tasks are:
  - Started
  - Recovered
  - Reconfigured
  - Shut down
- Typical relationships between processes/tasks are:
  - Waits for
  - Initiates
  - Shares
  - Broadcasts
  - In parallel



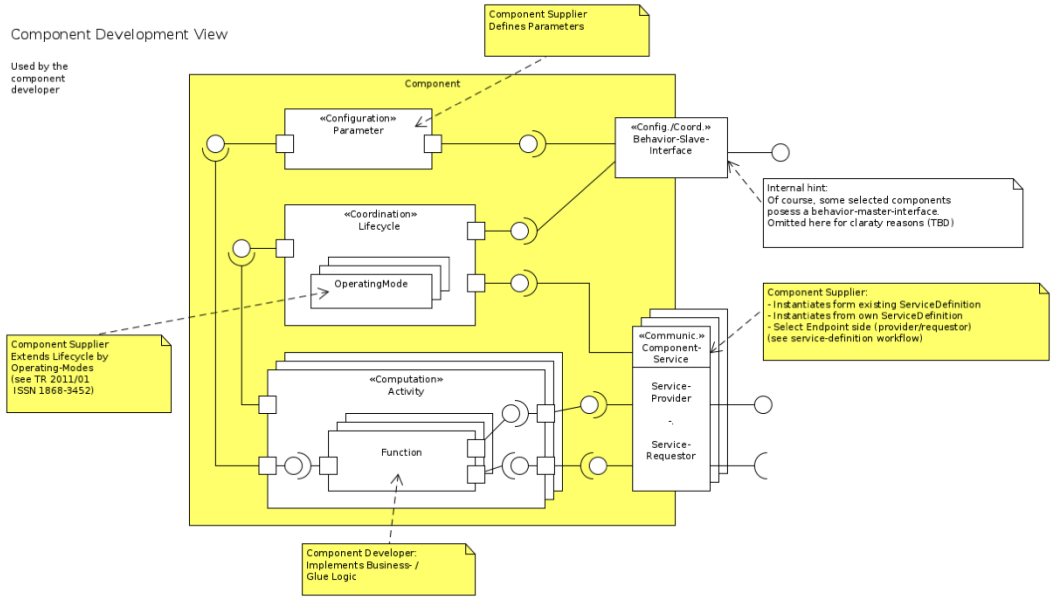
Example of process view

- Sensor process executes with fast period to always have fresh data.
- The two control algorithms have different deadlines, i.e. different periods.
- The communication process pushes the data to the network as soon as it gets something from a control loop.



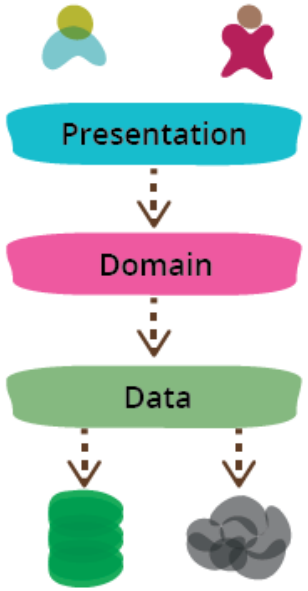
Development view

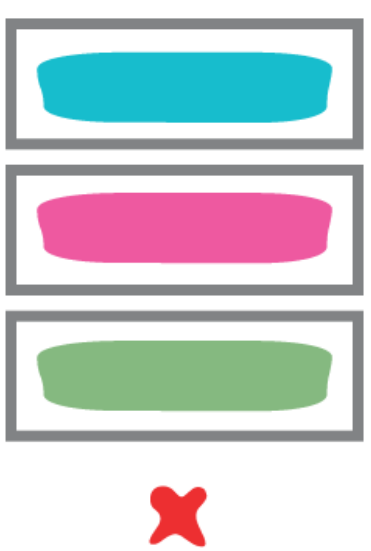
- The development architecture focuses on the on the software development environment.
- The software is packaged in small chunks - program **libraries**, or **subsystems** - that can be developed by one or a small number of developers.



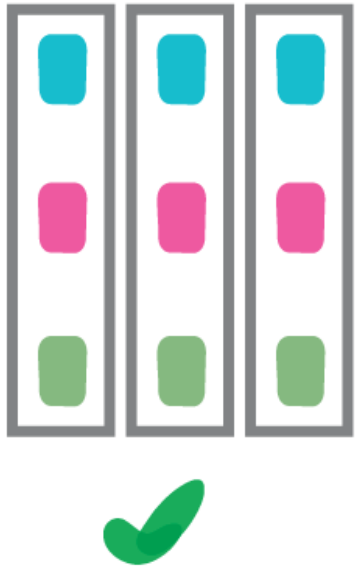
Development view can be arranged in layers

- In complex software this may not be a good idea!





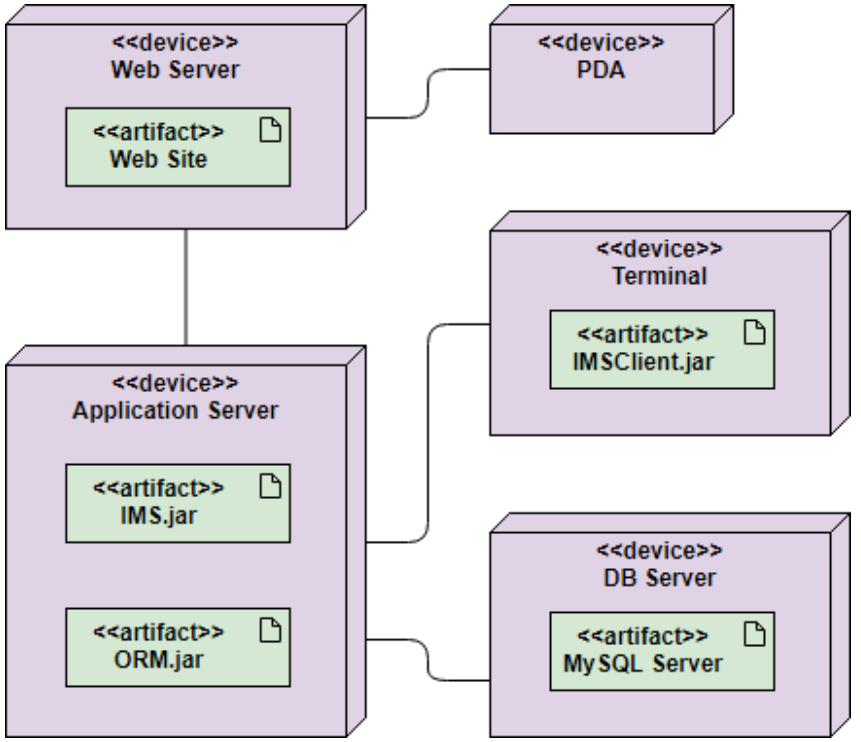
Don't use layers as the top level modules in a complex application...



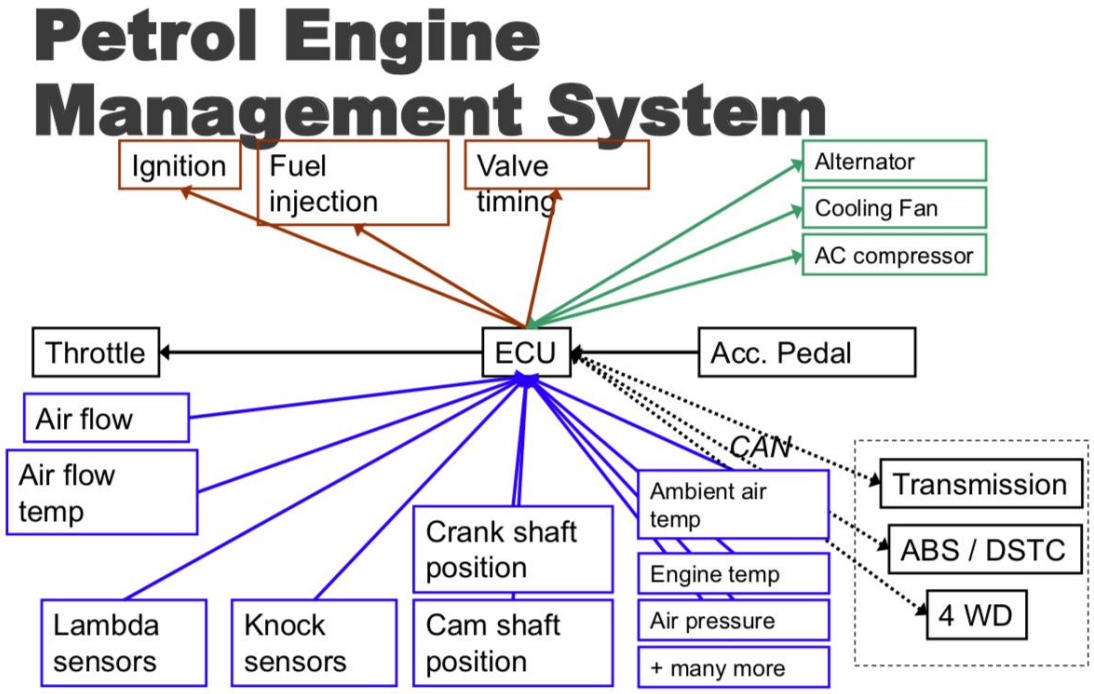
... instead make your top level modules be full-stack

Physical view

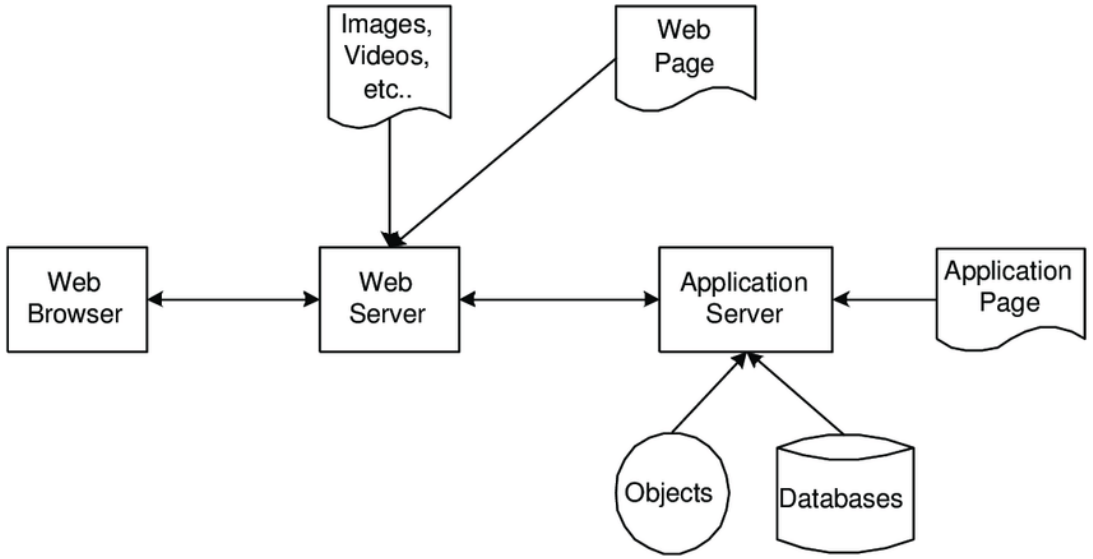
- The software executes on a network of computers, or **processing nodes** (or just nodes for short).
- The various running elements identified need to be mapped onto the various nodes.
- Different physical configurations can be used:
  - Some for development and testing.
  - Others for the deployment of the system for various sites or for different customers.



Examples



Web application



Scenarios

- Scenarios (**instances of more general use cases**) show how the elements in the four views are shown to work together.
- The scenarios are in some sense an abstraction of **the most important requirements**.
- This view is redundant with the other ones (hence the “+1”), but it serves two main purposes:
  - As a developer to **discover** the architectural elements during the architecture design.
  - As a **validation** and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

Example scenarios

- Startup/shutdown/restart
  - How often?
- User interaction
  - What is the role of the user?
  - Does the system need to wait for user input?
  - What are the interaction flows?
- Software updates
  - Hot-swap or restart?
- Change of configuration and parameters
  - During run-time?
- Fault handling
  - What is the cause of the fault?
  - Desired behaviour in presence of faults?

To think about

- Write some scenarios for your smart fridge.

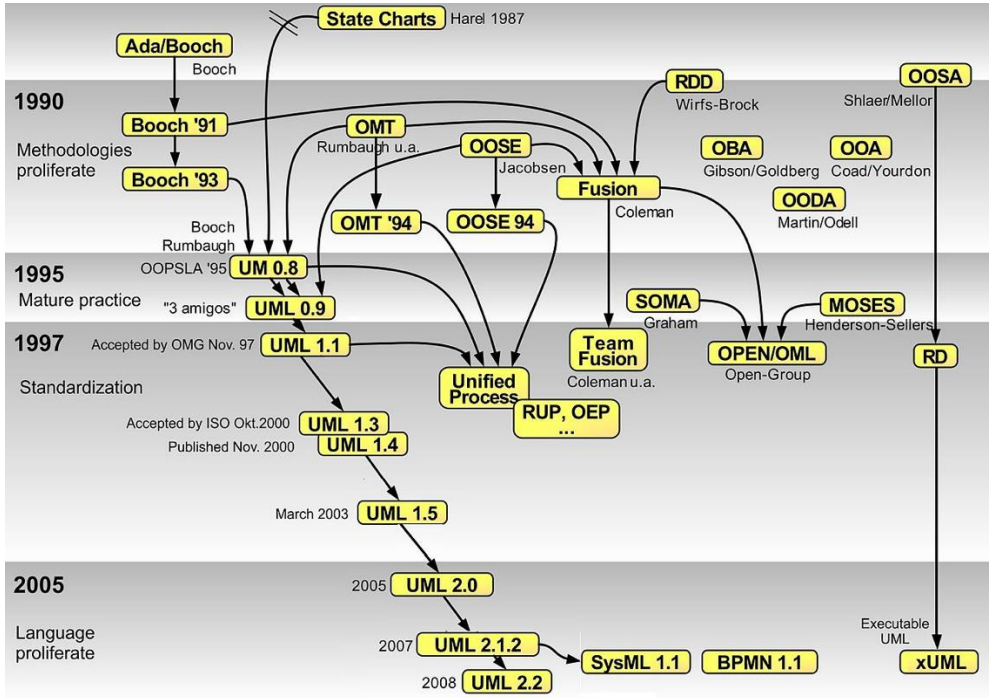
Choosing the correct diagram

- Drawing diagrams is **an art**.
- Diagrams must fulfil a **communication need**:
  - The same system can be *viewed* in different ways depending on the *viewer*.
  - Example: CEO, technical manager, developer of a component, hardware engineer, final user.
- There are maaaany diagram types you can choose from, and you can make your own!

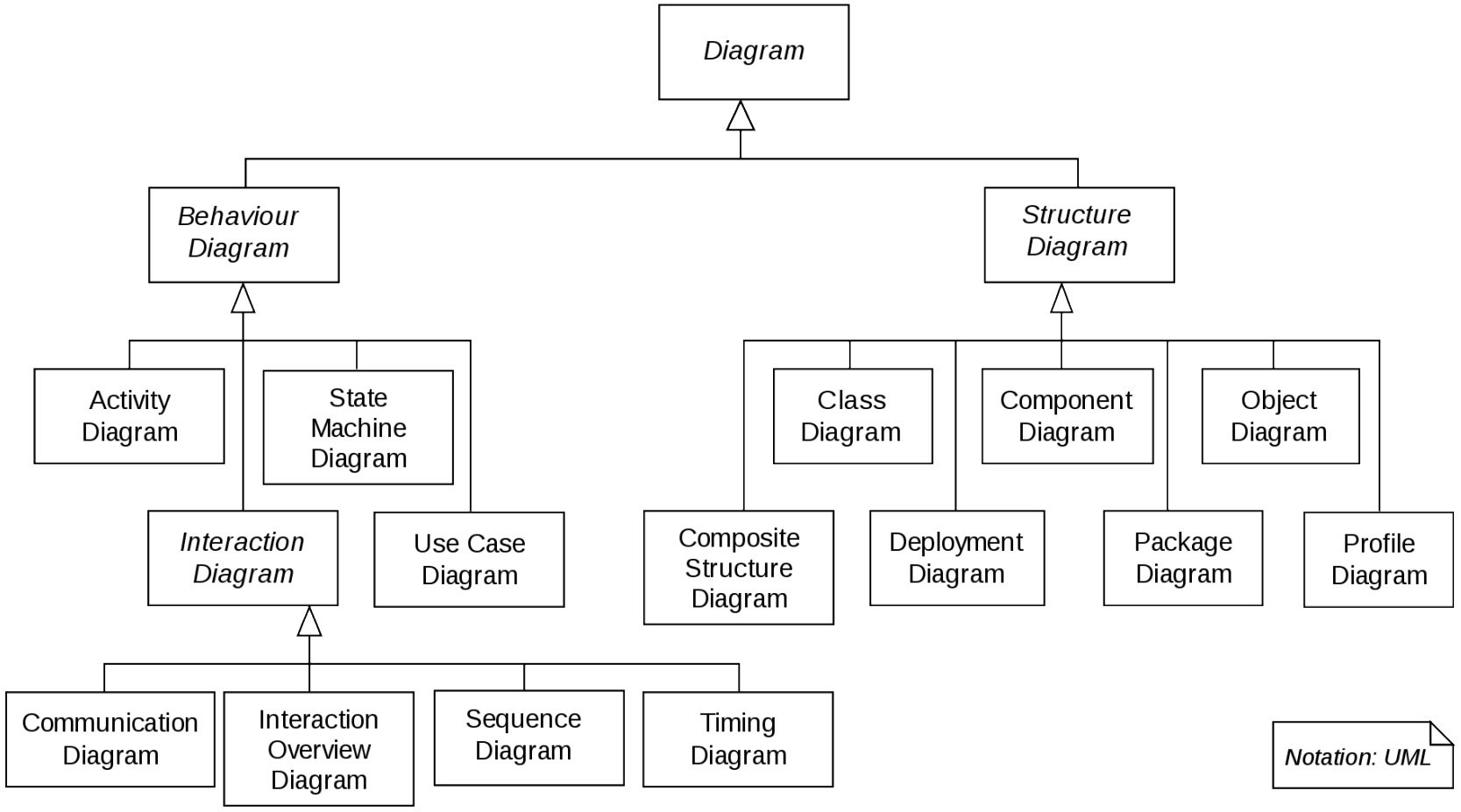
- Important: formalities are not important, the message is what counts!

UML

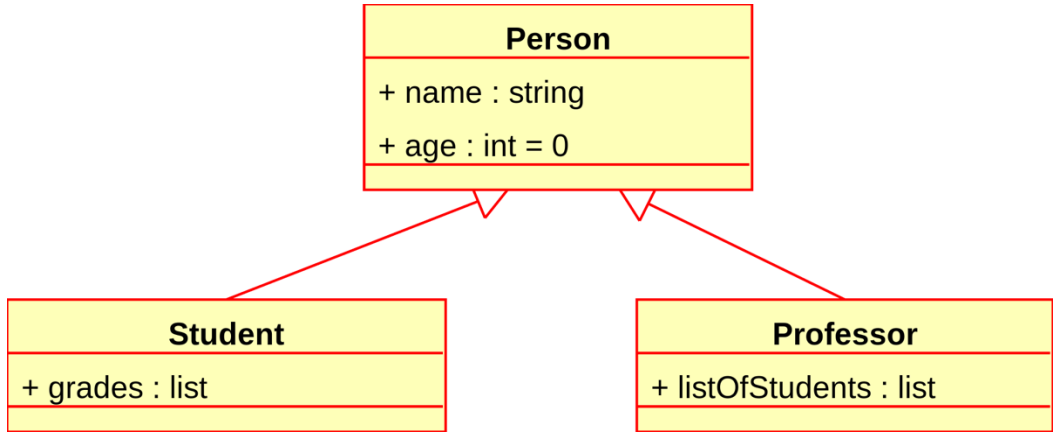
- **Unified Modeling Language** (UML) is a general-purpose, developmental, modelling language in the field of software engineering.
- Invented in the 90s, but later evolved.
- Captures **structure** and **behaviour**.
- There are many diagrams you can use.
- Specialised software to draw diagrams.
- Code can be directly converted into diagrams and vice versa.



UML diagrams



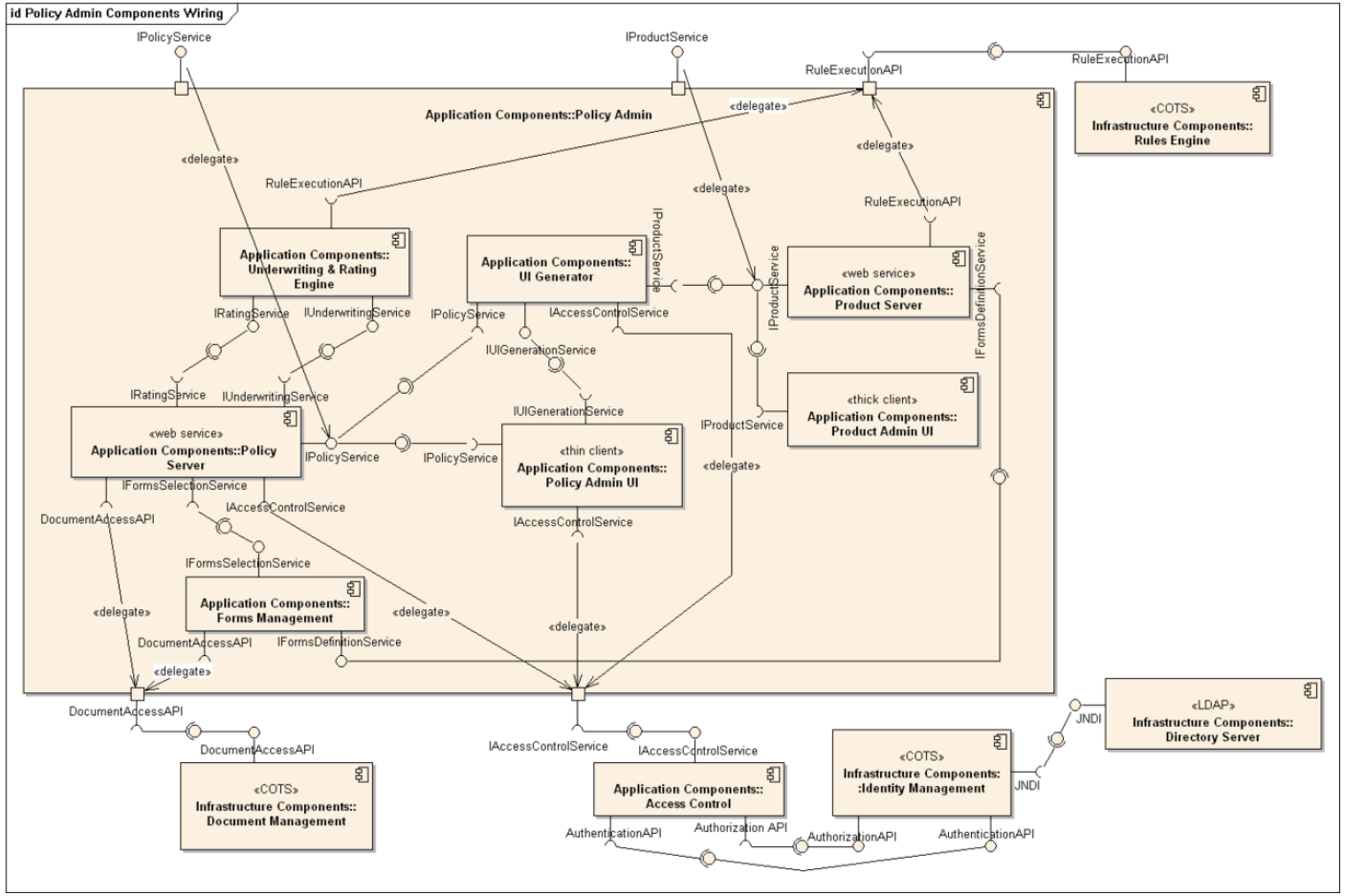
Class diagrams



- Describes the structure of a system by showing the system's classes (or modules), their attributes, operations (or methods), and the relationships among objects.
- Very relevant in Object Oriented programming.
- OO can be mimicked in C too (see extern and static) !

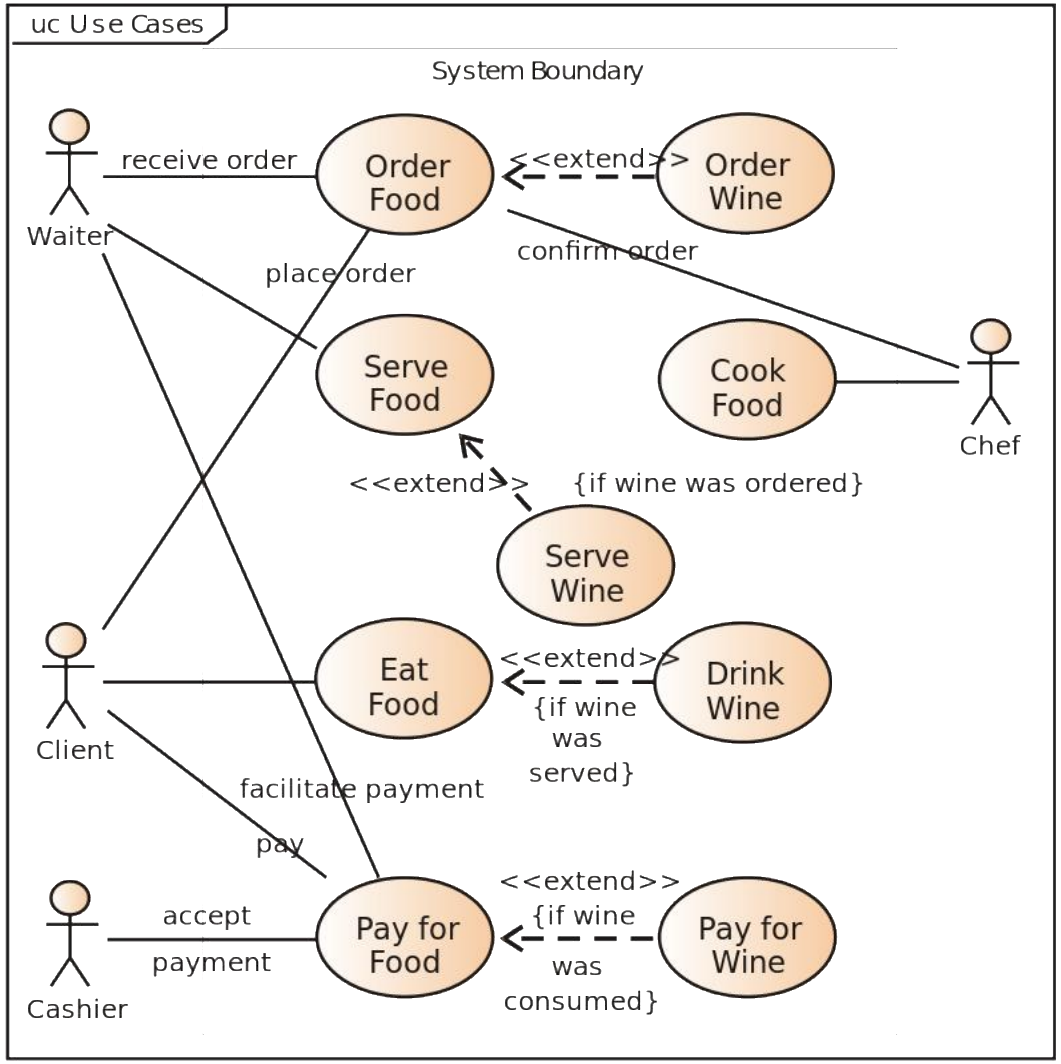
Component diagram

- A component diagram depicts how components are wired together to form larger components .
- Components are connected through parts of their programming interfaces.



Use case diagram

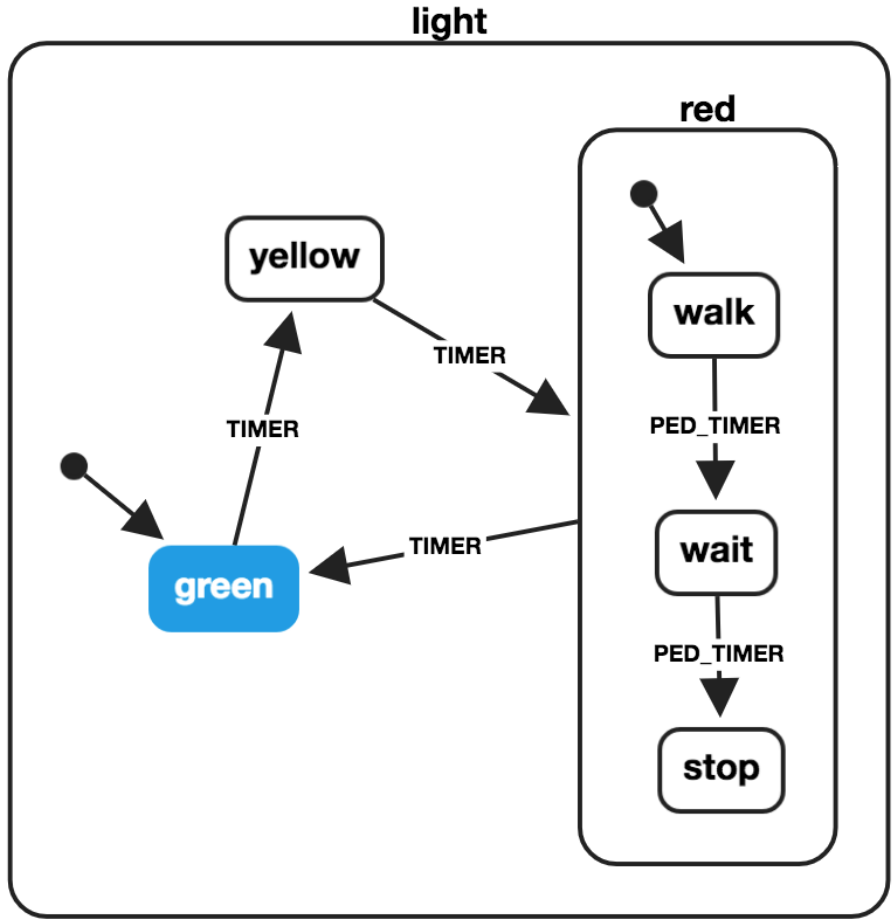
- A use case diagram is a representation of a **user's interaction with the system**.
- Shows the relationship between the user and the different use cases in which the user is involved.
- Different users can be represented, **also non-human**!
- Use cases can have hierarchies (*order wine* is a sub-case of *order food*)



State machine

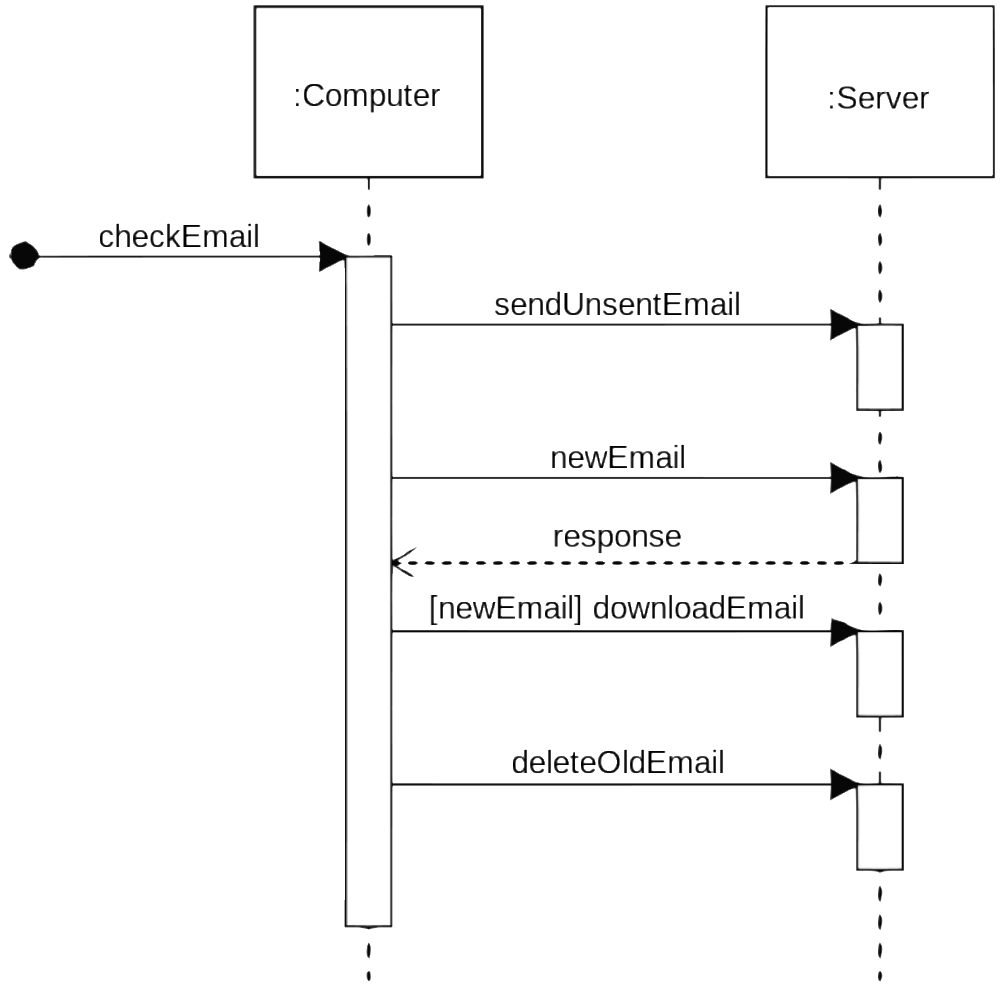
- Represents states the system can be in.
- **States are linked by transitions**.
- Transitions can be triggered by external events or by the code itself.
- All possible states should be captured.

- One of the most useful diagrams!



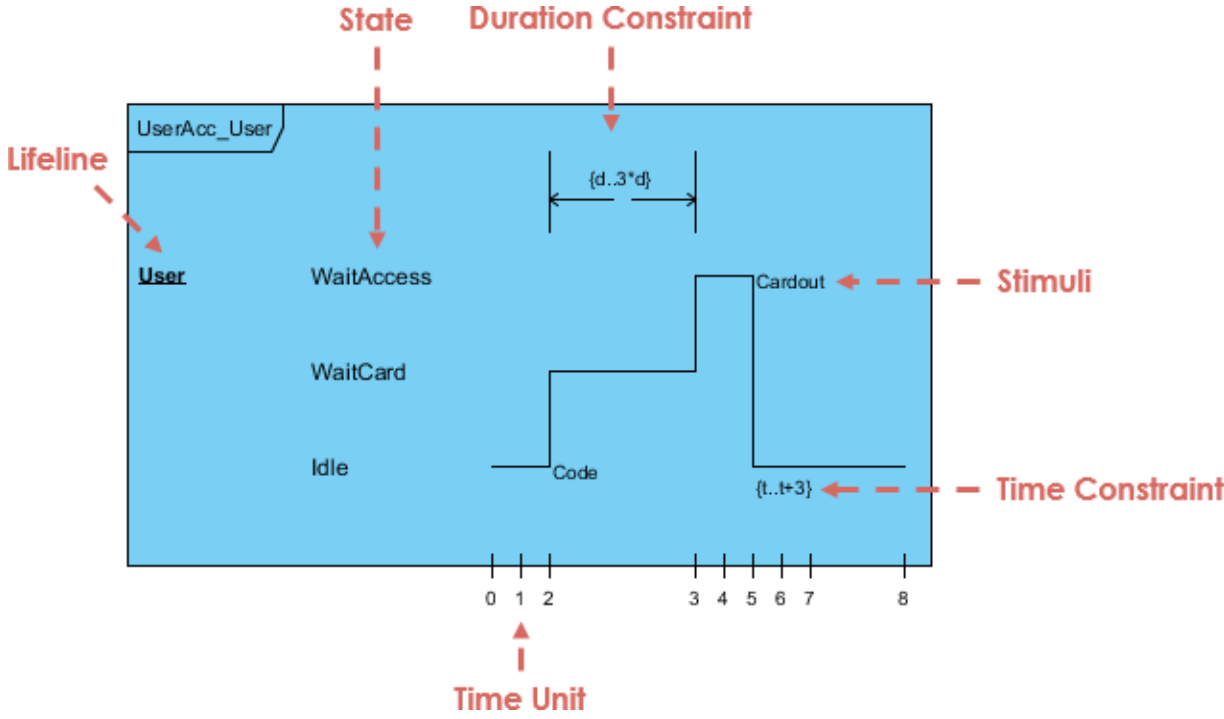
Sequence diagram

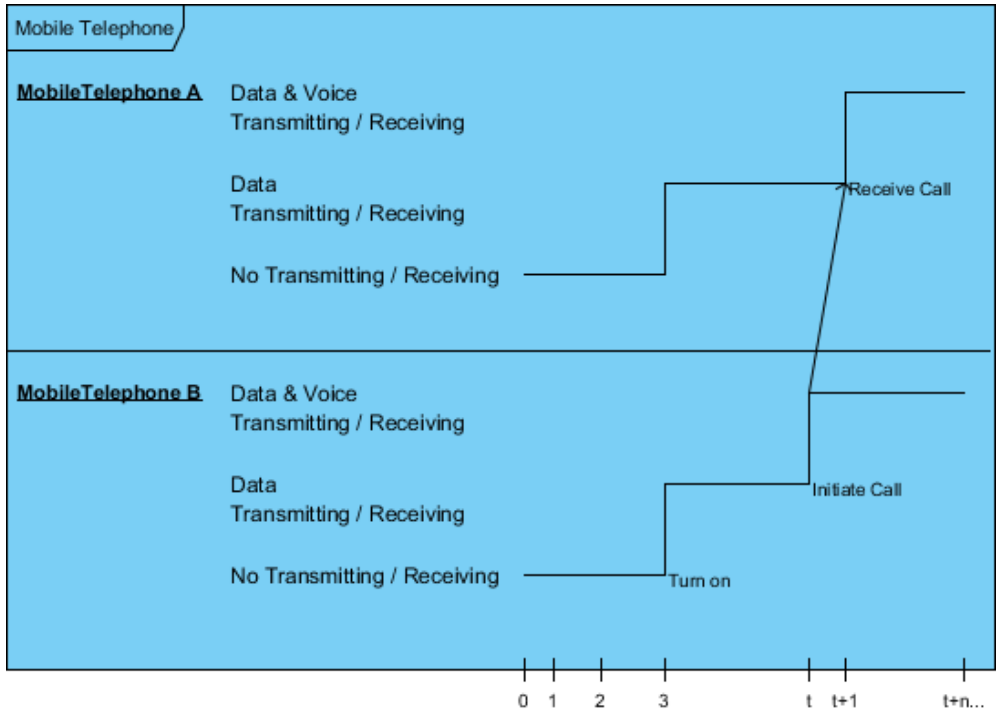
- A sequence diagram shows object interactions arranged in **time sequence**.
- Typically mapped to use cases.
- It depicts the objects and classes involved in the use case and the **sequence of messages** exchanged between the objects needed to carry out the use case.
- Extremely useful!



Timing diagram

- A timing diagram is a specific behavioural modelling diagram that focuses on timing constraints.
- It is similar to the sequence diagram, but **time is explicitly expressed**.
- Useful for some real time scenarios !





The problem with UML

- UML is bloated.
- UML stresses the design aspects too much.
- UML implies **a lot** of documentation time.
- Agile methods (develop quickly, deploy, evaluate, restart) is not compatible with heavy documentation.

- Its usefulness is still controversial!

https://www.quora.com/Do-prestigious-software-companies-regularly-use-UML

https://news.ycombinator.com/item?id=7624601

https://www.reddit.com/r/learnprogramming/comments/46118w/oo_do_people_still_use_uml_if_not_then_what/

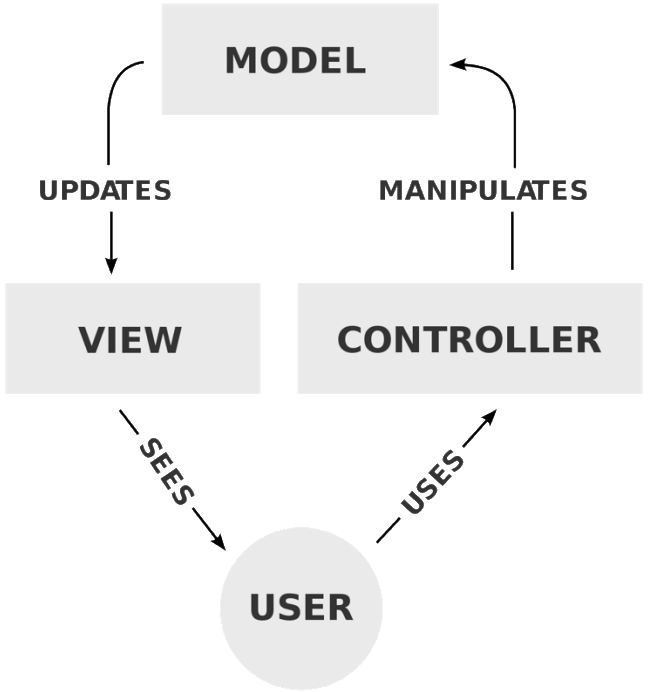
	Logical	Process	Development	Physical	Scenario
Description	Shows the component (Object) of system as well as their interaction	Shows the processes / Workflow rules of system and how those processes communicate, focuses on dynamic view of system	Gives building block views of system and describe static organization of the system modules	Shows the installation, configuration and deployment of software application	Shows the design is complete by performing validation and illustration
Viewer / Stake holder	End-User, Analysts and Designer	Integrators & developers	Programmer and software project managers	System engineer, operators, system administrators and system installers	All the views of their views and evaluators
Consider	Functional requirements	Non Functional Requirements	Software Module organization (Software management reuse, constraint of tools)	Nonfunctional requirement regarding to underlying hardware	System Consistency and validity
UML – Diagram	Class, Object Diagram	Activity, State, Timing, Sequence Diagram	Component, Package diagram	Deployment diagram	Use case diagram

To think about

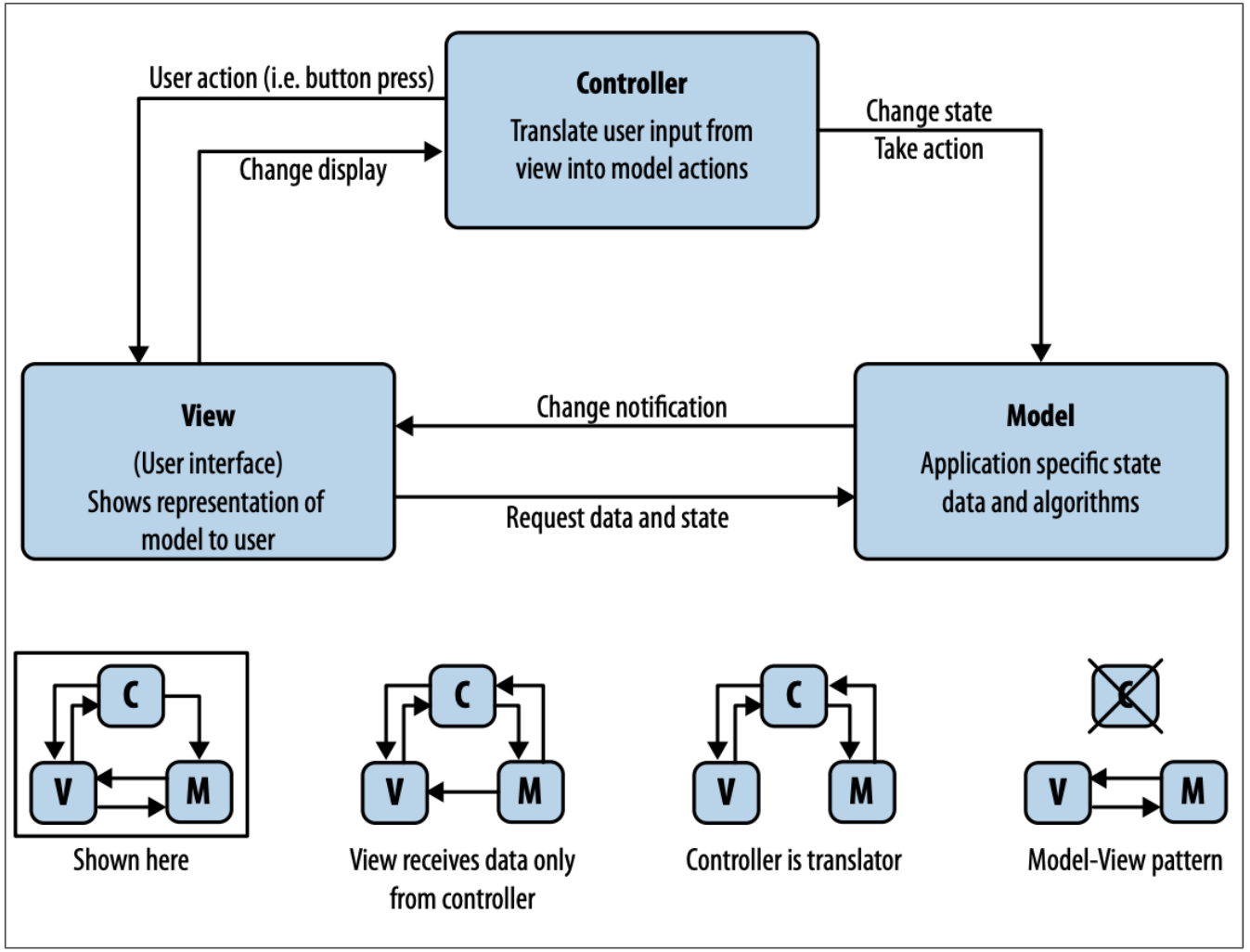
- In your smart fridge, which diagram(s) would you use?
  - Class diagram?
  - Component diagram?
  - Use case diagram?
  - State machine?
  - Sequence diagram?
  - Timing diagram?
- Try sketching one!

Model View Controller

- MVC is an architectural pattern commonly used for developing user interfaces that divides an application into:
  - **Model**:
    all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
  - **View**:
    all the UI logic of the application.
  - **Controller**:
    acts as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.



How many MVCs?



Common pitfalls when drawing diagrams

- **Inconsistency**. For example different diagram elements or colours used differently.
- Diagram elements, colours etc. **not well explained**.
- **Key relationships** between diagram elements missing or ambiguous.
- **Key technology** choices (or options) omitted.
- **Mixed levels** of abstraction.
- Trying to show **too much detail**.
- No clear ”message” or “viewer”.

Project files structure in C

- There are several ways to organise the code in your project.
- In most cases the IDE will suggest a structure for you. Use that unless you have a good reason for not doing so.
- A common structure is:
- repository folder/
  - **src/** source code
  - **tests/** automatic tests
  - **libs/** 3rd party libraries folder
  - **docs/** documentations
  - project file
  - README file
  - LICENSE file

Some best practices

- In large projects with many source files use a folder structure under `src` where **folders and subfolders mimic the structure of the code**, e.g. major modules or FreeRTOS tasks.
- Keep the h-files in the same folder as the c-files they are associated to.
  - This makes it much easier to find and update them at the same time at the expense of more effort when defining in which folder the compiler should look for declarations.
  - The exception is if the h-file is associated with a 3rd-party library.
- Never have declarations of functions in a c-file.
- Create an h-file even if they are only used in one place.
- Each c-file always includes its own h-file.

Code structure

1. Bootloader
   - Program updates etc
2. Initialisation
   - Hardware
   - Scheduling
3. Application
   - Run tasks

Identifying tasks in an RTOS

Inside-out: starting from the logical architecture

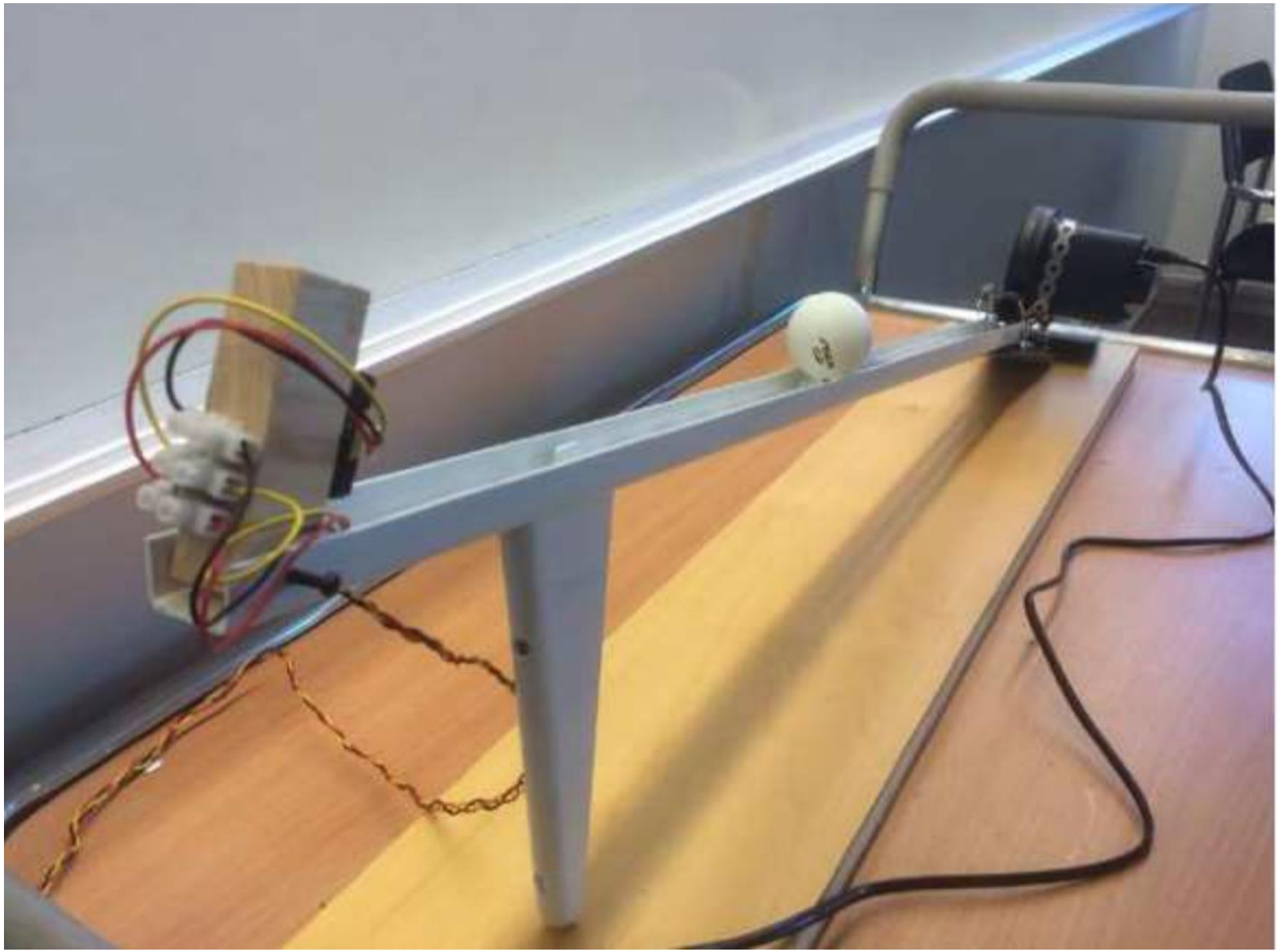
- Define tasks which:
  - Multiplex a single thread of control across multiple active modules;
  - Modules whose persistency or life is subordinate to another active module are also executed in the same task;
  - Several modules that need to be executed in mutual exclusion, or that require only small amount of processing share a single agent.
- This clustering proceeds until we have reduced the tasks to a reasonably small number that still allows distribution and use of the physical resources.

Outside-in: Starting from the physical architecture

- Identify external stimuli (requests) to the system.
- Define tasks to handle the stimuli.
- Define the right set of synchronisation tools, e.g. mutex.
- Identify modules for the various tasks.
- Identify which modules must be simultaneous.

Example

- A fan has to keep a Ping-Pong ball in a constant position on a slope with the help of a control algorithm.
- The fan at the bottom of the slope and the distance sensor at the top are both connected to the same development board.
- In parallel to executing the control the development board should be connected to the seerial port of a PC running Matlab.
- The Matlab program is out of scope for this problem, but it can change the set value of the position of the ball, as well as change the variables for control.
- Real-time results of the control are displayed as graphs in Matlab:
  - the control value to the fan
  - the error in the position
  - the actual position of the ball



https://www.slideshare.net/JamesGoddings/position-control-of-pp-ball

https://www.youtube.com/watch?v=dXYT1-Ft_l4

Example: tasks decomposition

- **Task 1** with the highest priority carries out the control of the fan itself.
  - Reads the distance value, then controls the fan.
  - Sampling is scheduled with RTOS with a period between 50ms - 100ms.
  - Why this timing?
- **Task 2** with the lowest priority handles the transfer of variables between Matlab and Arduino via the serial serial port (USB).
  - Matlab should be able to set control values like distance and the sampling time.
  - At the same time, the result of the control should be shown as a graph in Matlab.

Example : fitness tracker

- A device includes a microcontroller, an accelerometer, a touch-based user interface and a Bluetooth Low Energy (BLE) chip.
- The device should measure the number of steps walked with a known algorithm.
- We should save battery as much as possible.

Example: Modules

1. Get acceleration from sensor.
2. Step counting (accelerometry -> steps).
3. Data storage (read/write on persistent memory).
4. Communication (data sync with mobile app).
5. UI (responds to user interaction).
6. Control.

These could work independently.

Example: Tasks

1. Samples accelerometer and stores over a buffer.
   - Priority: high.
   - Period: 10 Hz
2. Counts steps, stores the value
   - Priority: medium.
   - Period: 1 Hz
3. BLE service: if connection is established, deals with sending data.
   1. Priority: medium.
   2. Period: no periodicity if connected (send and wait for replies)
4. If any user interaction, reacts by showing info on display
   1. Priority: low
   2. Period: no periodicity, can just react to interrupts
5. Switches subsystems off if device is not used (e.g. switches display off after X seconds if no interaction, slows down steps detection if no activity in accelerometer, CPU is switched off when idle).
   1. Priority: low
   2. Period: seconds

Example exam questions

- How would you define a software architecture?
- What are the 5 views of the 4+1 view model? Describe them with your words.
- What UML diagram or diagrams would you use for the logic view of 4+1 view model?
- Describe the MVC design pattern.
- How could the MVC design pattern fit inside the 4+1 view model?
- Design the physical view and the logic view of the following embedded system:
  - A medical wearable device is placed on the chest of the user. The device has two electrodes and an amplifier from which it can sense the electrical activity of the heart (ECG). From the ECG, the device can extract information about the heart activity, like the heart rate (beats per minute) or indications of the heart's health like the presence of atrial fibrillations. These extracted parameters are stored on a memory. The device can send the parameters over Bluetooth to an app on the user's smartphone.
  - You don't need to draw the logic view of the app.