Dealing with Software Complexity

Prof. A. M. Calvagna

Dealing with Complexity

Object oriented design give us these tools:

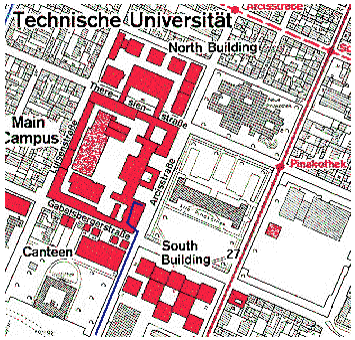
1. Abstraction
2. Decomposition
3. Hierarchy

1. Abstraction

- Inherent human limitation to deal with complexity
  - — The 7 +- 2 phenomena
- Chunking: Group collection of objects
- Ignore unessential details: => Models

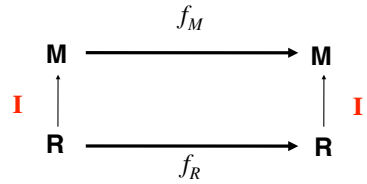
What is modeling?

- Modeling consists of building an abstraction of reality.
- Abstractions are **simplifications** because:
  - — They ignore irrelevant details and
  - — They only represent the relevant details
- What is relevant or irrelevant **depends on the purpose** of the model.



What is a "good" model?

- I : Mapping of real things in reality R to abstractions in the model M abbildet (Interpretation)
- f_M: relationship between abstractions in M
- f_R: relationship between real things in R



Relationships, which are valid in reality R, must also be valid in model M.

Systems, Models and Views

- A ***model*** is an abstraction describing a subset of a system
- A ***view*** depicts selected aspects of a model
- A ***notation*** is a set of graphical or textual rules for depicting views
- Views and models of a single system may overlap each other

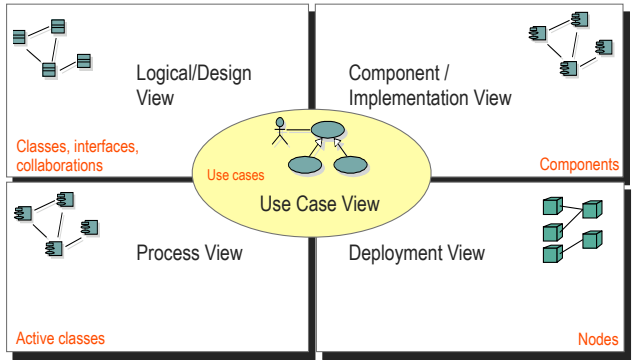
Examples:

- System: Aircraft
- Models: Flight simulator, scale model
- Views: All blueprints, electrical wiring, fuel system

OO modelling methodologies

- Late 80's early 90: several OO methodologies developed
  - different notations
  - different processes
- Main approaches
  - Booch
  - Rumbaugh
  - Jacobson
- Unificati in UML...

UML ha 4+1 'viste'



Model-based software Engineering:

Problem Statement : A stock exchange lists many companies.
Each company is identified by a ticker symbol

Analysis phase results in object model (UML Class Diagram):

Implementation phase results in code

```
public class StockExchange
{
 public Vector m_Company = new Vector();
};
```

```
public class Company
{
 public int m_tickerSymbol
 public Vector m_StockExchange = new Vector();
};
```


Why model software?

Software is getting increasingly more **complex**

Windows 10 > 40 millions lines of code

A single programmer cannot manage this amount of code in its entirety.

Code is not easily understandable by developers who did not write it

We need simpler representations for complex systems

Modeling is a mean for dealing with **complexity**

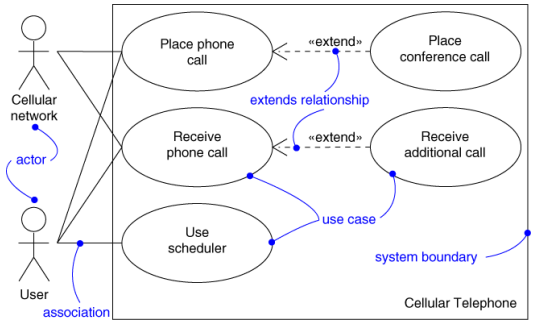
Models are used to provide abstractions

- System Models:
  - **Functional model**: What are the functions of the system? How is data flowing through the system? ===>(**use case diagrams**)
  - **Object Model**: What is the structure of the system? What are the objects and how are they related? ====>(**class diagrams**)
  - **Dynamic model**: How does the system react to external events? How is the event flow in the system ? ==>(**sequence/activity/state diagrams**)

Use Case Diagram: Functional modelling

An actor models an external entity which communicates with the system:
User
External system
Physical environment

Use Cases are the user-tangible functionalities



- Bridging the gap between user and developer:
  - *Scenarios:* **Example of the use of the system in terms of a series of interactions with between the user and the system (one story card)**
  - *Use cases:* **Abstraction that describes a class of scenarios**

Analysis: Object Modeling

How to identify objects from requirements

Finding Participating Objects in Use Cases

- Pick a ***use case*** and look at its ***flow of events***
  - Look for recurring nouns
  - Identify real world entities NAMES that the system needs to keep track of
  - Identify real world PROCEDURES that the system needs to keep track of
  - Identify DATA sources or sinks (e.g., Printer)
  - Identify interface ARTIFACTS (e.g., PoliceStation)
- Expect that some objects are still missing and need to be found:
  - Model the flow of events (in user stories/use cases) with **activity diagrams**

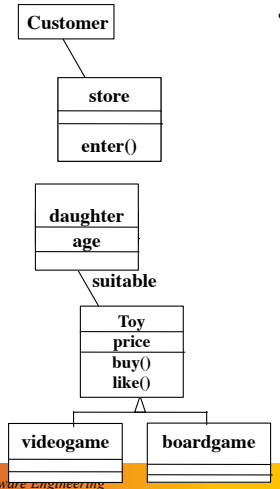
Example: Flow of events

- The customer enters a store with the intention of buying a toy for his child with the age of n.
- Help must be available within less than one minute.
- The store owner gives advice to the customer. The advice depends on the age range of the child and the attributes of the toy.
- The customer selects a dangerous toy which is kind of unsuitable for the child.
- The store owner recommends a more suitable doll.

Mapping parts of speech to object model components [Abbott, 1983]

Part of speech	*Model component*	*Example*
Proper noun	object	Jim Smith
Improper noun	class	Toy, doll
Doing verb	method	Buy, recommend
being verb	inheritance	is-a (kind-of)
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	use
intransitive verb	method (event)	enter, depends on

Generation of a class diagram from flow of events

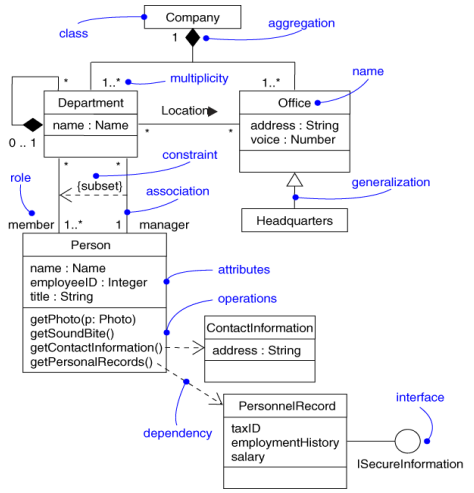


- The **customer enters** the **store** to **buy** a toy. It has to be a **toy** that his **daughter** likes and it must cost **less than 50 Euro**. He tries a **videogame**, which uses a data glove and a head-mounted display. He likes it.

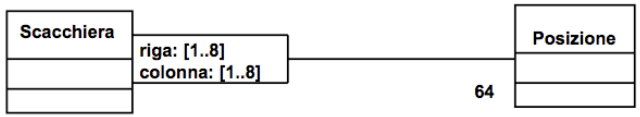
An assistant helps him. The suitability of the game **depends** on the **age** of the child. His daughter is only 3 years old. The assistant recommends another **type of toy**, namely a **boardgame**. The customer buy the game and leaves the store

Class Diagram

Captures the vocabulary of a domain or a system



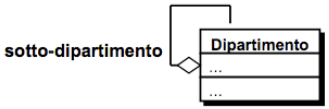
Associazioni qualificate



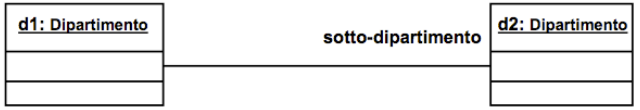
- **Un qualificatore è un attributo di un'associazione**
- **Il qualificatore individua univocamente un oggetto tra quelli collegati dall'associazione**

Ruoli

- **Autorelazione nel diagramma delle classi**



- **I ruoli discriminano tra oggetti dello stesso tipo**



Summary: Order of activities in modeling

1. Formulate a few scenarios with help from the end user and/or application domain expert.
2. Extract the use cases from the scenarios, with the help of application domain expert.
3. Analyse the flow of events, for example with Abbot's textual analysis.
4. Generate the class diagrams, which includes the following steps:
   1. **Class identification (textual analysis, domain experts).**
   2. **Identification of attributes and operations (sometimes before the classes are found!)**
   3. **Identification of associations between classes**
   4. **Identification of multiplicities**
   5. **Identification of roles**
   6. **Identification of constraints**

Analysis: Dynamic Modeling

Dynamic Modeling

- Definition of dynamic model:
  - A collection of multiple state chart diagrams, one state chart diagram for each class with important dynamic behavior.
- Purpose:
  - Detect and supply **methods** for the object model
- How do we do this?
  - Start with use case or scenario => activity diagram
  - Model interaction between objects => sequence/interaction diagram
  - Model dynamic behavior of a single object => statechart diagram

Diagramma delle attività

- Utilizzato per la modellazione di processi e workflow
- Mostra dettagli di funzionamento interno del sistema software da realizzare
- E' una vista sull'esecuzione delle attività
- Mostra dipendenze tra attività (o passi)

Attività: Apri file da browser

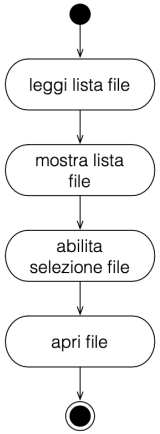


Diagramma attività UML per ordini

Notazione grafica per diagrammi UML delle attività

- Rettangoli arrotondati: passi di elaborazione
- Frecce continue: flussi
- Barre: sincronizzazione per fork o per join
- Rettangoli: oggetti o dati in input o in output
- Rombi: ramificazioni condizionali o merge
- Cerchi pieni: stati iniziale e finale (con circonferenza)

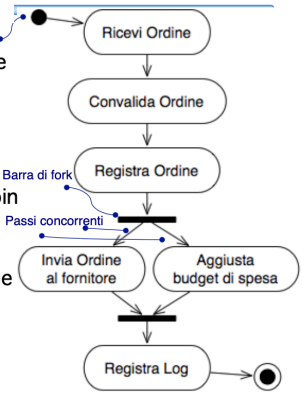


Diagramma per pagamento

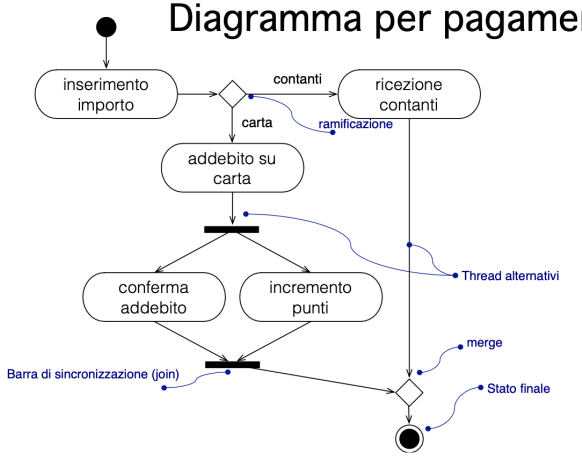
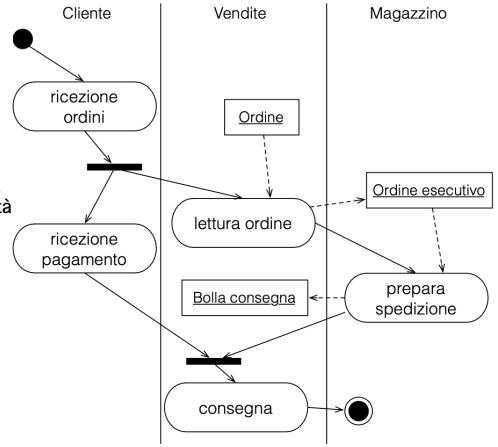


Diagramma attività con Corsie

- Corsie o Swimlane: sono partizioni che indicano il responsabile (attore, o modulo) dell'attività
- Rettangoli: oggetti o dati
- Frecce tratteggiate: input o output di un oggetto per/da una attività



Collaboration (interaction) Diagram

- Mostra interazioni tra oggetti
  - Il flusso dei messaggi è indicato da frecce accanto alle associazioni fra istanze che partecipano all'interazione
  - I messaggi sono mostrati da etichette sulle frecce ed hanno
    - Un numero sequenziale che indica l'ordine temporale con cui avvengono
    - Il metodo chiamato
    - Un valore di ritorno (opzionale)

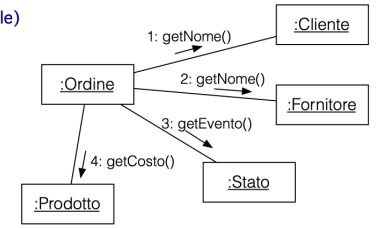


Diagramma di collaborazione per Stampa Ordini

Diagramma di sequenza

- Mostra interazioni tra oggetti
  - L'asse temporale è inteso in verticale
  - In alto in orizzontale ci sono i vari oggetti che ne prendono parte
    - In ciascuna colonna verticale, se l'oggetto che partecipa esiste è indicato con una linea tratteggiata, se è attivo con una barra (di attivazione)
  - Un messaggio è una freccia dalla barra di attivazione di un oggetto ad un altro
    - Frecce piene indicano comunicazione sincrona, viceversa vuote asincrona

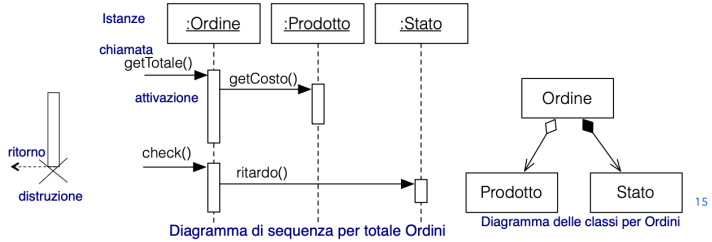
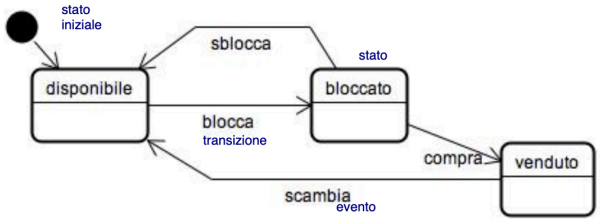


Diagramma di sequenza per totale Ordini

Diagramma delle classi per Ordini

Diagramma UML degli Stati

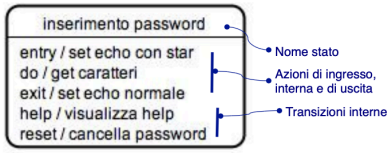
- Diagramma UML degli stati che mostra la vita di un biglietto per uno spettacolo



- Rettangoli con angoli arrotondati: stati
- Frecce: transizioni tra stati
- Cerchi pieni: stati iniziale e finale (con circonferenza)

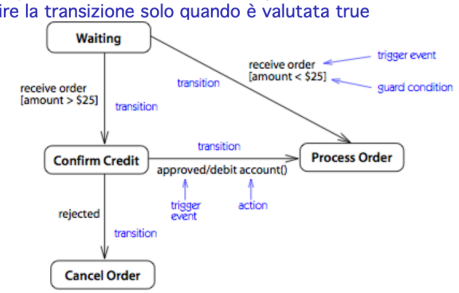
Stato

- Descrive un intervallo di tempo durante la vita di un oggetto
- È caratterizzato da
  - Valori di oggetti, o
  - Intervallo in cui un oggetto aspetta certi eventi, o
  - Intervallo in cui un oggetto fa certe azioni



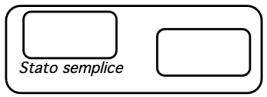
Transizione

- Permette di lasciare uno stato in risposta ad un certo evento
- Un oggetto gestisce un solo evento alla volta
- È caratterizzata da: `event-trigger [guard] / action`
  - Evento di inizio (trigger)
  - Condizione di guardia (espressione boolean), valutata quando l'evento avviene e che fa avvenire la transizione solo quando è valutata true
  - una azione
  - uno stato target

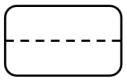


Stati composti

- Uno stato composto è uno stato che consiste di vari sottostati sequenziali o concorrenti
  - Uno stato semplice non consiste di sottostati
- Solo uno dei sottostati sequenziali può essere attivo in un certo momento
- Lo stato esterno rappresenta la condizione di essere in uno qualsiasi degli stati interni
- Una transizione verso o da uno stato composto può invocare varie azioni di entry (dalla più esterna) o exit (dalla più interna)



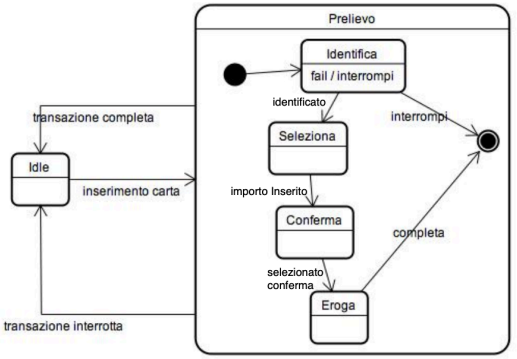
Stato composto sequenziale



Stato composto concorrente

Diagramma UML degli Stati per ATM

- Stato composto con sottostati sequenziali



Digramma degli Stati per Revisioni

- Revisione è uno stato composto con sottostati sequenziali
  - I sottostati di Revisione sono: Incompleto, Superato e Fallito
- Incompleto è uno stato composto con sottostati concorrenti
  - I sottostati di Incompleto sono: Test 1 e Test 2

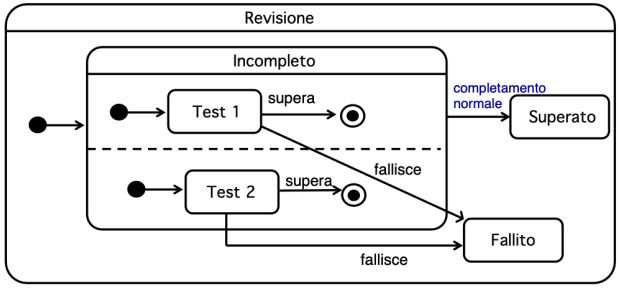
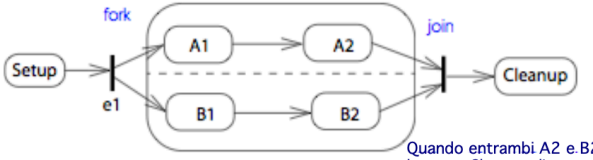


Diagramma degli Stati con fork e join

- Stato composto concorrente



Quando e1 avviene, A1 e B1 diventano attivi

Quando entrambi A2 e.B2 finiscono, lo stato Cleanup diventa attivo

2. Decomposition

- A technique used to master complexity ("divide and conquer")
- **Functional decomposition**
  - The system is decomposed into modules
  - Each module is a **major processing step (function)** in the application domain
  - Modules can be decomposed into smaller modules
- **Object-oriented decomposition**
  - The system is decomposed into classes ("objects")
  - Each class is a **major abstraction** in the application domain
  - Classes can be decomposed into smaller classes

Which decomposition is the right one?

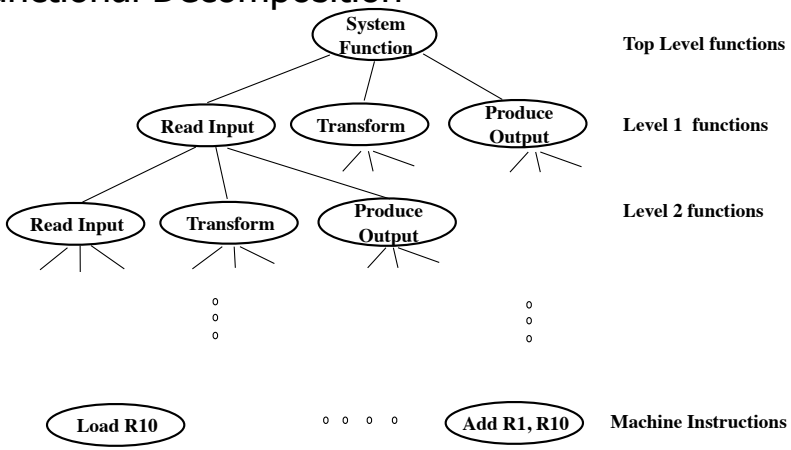
Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).

If you think you are politically correct, you probably want to answer: Object-oriented. But that is actually wrong.
Both views are important

Functional decomposition **emphasises** the ordering (of use) of **operations**

Object-oriented decomposition **emphasizes the agents that cause** the operations

Functional Decomposition



Functional decomposition

- Decompongo il goal principale in sottoproblemi (funzioni) più semplici
  - in uno stesso livello di astrazione, trovo altre funzionalità (sottoproblemi) di altri user goals di livello superiore, trovo anche ripetizioni
  - È un processo di decomposizione top-down, classico prima dell'avvento della paradigma object oriented
- Functionality is spread all over the system
  - All functions will have a common (algorithmic) structure:

    get input - transform - put output
  - La realizzazione di una funzionalità (goal e suoi sotto-goal) è **spalmata orizzontalmente e verticalmente** su tutti i livelli sottostanti di decomposizione
  - Non c'è alcuna traccia di modularità in questa decomposizione…

Functional Decomposition

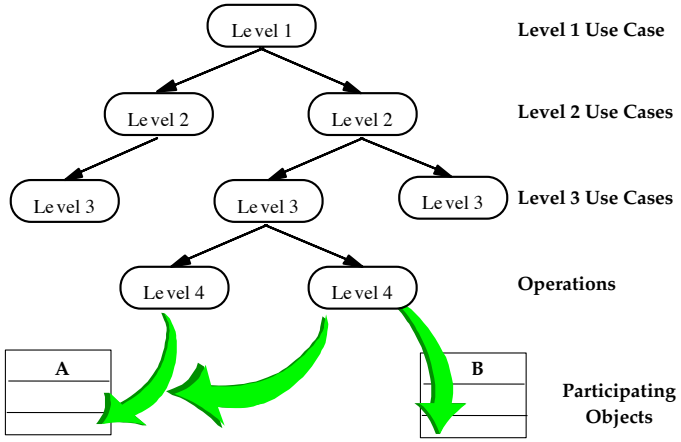
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - Codes are hard to understand
  - Code that is complex and impossible to maintain
  - User interface is often awkward and non-intuitive

Devo fermarmi ad un livello di granularità corretto, non decomporre all'infinito

La granularità corretta è quella degli oggetti del dominio: che devo avere già identificato

From Use Cases to Objects

Mi fermo quando le funzionalità utente sono ad un livello di astrazione che corrisponde a responsabilità associabili agli oggetti che ho identificato nel sistema



Class identification: Modeling a Briefcase

BriefCase
Capacity: Integer Weight: Integer
Open() Close() Carry()

- Supponiamo di avere identificato questo oggetto nel mio modello…

A new Use for a Briefcase: Sit On It

Why did we not model it as a chair?

BriefCase
Capacity: Integer Weight: Integer
Open() Close() Carry() SitOnIt()

- Dal modello funzionale deriva SitOnIt(), ed è l'unica applicabile a quell'oggetto…

Correct object identification

Why did we model the thing as “Briefcase”?

What do we do if the SitOnIt() operation is the most frequently used operation?

The briefcase is only used for sitting on it. It is never opened nor closed.
Is it a “Chair”or a “Briefcase”?

E’ **in base all’uso, alla loro interfaccia,** che il tipo degli oggetti del dominio è identificabile correttamente

Partire dalla loro descrizione non funzionale, ovvero per attributi, potrebbe essere fuorviante

la modellazione funzionale dei requisiti è fondamentale

We got abstractions and decomposition

This leads us to **chunks** (classes, objects)

Another way to deal with complexity is to **provide simple relationships** between the chunks

One of the most important relationships is hierarchy (but not the only one)

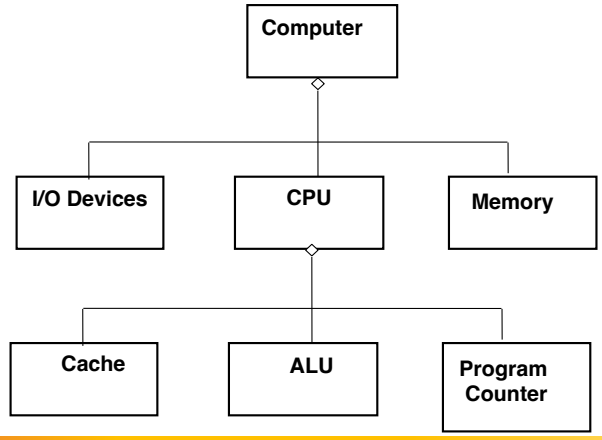
This lead us to talk about architectures….

3. Hierarchy

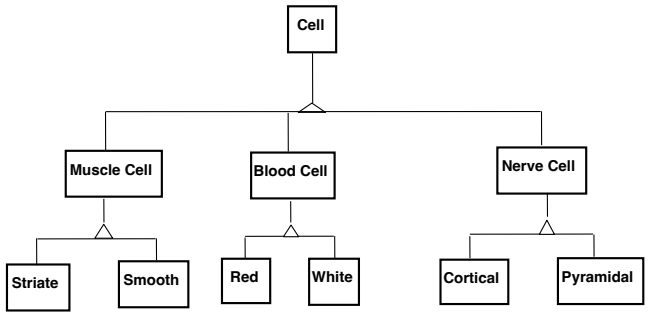
- 3 important hierarchies
  - “Part-of” hierarchy (UML composite structure diagrams)
  - "Is-kind-of" hierarchy (is-a only class diagrams: taxonomy)
  - “Uses” hierarchy (dependences/uses only class diagrams)

Part of Hierarchy

UML Composite structure diagrams



Is-Kind-of Hierarchy (Taxonomy)



Can create separate Class diagrams to report the is-a hierarchy only

Summary: Software Lifecycle Activities