



# **INGEGNERIA DEL SOFTWARE: INTRODUZIONE**

**Ingegneria del Software 2023-2024**

# DOCENTI



## **Filippo Ricca**

*Associate Professor, Ph.D*

Web Page:

<http://www.disi.unige.it/person/RiccaF/>

**E-mail:** [filippo.ricca@unige.it](mailto:filippo.ricca@unige.it)



## **Dario Olianas**

*Postdoc, Ph.D*

**E-mail:**

[dario.olianas@dibris.unige.it](mailto:dario.olianas@dibris.unige.it)



## **Maurizio Leotta**

*Assistant Professor, Ph.D*

Web Page:

<http://www.disi.unige.it/person/LeottaM/>

**E-mail:** [maurizio.leotta@unige.it](mailto:maurizio.leotta@unige.it)

# META-CONSIDERAZIONI SU FDIS

- Insegnamento meno tecnico rispetto a IP, ASD, LPO
- Insegnamento considerato ‘difficile dagli studenti’
  - Serve **abilità di astrazione** e **problem solving**
  - Non si ragiona ‘solo’ a livello di codice
  - Molto materiale
- **Corso pratico**
  - Corso “**by example**”: tanti esempi ....
  - Tool demo, esercizi, laboratori, ...
- Corso il più **autocontenuto possibile** ma la materia è troppo vasta
  - l’obiettivo non è insegnarvi IS ma:
    - cercare di farvi apprezzare la materia e fornire concetti utili per il mondo del lavoro
  - Approfondimenti: nei corsi della magistrale, curriculum: ‘**Software Security & Engineering**’

# SOFTWARE SECURITY & ENGINEERING

## Software engineering track

Software System Design and Modelling  
IT Project Management  
Functional and Security Testing Techniques  
Mobile Development  
Internet of Things

...

*Corsi*

<https://corsi.unige.it/en/corsi/10852>



## Tipiche figure professionali in uscita

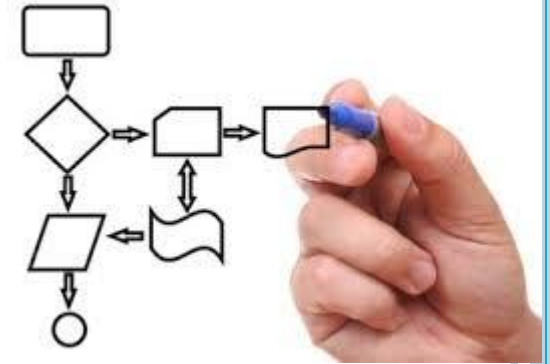
- Senior software/Web/Mobile developer
- Software Engineer
- Software Architect
- Technical leader
- IT consultant
- Information Security Officer
- Data Protection Officer (DPO)
- Cybersecurity consultant





# INTRODUZIONE ALL'INGEGNERIA DEL SOFTWARE

# AGENDA (LEZIONE DI OGGI)



- Sviluppare il software è complesso
- **Crisi del Software** ed esempi di fallimenti clamorosi
- Un po di storia ...
- Che cosa è l'Ingegneria del software?
  - Importanza di questa disciplina
- Processo di sviluppo
  - Le varie fasi ...
- Figure professionali coinvolte

# DATO DI FATTO

- Sviluppare un **sistema software** di **buona qualità** è estremamente complesso
  - programmazione in grande (**programming-in-the-large**)
    - Molte persone, periodo > 6 mesi, molti programmi (milioni di LOCs)
- Vale per tutti i sistemi ma noi ci concentreremo:
  - **+ Sistemi informativi/gestionali**
    - **Aziende:** Contabilità (Incassi e Pagamenti), ordini cliente, fatture, gestione magazzino, ...
    - **Pubblica amministrazione:** Gestione bilancio, gestione tributi, acquedotto
    - **Altro:** Software bancari, software per la gestione di biblioteche, ....
  - **– Real time systems e embedded systems**
    - Soggetti **a constraint di tempo** (performance)
      - Sistemi di guida (razzi, missili, auto)
      - Sistemi di controllo di automazione industriale
      - Software per apparecchiature mediche
      - Software per elettrodomestici
      - ...

# CHI LO DICE?

- Tutti i libri di **Ingegneria del Software**
- Alcuni (tristemente) famosi ‘disastri’ software
- Rapporti dello **Standish Group**
  - IT research and consulting company located in West Yarmouth, Mass. They are most famous for their research on **why IT projects fail ...**
- Personale IT
  - Bill Gates: “*we have to build the first skyscraper every time*”

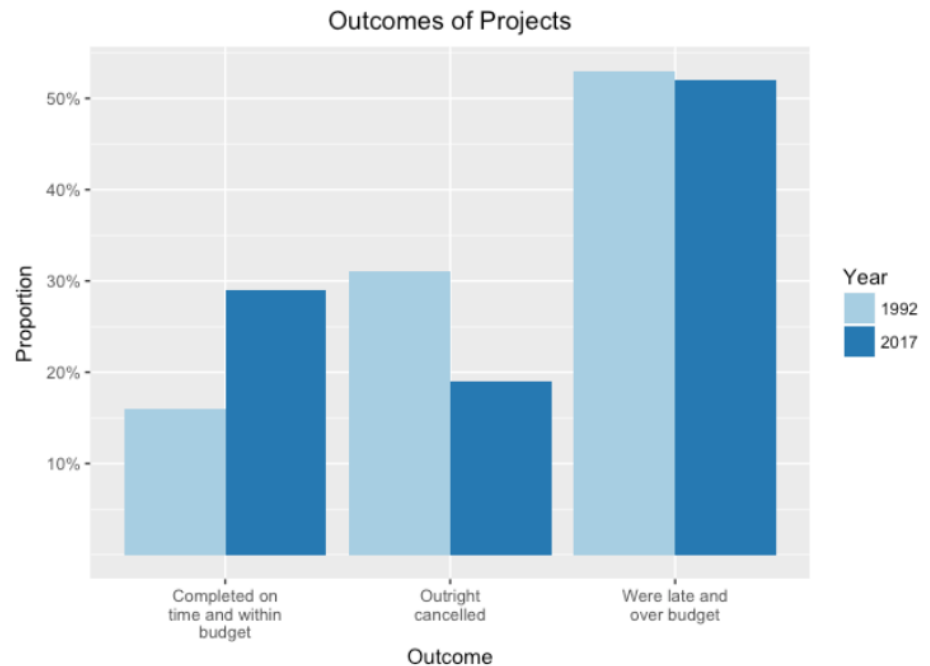




# STANDISH GROUP

○ Survey of 50,000 projects conducted in **2021** around the world revealed:

- **31 percent** of IT projects were successful, coming in on time and on budget
- **19 percent** failed completely, canceled prior to completion or delivered but never used
- **50 percent** were “challenged,” arriving over budget, late (over time), or with less-than-required features



Challenged = “che ha un handicap”

# PROBLEMI CHE SI RISCONTRANO?

- Sfondamento dei costi previsti
  - **Over budget**
- Ritardi nella consegna
  - **Over time**
- Software consegnato con funzionalità parziali
- Software di “bassa qualità” →
- Progetti cancellati
- Insoddisfazione dei clienti
  - Il software **non corrisponde** a quanto richiesto ...



**BUG**

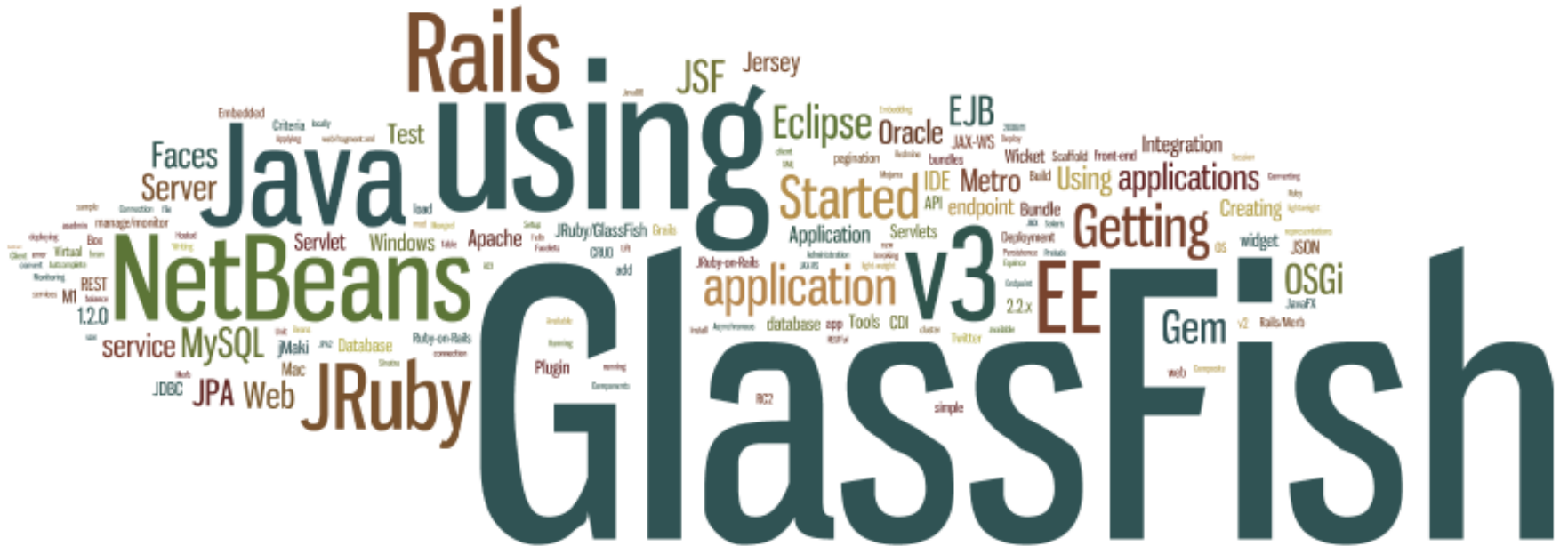


# PERCHÈ?

## Non tanto:

- complessità intrinseca del problema
- problemi tecnologici
  - Es. scelta di un linguaggio, Application server, IDE, Framework o DBMS ...

Anche se spesso orientarsi è spesso un “delirio” ...



MA ....



- **problemi di comunicazione** tra gli “interessati” ed esperti IT
  - **Stakeholders** (“portatore di interesse”)
    - Es. Utenti, tecnici, commerciali, managers, ...
- difficoltà a comprendere i bisogni del cliente
- problemi di gestione (management) e organizzativi
  - Pianificazione, allocazione risorse, gestione del personale, ...
- difficoltà a lavorare in gruppo
- difficoltà a comprendere il “**dominio del business**”
  - **Dominio del business** = contesto in cui il software opera
    - Es. Il concetto di *Ordine* o di *Fattura* e la loro reciproca relazione in un sistema di fatturazione

# COSA ACCADE IN MOLTI PROGETTI REALI

Difficoltà di comunicazione



How the customer explained it



How the Project Leader understood it



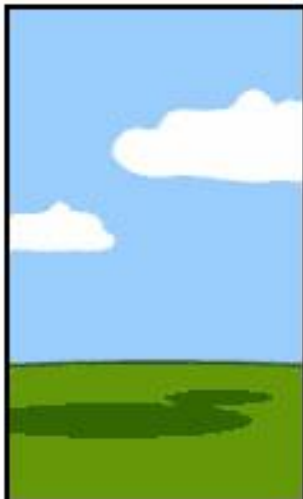
How the Analyst designed it



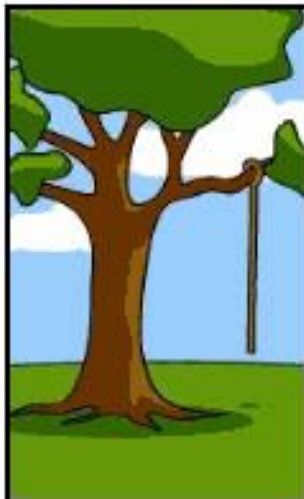
How the Programmer wrote it



How the Business Consultant described it



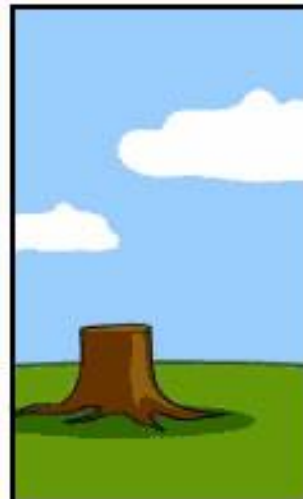
How the project was documented



What operations installed



How the customer was billed

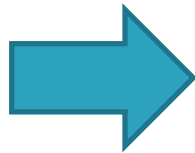


How it was supported



What the customer really needed

# ARIANE 5 (RAZZO NO ASTRONAVE CON UOMIONI A BORDO)



37 secondi dopo il lancio

04/06/1996 – Alla partenza (viaggio inaugurale)



# ARIANE 5 (RAZZO NO ASTRONAVE CON UOMIONI A BORDO)



## Esplosione di Ariane 5

- Il sistema di guida del razzo si spegne in volo a causa del tentativo di memorizzare un floating point a 64-bit in un intero a 16-bit (**arithmetic overflow**)
- Il razzo esplode nel suo viaggio inaugurale (10 anni di lavoro - 7 miliardi di \$) [NY Times Magazine]



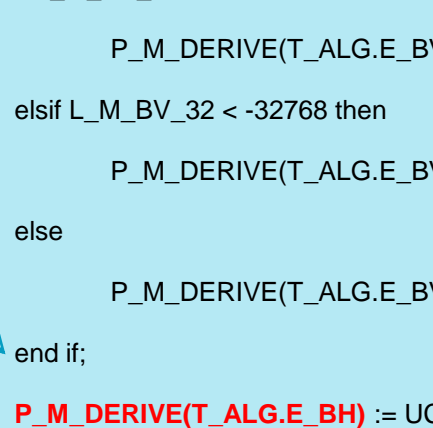
37 secondi dopo il lancio

04/06/1996 – Alla partenza (viaggio inaugurale)

# CAUSA DIRETTA DEL DISASTRO

## ADA CODE

```
L_M_BV_32 := TBD.T_ENTIER_32S ((1.0/C_M_LSB_BV) * G_M_INFO_DERIVE(T_ALG.E_BV));  
if L_M_BV_32 > 32767 then  
    P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;  
elsif L_M_BV_32 < -32768 then  
    P_M_DERIVE(T_ALG.E_BV) := 16#8000#;  
else  
    P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));  
end if;  
  
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M_LSB_BH) * G_M_INFO_DERIVE(T_ALG.E_BH)));
```



Variabile intera a cui è stato assegnato un valore troppo grande ...

Per motivi di efficienza era stato tolto “if” per fare il controllo che il numero non fosse troppo grande (come invece c’è nelle righe sopra)



# CAUSA DIRETTA DEL DISASTRO

## ADA CODE

```
L_M_BV_32 := TBD.T_ENTIER_32S ((1.0/C_M_LSB_BV) * G_M_INFO_DERIVE(T_ALG.E_BV));
```

```
if L_M_BV_32 > 32767 then
```

Sebbene la causa diretta è stato un **software bug**  
il motivo di questo disastro è attribuibile ad una  
cattiva gestione del progetto ...

```
P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
```

```
end if;
```

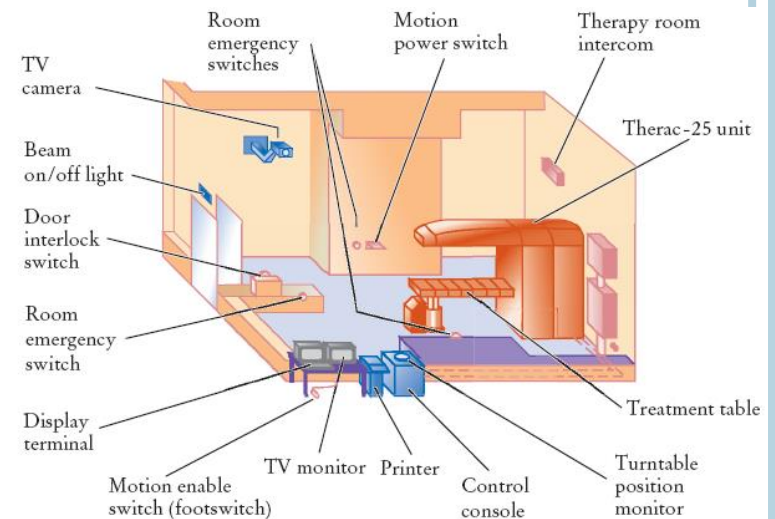
```
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M_LSB_BH) * G_M_INFO_DERIVE(T_ALG.E_BH)));
```

Variabile intera a cui è stato assegnato un valore troppo grande ...

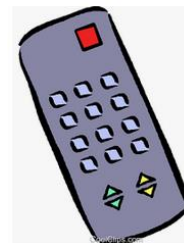
Per motivi di efficienza era stato tolto “if” per fare il controllo che il numero non fosse troppo grande (come invece c’è nelle righe sopra)

# THERAC-25

- **Overdose di radiazioni**
  - 1985-1987
- Peggior incidente in 35 anni di storia sulla **radioterapia**
- **Sei incidenti**, durante i quali venne somministrata una dose di radiazione 100 volte superiori al normale (**due morti**)
  - *IEEE Computer*, Vol. 26, No. 7, July 1993, pp. 18-41
- Therac-25 doveva fornire 2 trattamenti:
  - Terapia a base di fasci di elettroni
  - Terapia a raggi X (**con filtro**)
- Con una combinazione di tasti del sistema di controllo, la macchina, per errore, somministrava una **Terapia a raggi X senza filtro ...**



**Figure 9** Typical Therac-25 Facility



# THERAC-25

- Overdose di radiazioni
  - 1985-1987

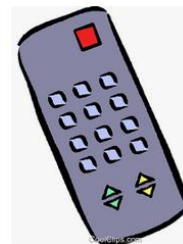
**Software scritto e sviluppato male ....**

• Mancata implementazione di una **constraint** nel sistema, la funzionalità "Terapia a raggi X senza filtro" non doveva essere permessa

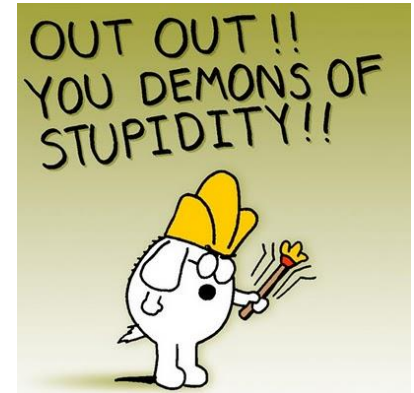
**No software testing**

• Non sono stati eseguiti dei test operativi del Therac-25 e del relativo programma prima che i vari esemplari venissero installati negli ospedali

- Con una combinazione di tasti del sistema di controllo, la macchina, per errore, somministrava una **Terapia a raggi X senza filtro ...**



# UN LUNGO TUNNEL DELL'ORRORE



- <http://www.cs.tau.ac.il/~nachumd/horror.html>
- Alcune storie divertenti...
  - ... when they shut down the **IBM 7094** at **MIT** in 1973, they found a **low-priority process** that had been submitted in 1967 and had not yet been run. [Silbershatz and Galvin, pp. 142-143].
- ... e molti episodi realmente catastrofici
  - The Korean Airlines KAL 901 accident in **Guam** killed 225 out of 254 aboard. A worldwide bug was discovered in barometric altimetry in **Ground Proximity Warning System (GPWS)**. [ACM SIGSOFT Software Engineering Notes, vol. 23, no. 1.]

GPWS = segnala possibili situazioni di pericolo per l'aeromobile quando si avvicina al suolo

# INGEGNERIA DEL SOFTWARE (UN PO' DI STORIA)

< 1965 – **Le origini.** Il software viene prodotto in modo artigianale. Software usato in ambito militare e progetti scientifici

Algoritmi +  
Strutture dati +  
Linguaggi (Algol 58/60) +  
Programmazione strutturata

1965 a 1985 – Sviluppo software per più utenti. Primi problemi

**La crisi del software.** bassa qualità e bassa produttività; la buona programmazione non basta!

1968 – **conferenza NATO.** Viene riconosciuto il problema e viene coniato il termine “Software Engineering”. **Nasce la disciplina ...**

1985 a 1990 – **Utilizzo massivo del software (industria + attività commerciali)**

**No Silver Bullet (1986).** Fred Brooks: nessuna tecnologia e nessuna metodologia/metodo migliorerà lo stato delle cose nei prossimi 10 anni!

1991 a 1999 – **Web engineering.** La nascita del Web sconvolge tutto e tutti ... Esplosione dell'utilizzo del Web

Component-based software engineering

Fallimento dei metodi formali: (per sw non safety critical)

**2000 ad adesso** – Metodi agili, Model Driven Development, nuovi linguaggi/framework/IDE, Microservizi, Continuous Integration, nuovo processo di sviluppo: **DevOps**

Crisi del software

# CONFERENZA NATO (1968)

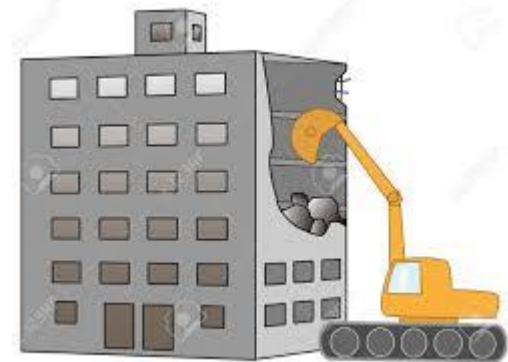


# CHE COSA È L'INGEGNERIA DEL SOFTWARE?

- L'ingegneria del software è una **disciplina** ingegneristica che si occupa di tutti gli aspetti relativi allo **sviluppo del software**
  - disciplina = materia/soggetto di studio di corsi
- L'ingegneria del software è un insieme di teorie, metodi, tecniche e strumenti (tool) per sviluppare software di qualità in maniera professionale

- **Idea:** considerare il SW come un prodotto 'qualsiasi' ed usare per il SW lo stesso approccio degli altri settori dell'ingegneria

  - Civile, industriale, navale, ...



***Costruire un software è come costruire un edificio/nave!***



# PROCESSO DI SVILUPPO

- **Obiettivo** dell'Ingegneria del software:
  - definire **metodi, tecniche, tool e procedure** per ottenere sistemi software di **grandi dimensioni, di alta qualità, a basso costo, ed in breve tempo**
- Per produrre 'SW di qualità' occorre puntare sulla qualità del ***processo di sviluppo*** del software
  - Come per le altre industrie manifatturiere
    - il software come prodotto 'qualsiasi'
    - Qualità del processo di sviluppo => Qualità del prodotto





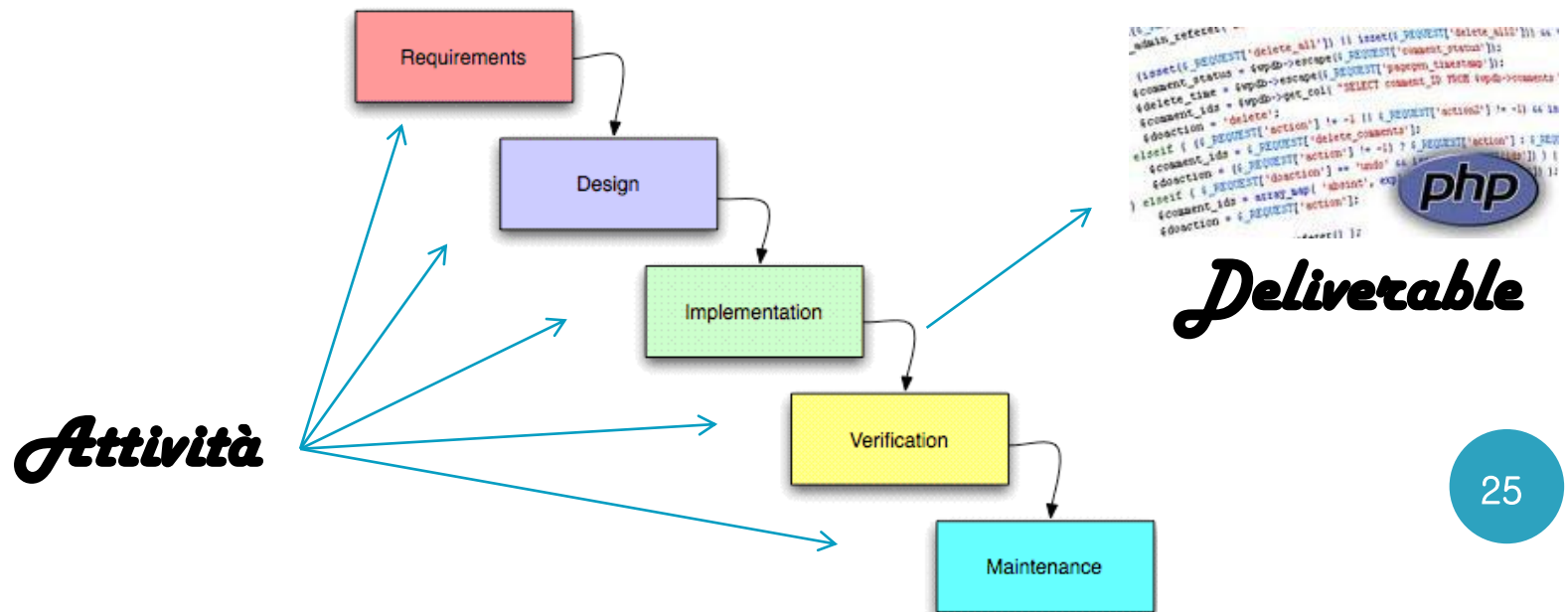
# MODELLI DEL PROCESSO SOFTWARE

- **Processo = sequenza di attività/fasi da seguire**

- *Deliverable* = prodotto di un attività (artefatto)
- I deliverable prodotti in una fase sono input alla fase successiva

- Diverse tipologie di modelli (li vediamo prox lezione):

- ad esempio: **Waterfall (a cascata)**



# TRE MACRO-CATEGORIE 'DI MODELLI DI PROCESSO'

+ libertà

## ○ Metodi agili:

- + software funzionante e — documentazione
- Customer “on-site”
- + importanti i programmatori che strumenti/processo
- Risposta veloce ai cambiamenti

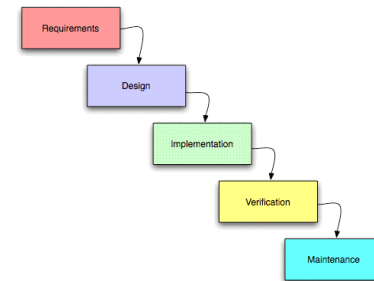
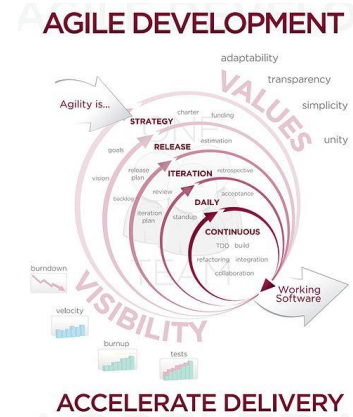
## ○ Plan-driven:

- + processo
- + documentazione
  - requisiti e design
- codice deriva dal design

## ○ Metodi formali:

- “quasi dimostrazioni matematiche”
  - Definire precisamente i requisiti
  - Provare la correttezza dell'implementazione
  - Provare che alcune proprietà del sistema sono vere
    - No deadlock!

+ rigore e  
disciplina



```

$$\forall (oz, uml) : project \bullet$$

$$\{c : oz \cap Classdef \bullet c.name\} = \{c : uml.classes$$

$$\bullet c.name\} \bullet \forall c_1, c_2 : oz \cap Classdef \bullet \exists_1 c' :$$

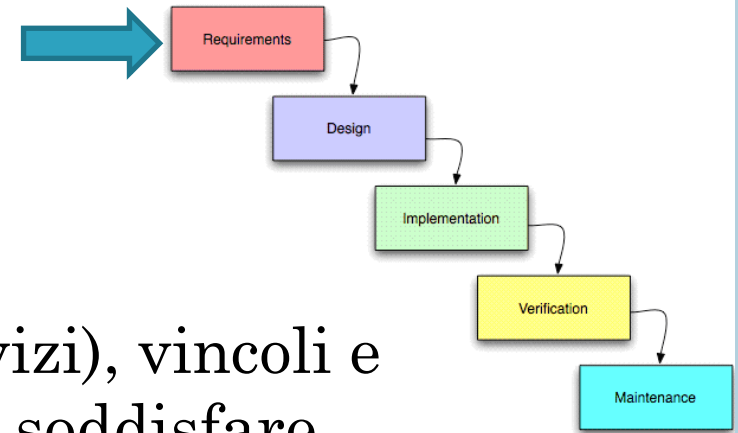
$$uml.classes \bullet c'.name = c_1.name$$

$$c'.attris = \{cls : Classdef \mid cls \in oz \bullet cls.name\}$$

$$\triangleleft c_1.state.decpart$$

```

# RACCOLTA, ANALISI E SPECIFICA DEI REQUISITI



- Definizione di funzionalità (servizi), vincoli e prestazioni che il sistema dovrà soddisfare
  - **Raccolta:** tramite 'interviste' con il committente/cliente
  - definizione di che cosa deve essere fatto e non come!
  - **Fase molto importante: fare degli errori qui pregiudica la buona riuscita del progetto!**
- Deliverable: Requirement Specification/Definition
  - **definition** [+ informale]
    - Linguaggio naturale / Use cases (li vedremo in seguito)
  - **specification** [+ rigorosa]
    - Linguaggi di specifica formali
      - *es. Linguaggio Z*

# VARI MODI DI RAPPRESENTARE I REQUISITI

## ○ Informale (Linguaggio naturale):

- The value of x will be between 1 and 5, until some point where it will become 7. In any case it will never be negative

## ○ Formale:

- $(1 \leq x \leq 5 \cup x = 7) \wedge (\square x \geq 0)$

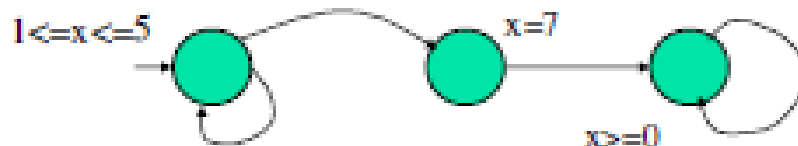
Finchè ad un certo punto

sempre

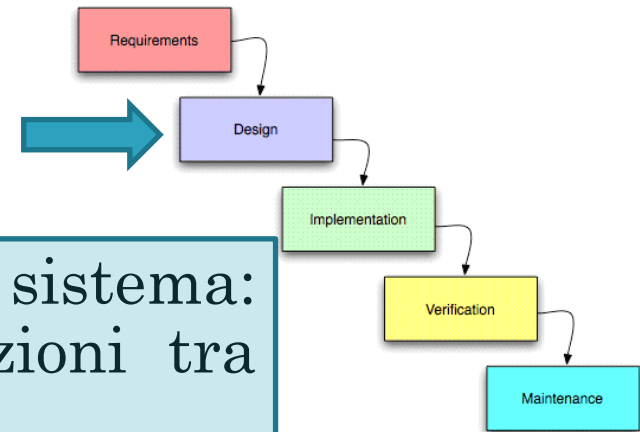
*Linguaggio 'Matematico'*

## ○ Visuale:

NON VEDREMO NOTAZIONI FORMALI TIPO QUESTA



# FASE DI DESIGN: “COME”

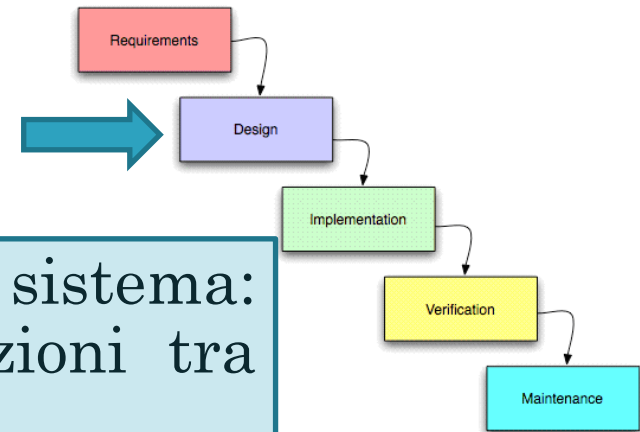


- Definizione dell'architettura del sistema: **componenti del sistema** e relazioni tra questi
  - progetto di alto livello (**High Level Design**)
- Definizione della struttura interna di ciascun componente
  - progetto di dettaglio (**Low Level Design**)

- “Astratto il più possibile”: non riguarda aspetti implementativi di dettaglio (**platform independent design**)
  - No piattaforma SW, No linguaggio, No algoritmi
- Deliverable: Design specification (HLD e/o LLD)
  - **Linguaggio UML** (lo vedremo in seguito) *Standard de-facto*

*In analogia con la costruzione di una casa il design è “l’elaborato dell’architetto”*

# FASE DI DESIGN: “COME”



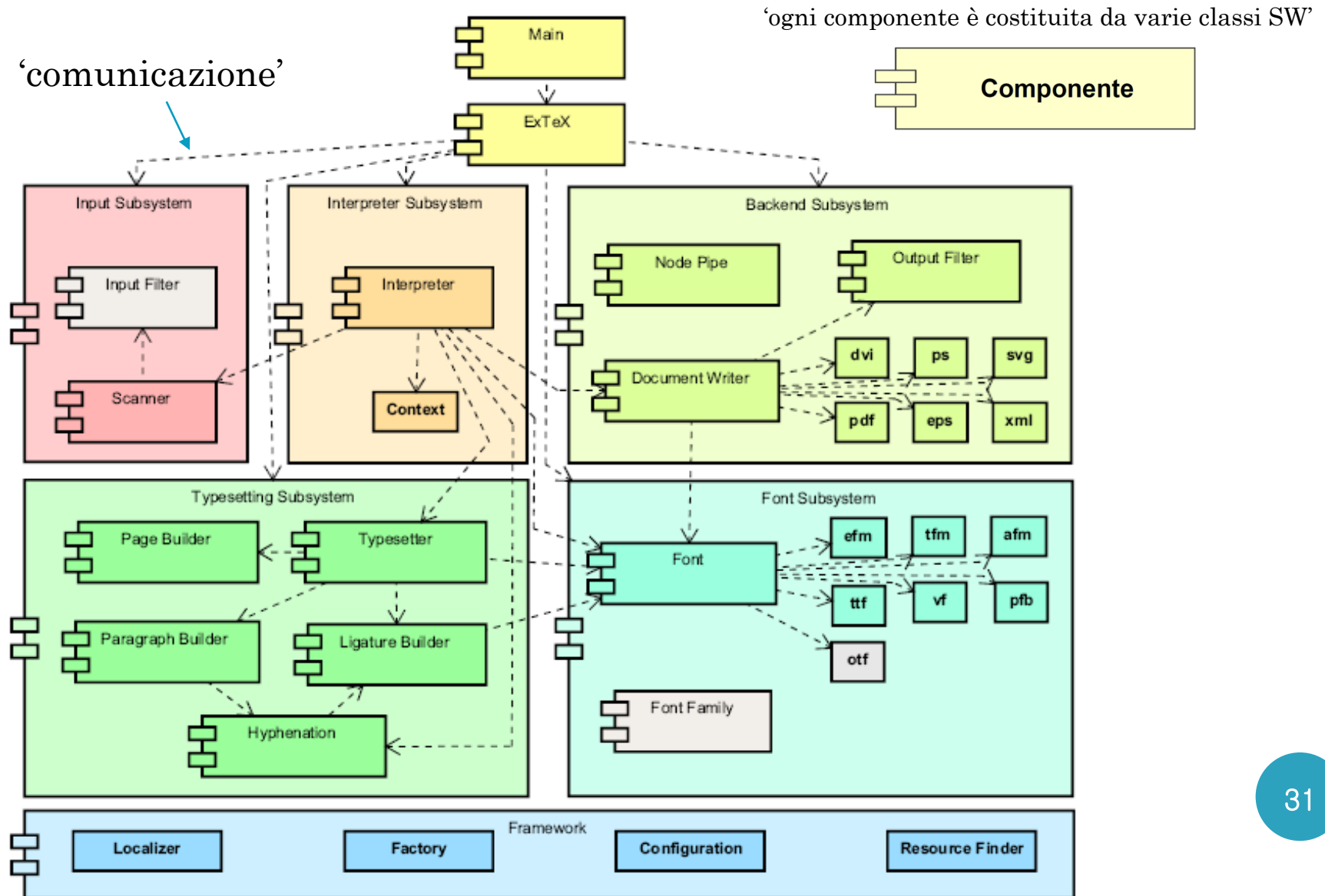
- Definizione dell'architettura del sistema: **componenti del sistema** e relazioni tra questi
  - progetto di alto livello (**High Level Design**)
- Definizione della struttura interna di ciascun componente
  - progetto di dettaglio (**Low Level Design**)

• **Documentazione** = i prodotti dello sviluppo di un sistema software. Documento dei requisiti + HLD + LLD + Test Plan + ...

- No piattaforma SW, No linguaggio, No algoritmi
- Deliverable: Design specification (HLD e/o LLD)
  - **Linguaggio UML** (lo vedremo in seguito) *Standard de-facto*

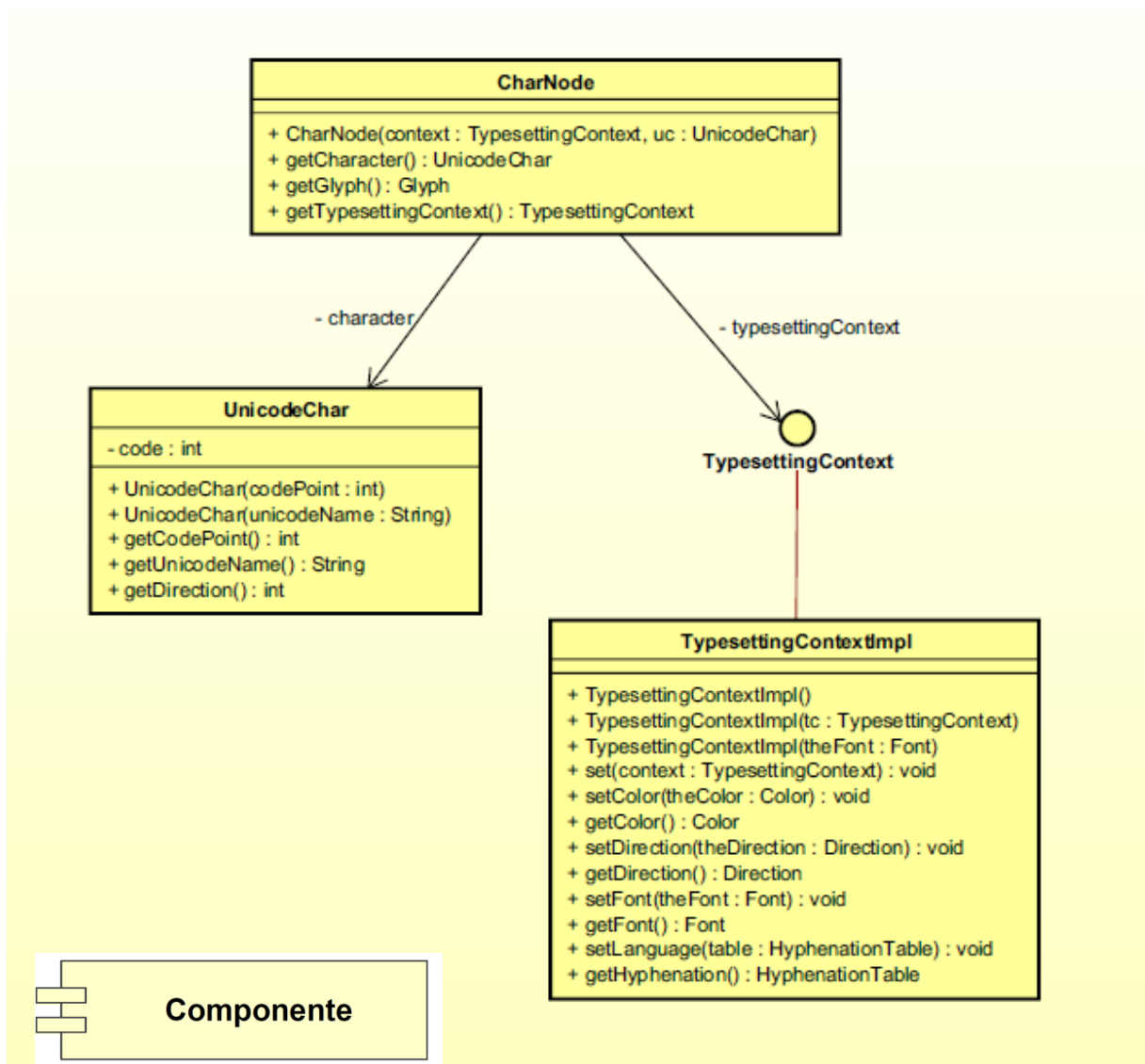
*In analogia con la costruzione di una casa il design è “l’elaborato dell’architetto”*

# ExTeX (TYPESETTING SYSTEM): HLD



*livello di granularità più fine: siamo a livello di classi ...*

## EXTEX: PORZIONE DI LLD





# IMPLEMENTAZIONE/CODIFICA

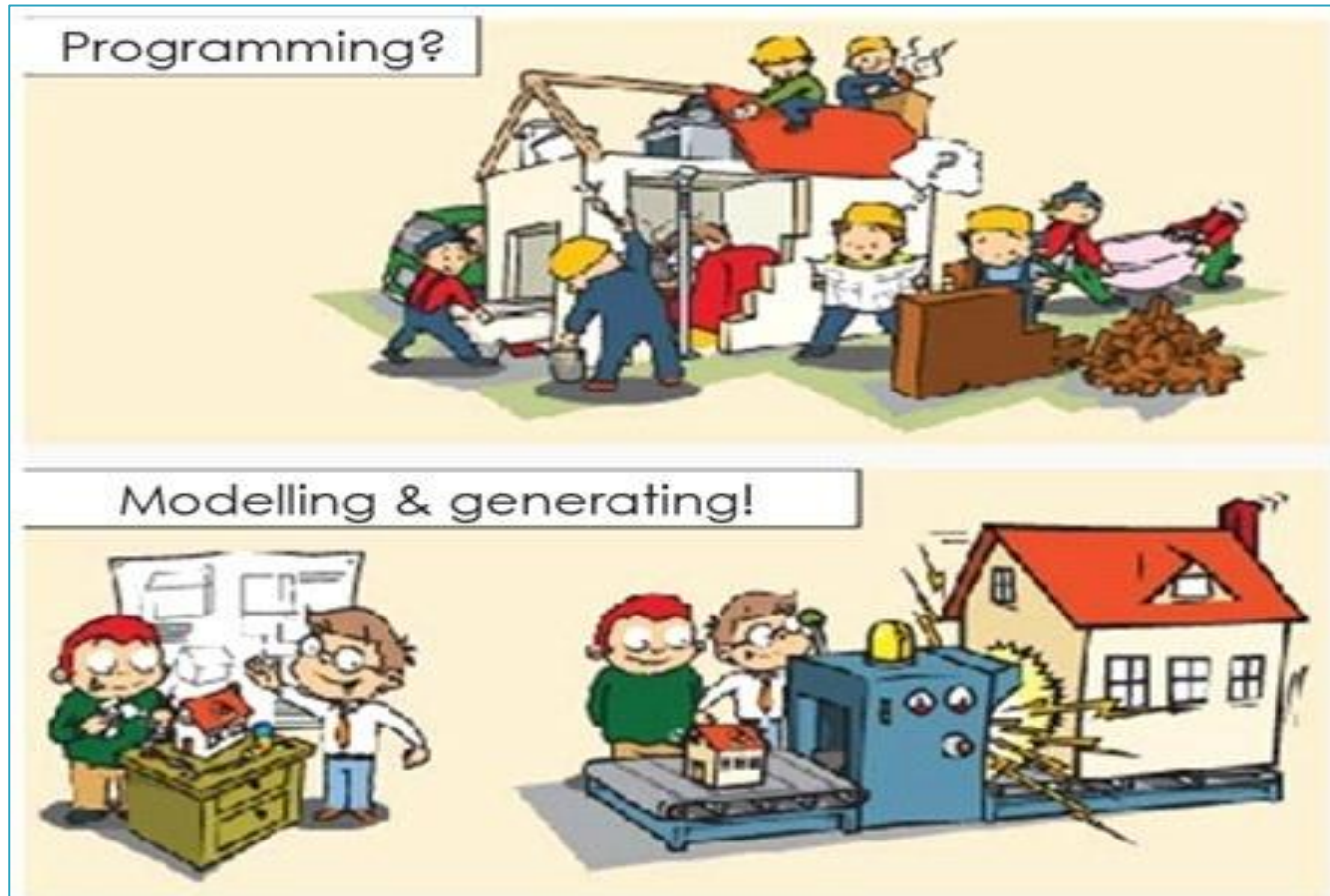


- Codifica in un linguaggio di programmazione
- **Se design molto dettagliato/preciso**
  - Fase “meccanica” e poco creativa ...
    - Addirittura nel **Model Driven Development (MDD)** questa fase viene abolita
      - Si esegue direttamente il modello o il codice viene generato in modo automatico
- **Altrimenti attività creativa**
  - Vedere “metodi agili”
- Deliverable: Codice del sistema
  - in genere costituito da diversi moduli/programmi

# MODEL DRIVEN DEVELOPMENT (MDD)

**Idea:** Si esegue direttamente il modello di design oppure il codice viene generato in automatico a partire dal modello

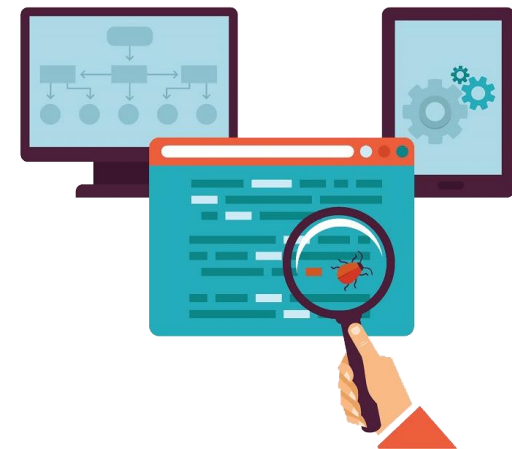
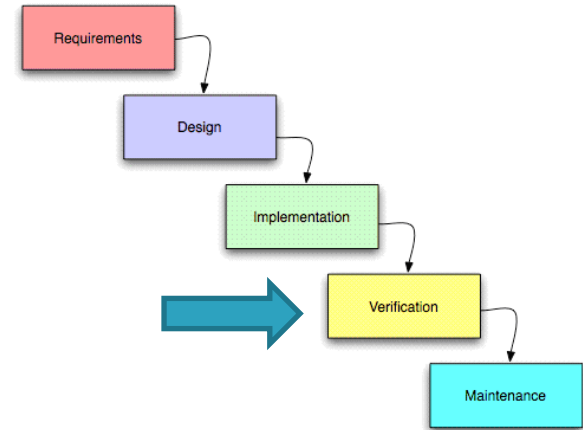
Sviluppo tradizionale



MDD

# TESTING (VERIFICA O COLLAUDO)

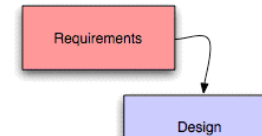
- Testing = procedimento utilizzato per individuare **failure (malfunzionamenti ...)**
- Consiste nell'eseguire il software e nel valutare se il comportamento del software rispetta i requisiti
  - e/o soddisfa i bisogni dell'utente



- Deliverable: Test Plan & Test Report

*Test plan = cosa verrà testato, con quale strategia, con quali input*

# TESTING (VERIFICA O COLLAUDO)



- Testing = procedimento per individuare **(malfunzionamenti ...)**
- Consiste nell'eseguire il nel valutare se il compor software rispetta i requisiti
  - e/o soddisfa i bisogni dell'u

- Deliverable: Test Plan &

*Test plan = cosa verrà testato, con quale strategia, con quali input*

## TEST PLAN For Pocket PC Tetris Game

### Introduction

This test plan is for the purpose of testing the Tetris game designed and developed by Intelligent Design. Intelligent Design plans to test the Tetris game utilizing a Pocket PC 2003 SE Emulator. This emulator will allow the game to be executed in a pocket pc environment. This product will be tested using a Black box strategy. White box testing will be done by the developer as he develops the code. As this product evolves, regression test will be performed for all new changes/additions to the game.

### Personnel

Name	Role	E-mail
Stephen McVey	Project Manager	<a href="mailto:csu17817@mail.claytonstate.net">csu17817@mail.claytonstate.net</a>
Barry Casebeer	Document manager	<a href="mailto:csu12598@mail.claytonstate.net">csu12598@mail.claytonstate.net</a>
David Hutcherson	Developer	<a href="mailto:csu20987@mail.claytonstate.net">csu20987@mail.claytonstate.net</a>
Rhonda Dillon	Tester & Customer Liaison	<a href="mailto:csu16900@mail.claytonstate.net">csu16900@mail.claytonstate.net</a>

### Test Cases

#### 1. Graphical Interface

##### Test Number 1.1

##### **Purpose**

This test case covers requirements 1.1, 1.2, 1.2.1, 1.2.2, 1.2.3, 1.2.4, 1.2.5, 1.2.6, and 1.3. The following tests will assure that the graphical interface coincides with the Pocket PC environment, and that the blocks are easily identified. This test case will assure that the color schemes are appropriate, and that the user can visually comprehend the game.

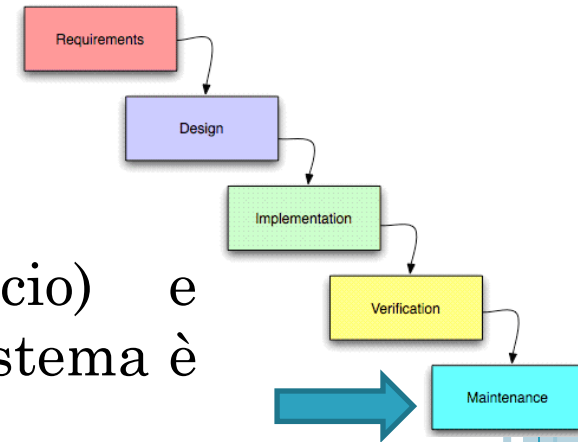
##### **Preconditions**

The game must be loaded on the Pocket PC 2003 SE Emulator.

##### **Post-conditions**

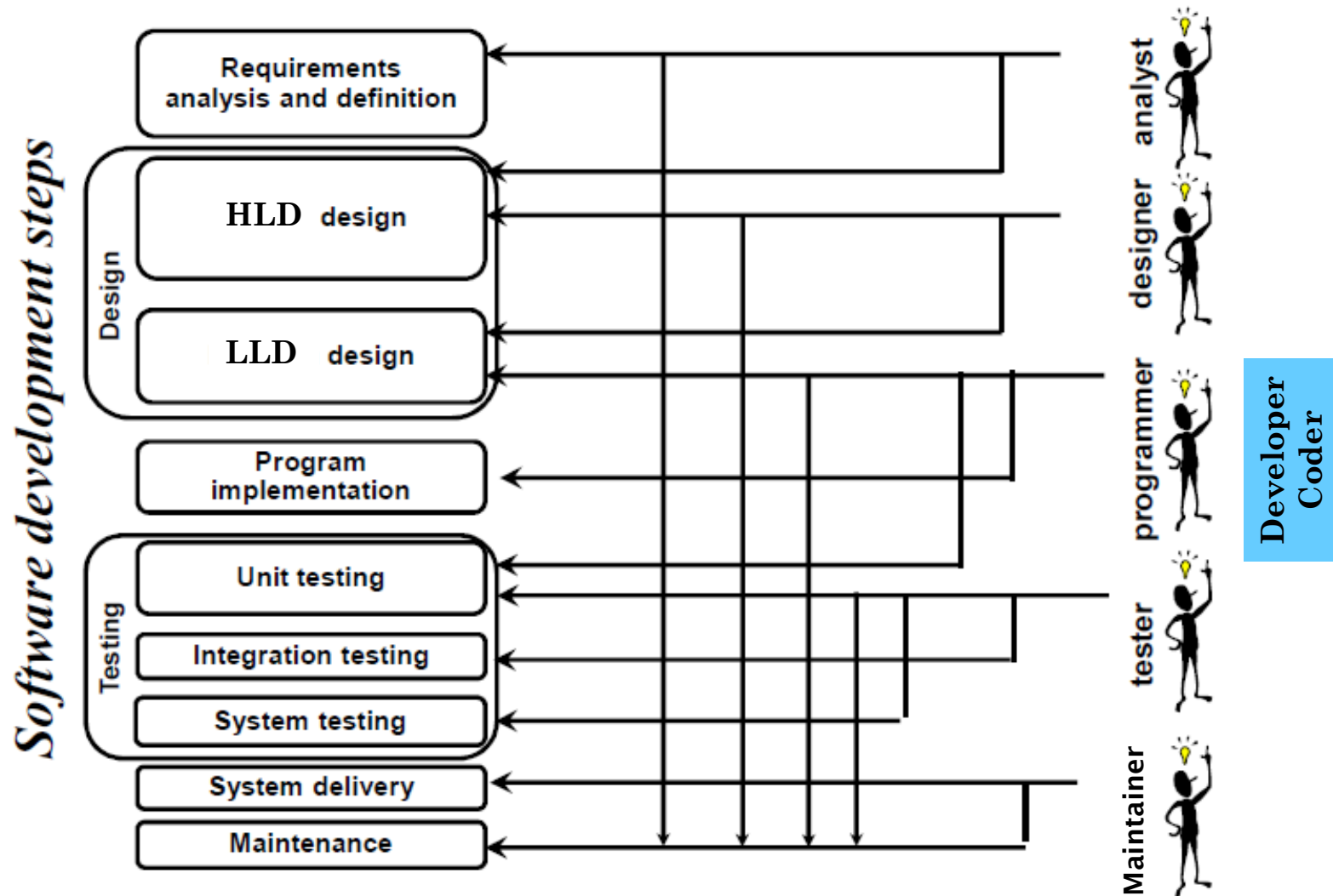
# DELIVERY E MANUTENZIONE

- **System delivery:** consegna (o rilascio) e installazione del sistema, dopo la quale il sistema è operativo
  - **NB:** In queste fase è fondamentale istruire gli utenti all'uso del sistema, utilizzando dei **manuali utente**
- **Manutenzione (evoluzione):** attività per **mantenere operativo** il sistema dopo la consegna all'utente fra cui
  - **rimediare ad errori**
  - **migliorare il sistema**
    - Aggiungere nuove funzionalità o modificarne di esistenti
  - **adattare** (il mondo evolve ...)
    - Es. - Sistema operativo Windows 10 --> Windows 11
    - Hibernate 3.6.8 --> Hibernate 3.7.2



# RUOLI TECNICI DELL'INGEGNERIA DEL SW

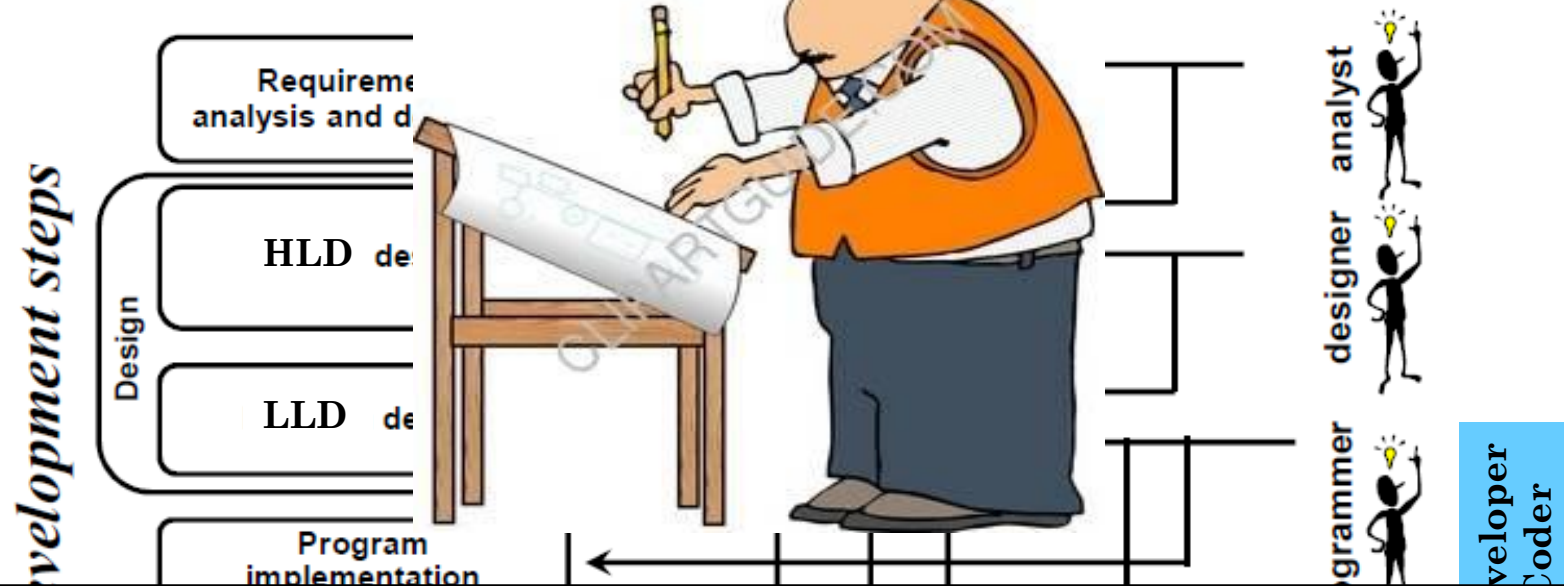
Aziende 'grandi'. Esistono figure specifiche, Aziende piccole 'factotum'



# RUOLI TECNICI DELL'INGEGNERIA DEL SW

Aziende 'grandi'. Es

ende piccole 'factotum'



**Project manager** = responsabile unico della valutazione, pianificazione e controllo di un progetto (**ruolo organizzativo**).

Si occupa di gestire il personale, stimare i costi e far sì che il progetto rispetti i tempi e i costi ....

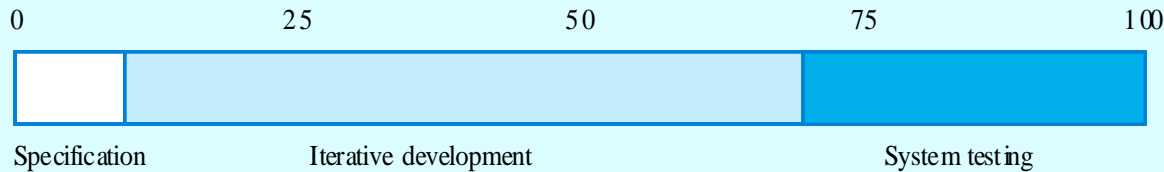
# RIPARTIZIONE DEL COSTO SOFTWARE (AZIENDA)

Modello di Sviluppo

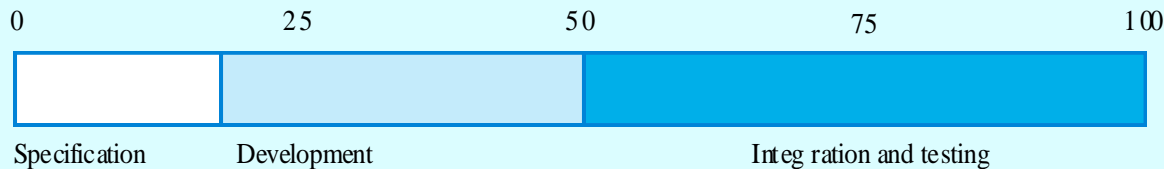
Water fall model



Iterative development



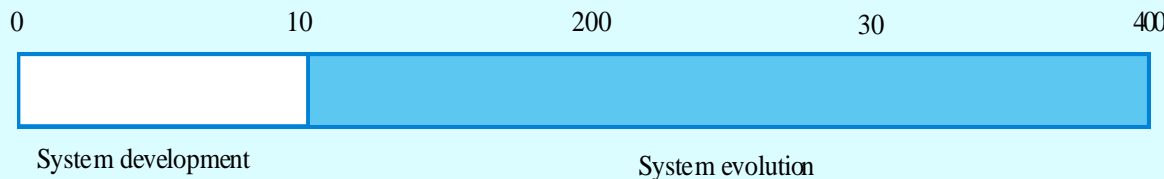
Component-based software engineering



**Testing**  
~ 40-50%

Evoluzione

Development and evolution costs for long-lifetime systems



**3 a 1**

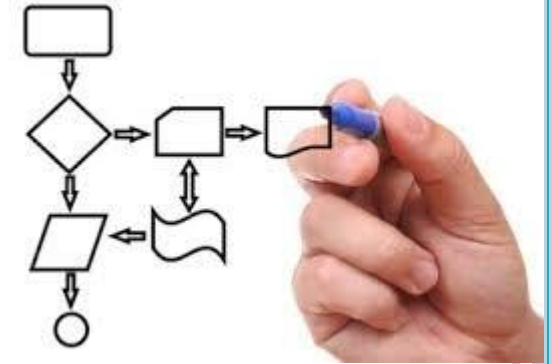
40





# INTRODUZIONE AL CORSO

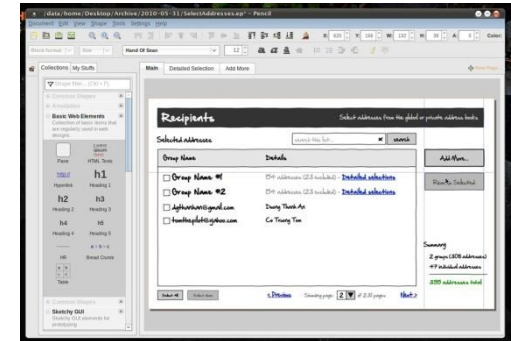
# AGENDA (LEZIONE DI OGGI)



- Cosa vedremo quest'anno?
  - Teoria
  - Laboratorio (Tool)
- Riferimenti bibliografici
- Questioni burocratiche “noiose”
  - Formato esame, AulaWeb, Crediti, etc.

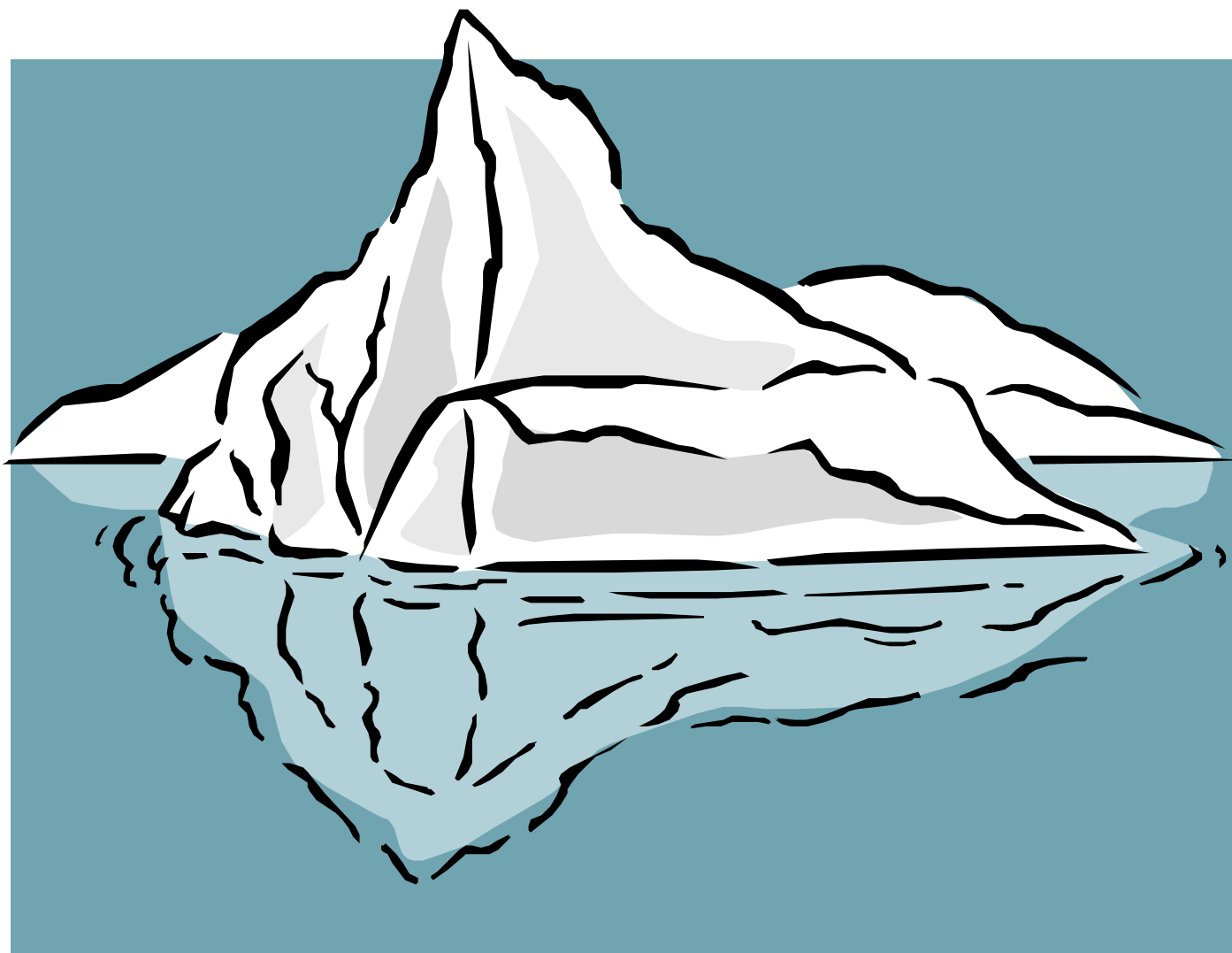
# COSA VEDREMO IN QUESTO CORSO?

- Il processo software: **Modelli di processo di sviluppo**
- Analisi e progettazione
  - Requisiti
    - Use cases (+ screen mockups)
  - Design (principi)
    - HLD (Architettura) e LLD
- Progettazione **orientata agli oggetti**
  - Linguaggio UML (alcuni diagrammi)
  - Cenni di Model Driven Development (MDD)
  - Pattern architetturali e di progettazione (cenni)
    - Design pattern = “soluzione a problema ricorrente”
  - Refactoring (cenni)
- Codifica
  - Da UML a Java
  - Cenni di Persistenza oggetti e interfacciamento DB (Prof.ssa Guerrini)?
- Software Testing
- Metodi agili (solo Extreme Programming)
- Cenni di Manutenzione/evoluzione

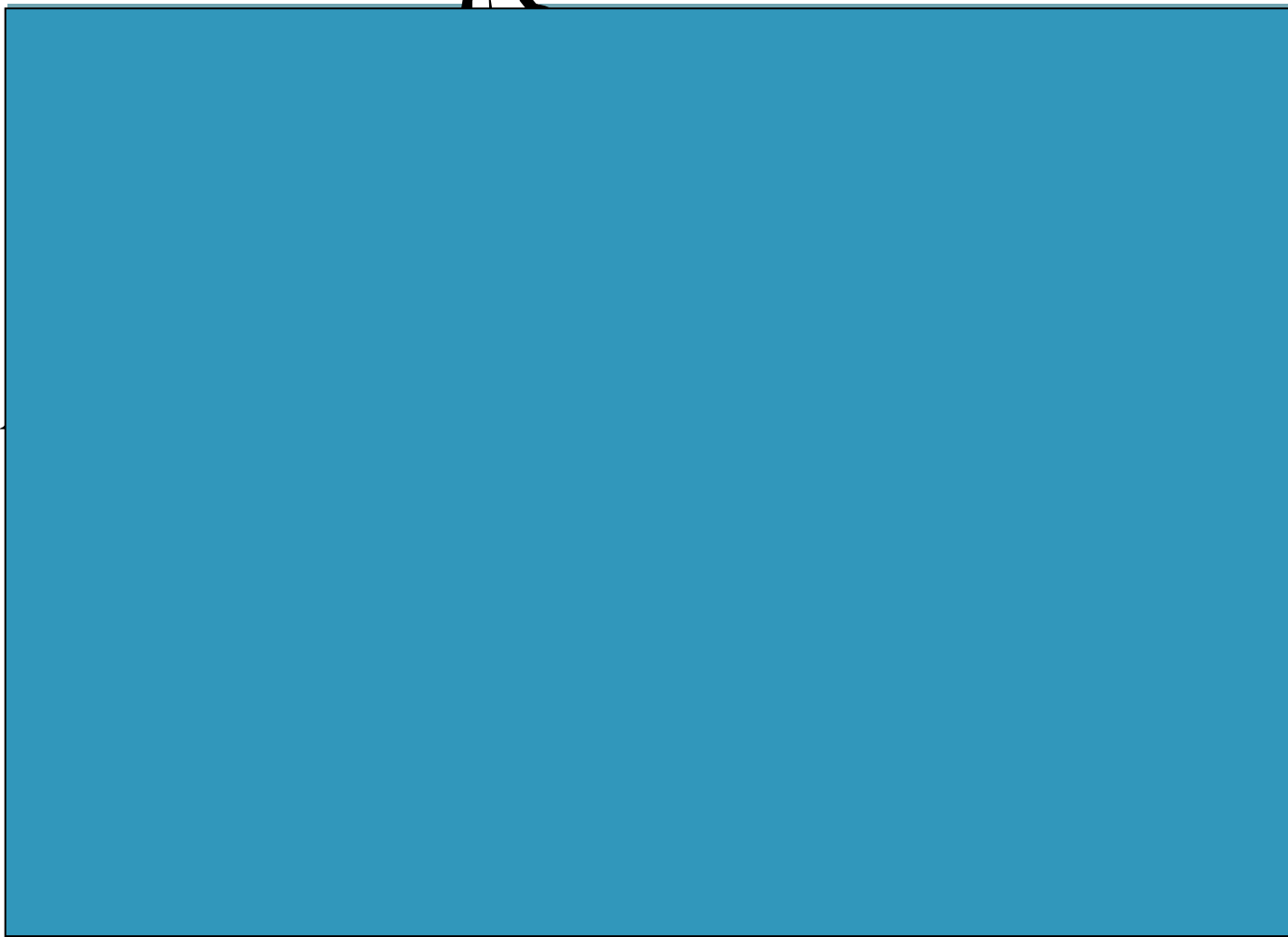


**6 crediti**

OVVERO SOLO ...



# O MEGLIO SOLO ...



# TOOL DEMO & LABO

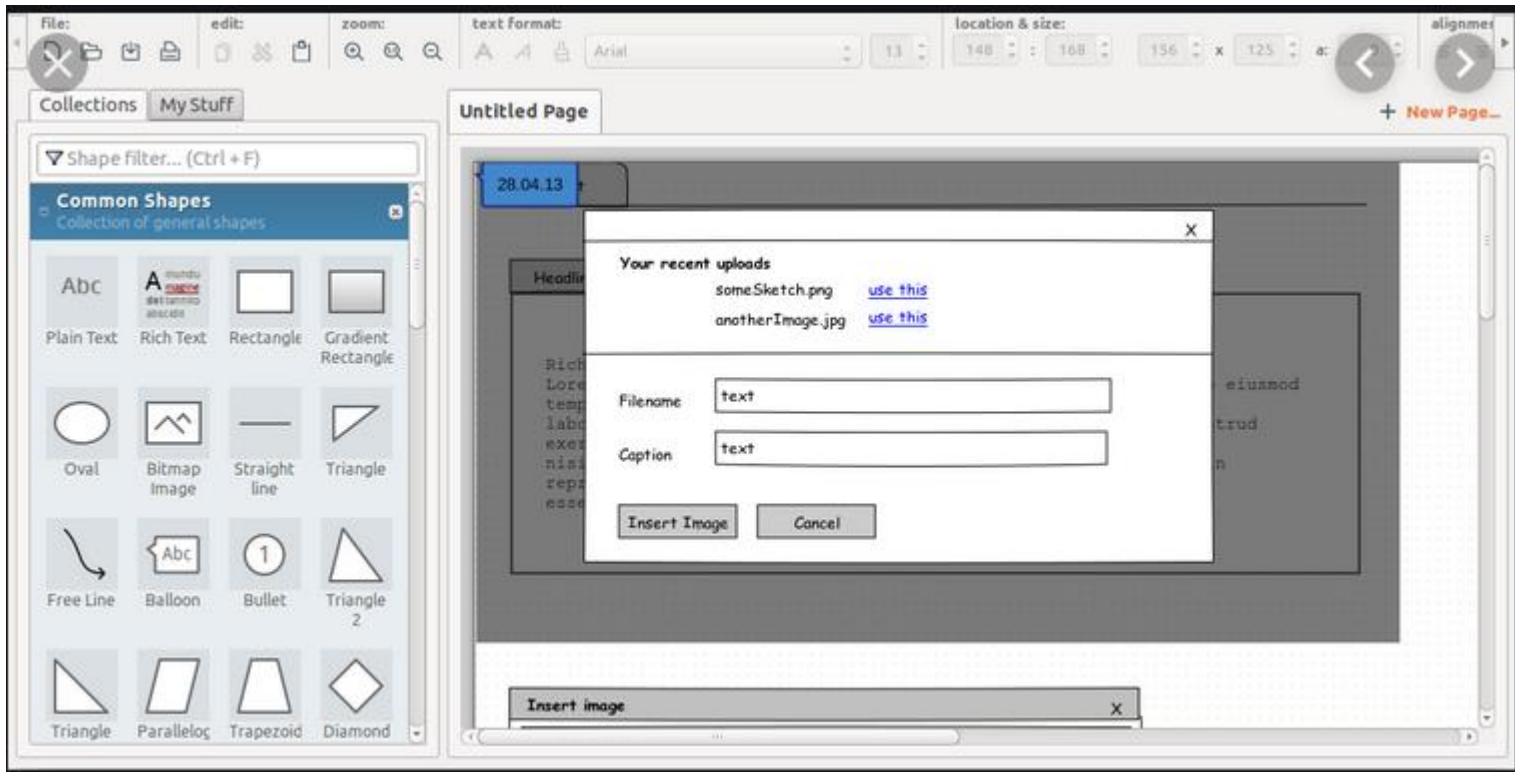
- Oltre alle lezioni teoriche e esercizi sono in programma:
  - 4/5 lezioni + orientate all'utilizzo di tool (Demo/Lab/esperimenti)
    - Tool (sia prototipi che SW professionale):
      - **Pencil**
      - **Stan 4J**
      - **Visual Paradigm for UML**
      - **WebRatio**
- 1/2 seminari tenuti da esperti (o presentazione aziende)

*Ad esempio ...*

# PENCIL

○ Per produrre screen mockups ....

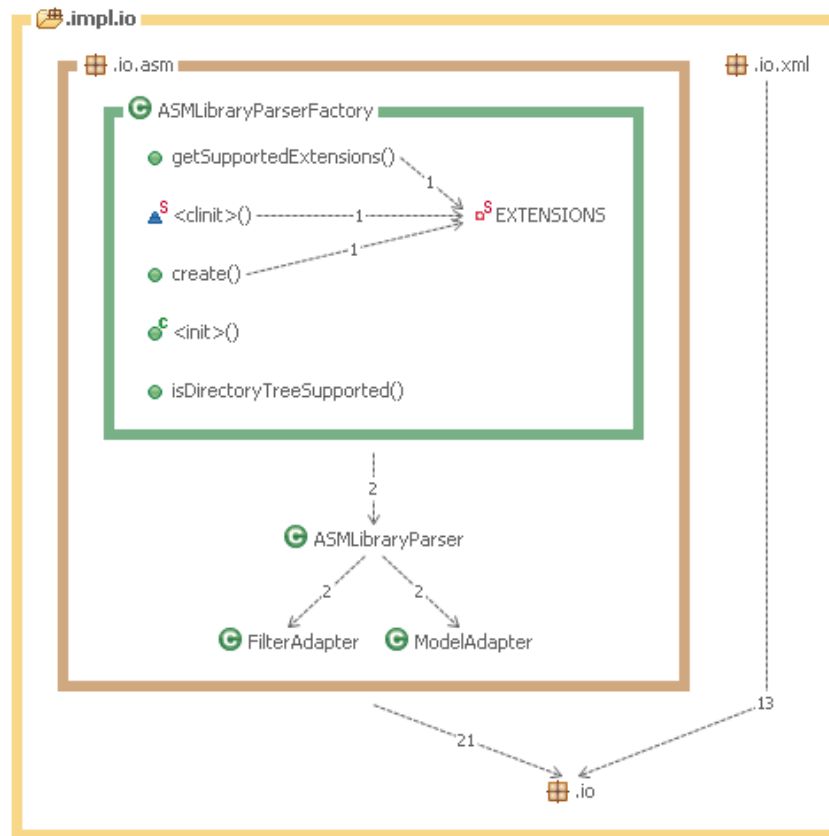
- Liste
- menu
- checkbox
- bottoni
- ...



# STAN4J

## Analisi dell'architettura di un sistema Java

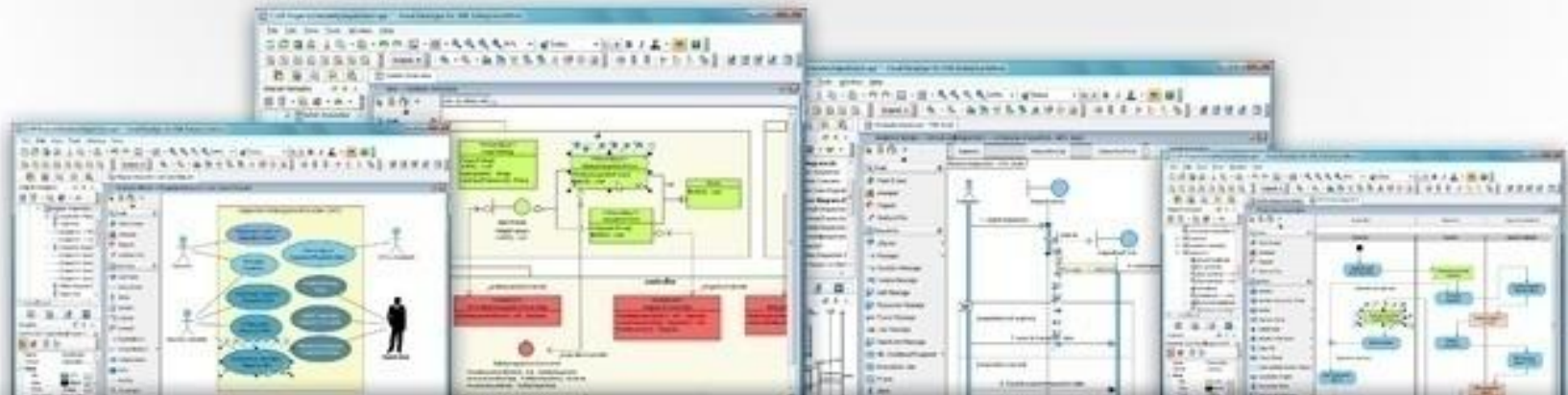
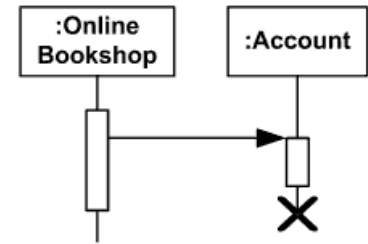
- varie viste e calcolo di metriche ....
  - Es. dipendenze tra moduli/classi, accoppiamento tra classi





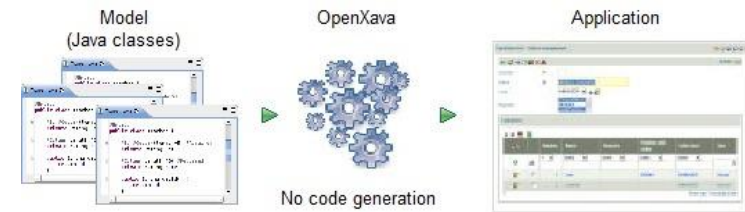
# VISUAL PARADIGM

- UML design tool / **UML modeller**
  - Serve per progettare un sistema software in UML

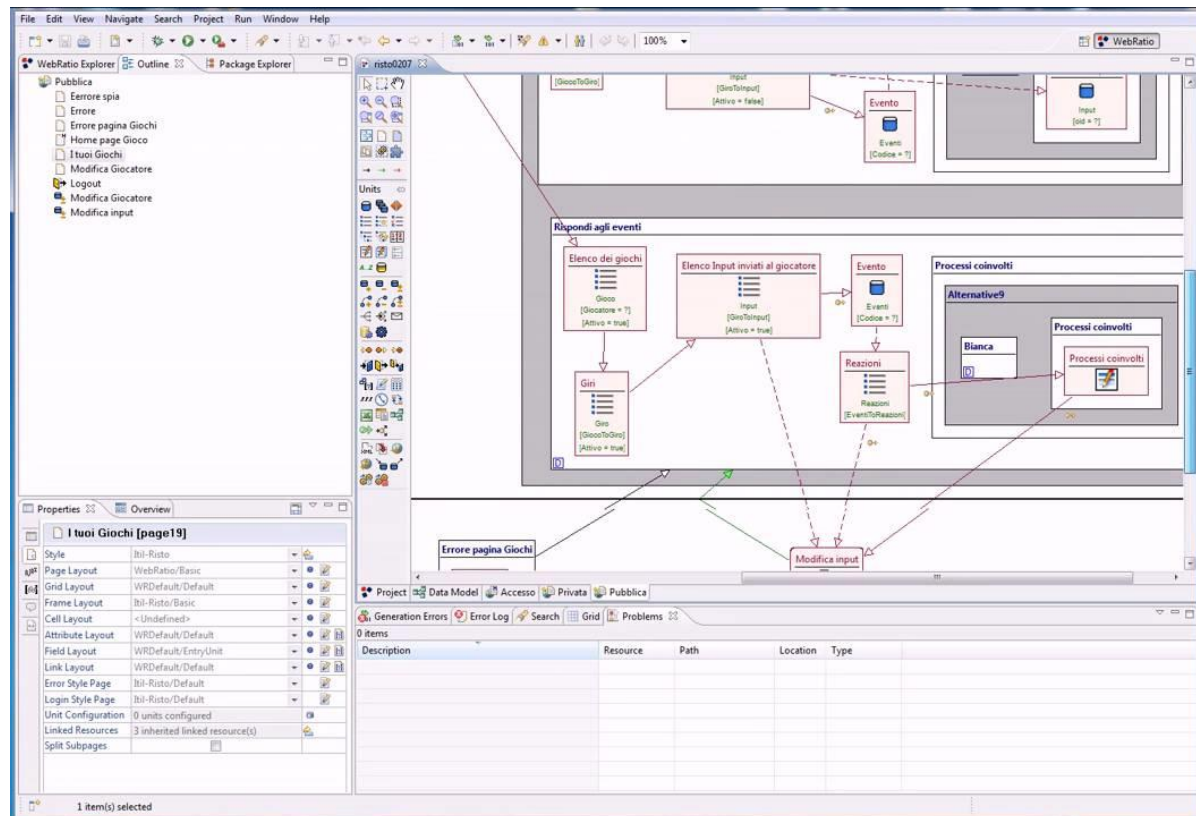


*Vari tipi di diagrammi ....*

# WEBRATIO



- Esempio di Model driven development tool e Component based SE
  - Plug-in di Eclipse



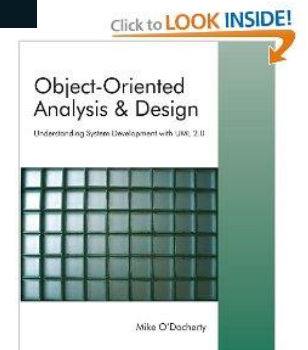
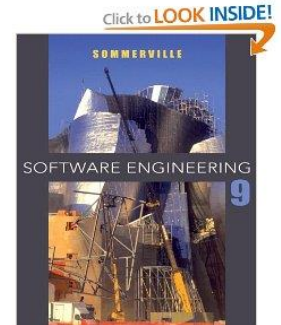
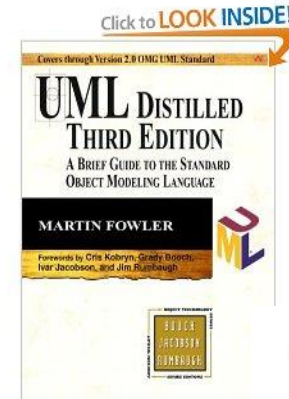
# LIBRI CONSIGLIATI

**Fowler M.** *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3<sup>rd</sup> edition. Addison-Wesley, 2003

**Sommerville I.** *Software Engineering* 10<sup>th</sup> edition, Addison Wesley, 2015

**Pfleeger S. L.** *Software Engineering Theory and Practice* 4<sup>rd</sup> edition, Prentice Hall, 2010

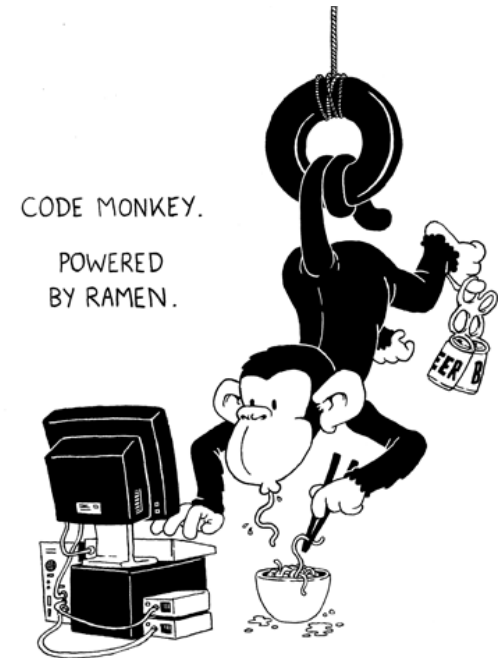
**O' Docherty M.** *Object-Oriented Analysis and Design: Understanding System Development with UML 2.0*, Addison-Wesley, 2005



# PERCHÉ STUDIARE INGEGNERIA DEL SW?

Per non diventare un **code monkey** ...

- Da wikipedia



L'espressione *Code monkey* (letteralmente, *scimmia programmatrice*) si riferisce generalmente ad un programmatore per computer; entrando nello specifico, il termine **si riferisce a quelle persone capaci unicamente di scrivere del codice, inabili dunque nel compiere lavori che richiedono un tasso di astrazione più elevato come il curare l'architettura, l'analisi ed il design del software**. In tal senso, il termine è considerabile leggermente offensivo ...

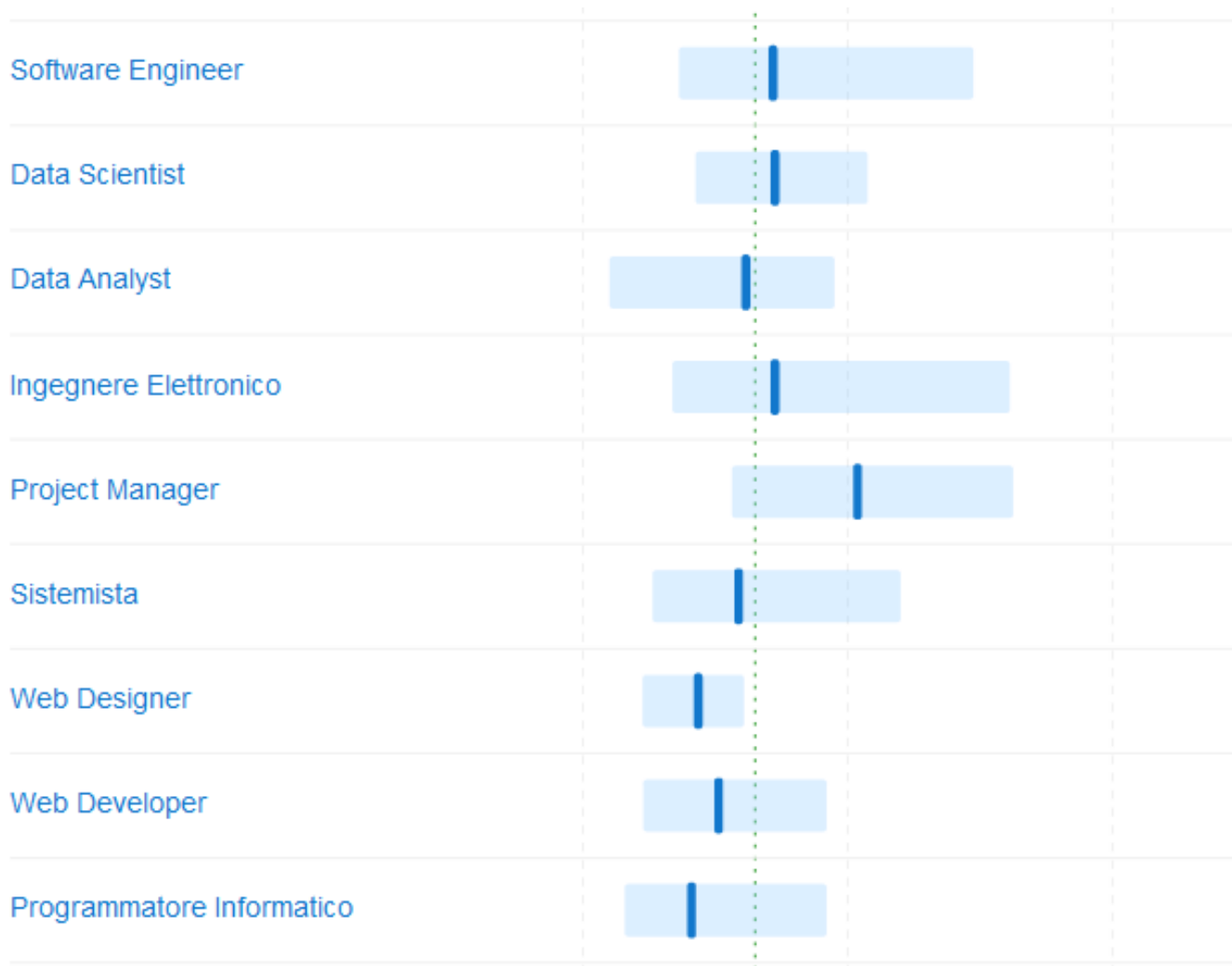
Ingegneria del SW

# STIPENDI IT ITALIA (ANNO 2023)



<https://www.jobbydoo.it/stipendio/programmatore-informatico>

circa 1.970 €  
netti al mese



circa 1.450 €  
netti al mese

# QUESTIONI BUROCRATICHE/LOGISTICHE

- Crediti: **6**
  - circa 150 ore di lavoro (48 ore lezione/laboratorio)
- **Lezioni:**
  - Lunedì 11-13
  - Giovedì 11-13
- Materiale: **AulaWeb**
  - Slides, **Podcast**, Esercizi, Forum, Date Esami, Risultati Esami ...
- **Esame:**
  - **Scritto**
    - Prima parte: Quiz di sbarramento [si/no]
    - Seconda parte: domanda teorica aperta e esercizio
  - **Laboratorio svolto in gruppo**
    - Gli elaborati fatti in Lab vanno consegnati!

27 punti

[0,4]  
punti

# THE END ...



## *Domande?*

# TERMINOLOGIA

- **Metodo** = insieme di regole, principi e tecniche da seguire per ottenere un risultato
  - Analogia in cucina: metodi di cottura a vapore, a micro onde, in forno
- **Procedura** = passi da fare per ottenere un particolare risultato
  - Analogia in cucina: una ricetta
- **Tool** = strumento software per lo sviluppo/analisi del software
  - Analogia in cucina: il frullatore