01. Ingegneria del Software

IS 2024-2025

Laura Semini, Jacopo Soldani
Corso di Laurea in Informatica
Dipartimento di Informatica, Università of Pisa

DOCENTE CORSO A

Laura Semini

Dipartimento di Informatica, Università of Pisa

laura.semini@unipi.it

RISORSE

Teams

Slide (caricate sul sito del corso)

D. C. Kung. **Software Engineering** (2ed), MCGraw Hill, 2023. ISBN: 978-1260721706

M. Seidl et al. **UML @ Classroom: An introduction to Object-Oriented Modeling**, Springer, 2015. ISBN: 9783319127415

Esercizi (compiti anni precedenti, cfr. http://didawiki.di.unipi.it/doku.php/informatica/is-a/start)

RISORSE (CONT.)

Testi di approfondimento/consultazione:

- M. Fowler. **UML Distilled: Guida rapida al linguaggio di modellazione standard** (4ed), Pearson, 2018. ISBN: 978-8891907820
- M. Pezzè, M. Young. **Software Testing and Analysis: Process, Principles, and Techniques**. Wiley, 2008. ISBN: 978-0471455936
  // Utile quando parleremo di testing

Dispense (scaricabili dalla pagina del corso):

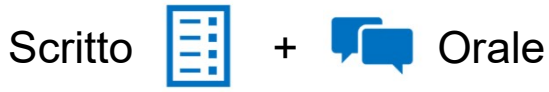
- C. Montangero, L. Semini. **Dispensa di architettura e progettazione di dettaglio**.
  // Utile quando parleremo di progettazione (circa metà corso).
- C. Montangero, L. Semini (a cura di). **Il controllo del software - verifica e validazione**.
  // Utile nelle ultime lezioni del corso

Altro materiale che verrà reso disponibile quando necessario

ESAME

Basato su un **caso di studio** (comune con Basi di Dati) il cui testo viene pubblicato **5 gg prima** dell'esame

L'orale può essere sostituito con le **due prove in itinere**, che rimangono valide per la **sessione invernale**



Il voto dello scritto vale per l'**intero anno accademico**, ovvero fino all'appello di Settembre (incluso)

OBIETTIVI DI APPRENDIMENTO

Introduzione alle tecniche di modellazione dell'ingegneria del software

Conoscenze

- Principali modelli di processi di sviluppo software
- Tecniche di modellazione proprie delle varie fasi

Capacità

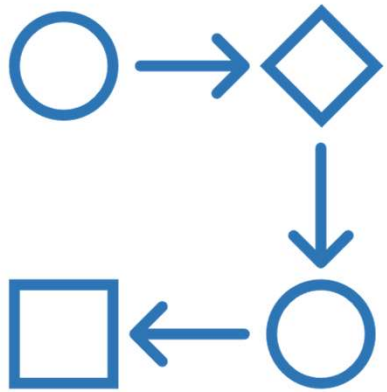
- Utilizzare notazioni di modellazione come UML2 per
  - analisi dei requisiti e
  - progettazione sia architettonica sia di dettaglio di un sistema software

MA COS'È L'INGEGNERIA DEL SOFTWARE?

PROCESSO SOFTWARE

Con **processo software** si intende il modo con cui produciamo il software

- Comincia quando iniziamo ad esplorare il problema
- Finisce quando il prodotto viene ritirato dal mercato
- Strutturato in **fasi**:
  - Analisi dei requisiti
  - Specifica
  - Progettazione
  - Implementazione
  - Integrazione
  - Mantenimento
  - Ritiro



- Riguarda tutti gli **strumenti**, le **tecniche** e i **professionisti** coinvolti nella varie fasi

INGEGNERIA DEL SOFTWARE: UNA DEFINIZIONE

L'ingegneria del software è un approccio sistematico per **sviluppo**, **operatività**, **manutenzione** e **ritiro** di software[1]

Lo scopo è quello di produrre software che sia:

- fault-free
- consegnato nei tempi previsti,
- rispetti il budget iniziale,
- soddisfi le necessità del committente,
- sia facile da modificare

1. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. https://ieeexplore.ieee.org/document/159342

INGEGNERIA DEL SOFTWARE: FAQs

A cosa serve l'ingegneria del software?
Disciplina il processo/modo di produzione del software (o processo software, in breve)

Quando **inizia** il processo software?
Quando cominciamo a esplorare il problema che deve essere risolto da un software

Quando **termina** il processo software?
Quando il software viene ritirato dal mercato

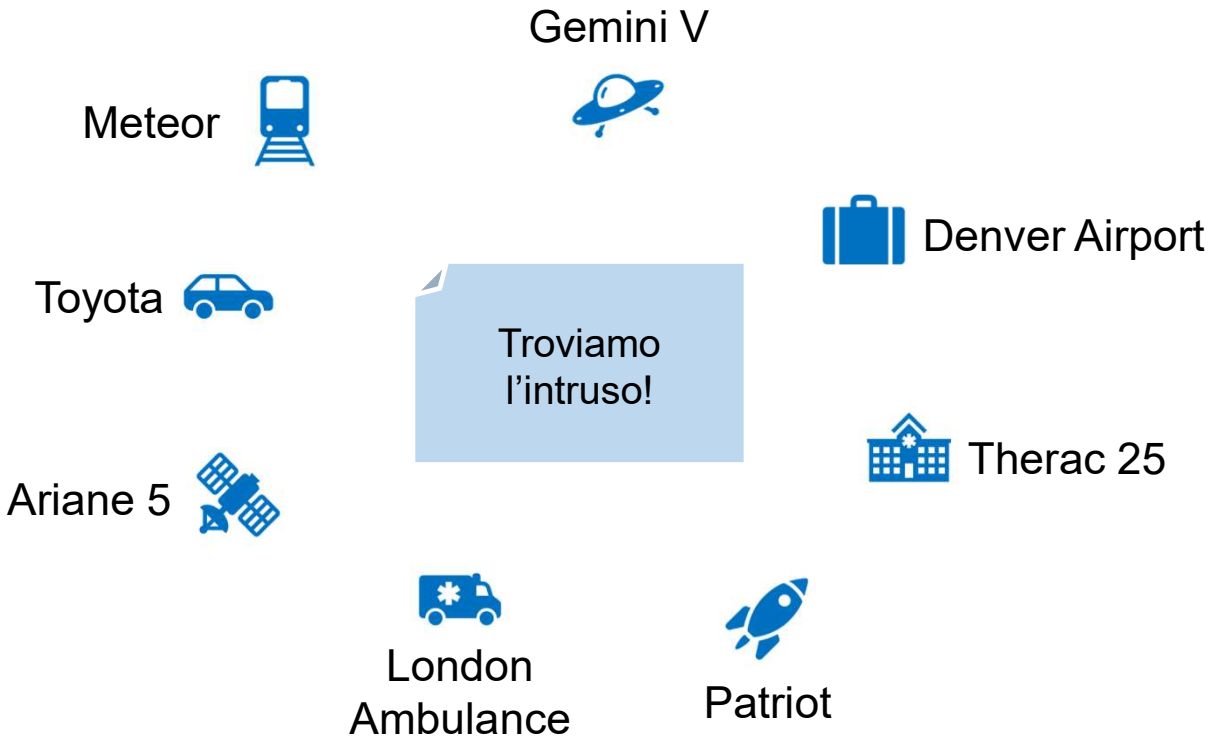
Quali sono le **fasi** del processo software?
Analisi requisiti, specifica, progettazione, sviluppo, integrazione, mantenimento, ritiro

Chi/cosa è **coinvolto** nel processo software?
Professionisti + strumenti/tecniche per specifica, svilippo e mantenimento del software

INGEGNERIA DEL SOFTWARE: FAQs

Perché è **importante** l'ingegneria del software?

Alcuni esempi:



GEMINI V (1965)

Missione nello spazio

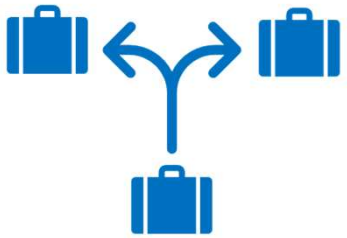
- con equipaggio (Neil A. Armstrong, Elliot M. See)
- parte del programma Gemini degli USA.

Al rientro, la navicella atterrò a 80km dal punto di atterraggio previsto!

Errore nel **modello**!

Uno sviluppatore inserì 360°/24h come velocità di rotazione della Terra (invece di 360,98°/24h)

DENVER AIRPORT (1995-2005)



Sistema automatico di smistamento dei bagagli

- 35 Km di rete, 4000 carrelli, 5000 “occhi”
- $ 193.000.000 di investimento

Problemi:

- Inaugurazione dell’aereoporto ritardata di 16 mesi (con $ 1.000.000 di perdite al giorno)
- Sforamento di 3,2 miliardi di dollari rispetto ai preventive
- Dopo anni di tentativi di “aggiustarlo”, abbandonato nel 2005

Progetto **troppo complesso** e scarsa gestione degli **errori**!

- Sbalzi di potenza che mandavano in tilt il sistema
- Più di 100 PC singoli fisicamente distribuiti
- Il guasto di un PC poteva causare un'interruzione: non esisteva un backup automatico per i componenti guasti (**no fault tolerance**)
  - si perdeva traccia di quali carrelli fossero pieni e quali vuoti dopo un riavvio
- Incapacità di rilevare gli inceppamenti: quando si verificava un inceppamento, il sistema continuava ad accumulare sempre più valigie, peggiorando la situazione (**no robustezza**)

ROBUSTEZZA? FAULT TOLERANCE?

Robustezza (aka. resistenza ai guasti)

- Capacità di un software di **mantenere** il suo **corretto funzionamento** anche quando viene sottoposto a condizioni anomale, errori nell'input dell'utente o situazioni impreviste
- Un software robusto è in grado di gestire errori, eccezioni e input non validi **senza interrompere** il suo **funzionamento** o causare danni irreversibili ai dati o all'utente
- Contribuisce a garantire che il sistema sia **affidabile** e che possa continuare a operare in modo accettabile anche in **situazioni critiche o inaspettate**

Fault tolerance (aka. tolleranza ai guasti)

- Capacità di un software di **rilevare, gestire** e **riprendersi** da **errori o guasti** senza causare interruzioni significative nell'erogazione dei servizi o la perdita permanente di dati critici
- Un software fault tolerant **continua/riprende a funzionare** in modo affidabile anche in presenza di guasti (es. ripristinando uno stato precedente o passando a modalità alternative)
- La tolleranza ai guasti mira a **minimizzare l'impatto** negativo di eventuali **malfunzionamenti**, mantenendo la continuità operativa e la disponibilità del sistema

THERAC-25 (1985-1987)

Macchina per la radioterapia (11 esemplari installati in ospedali di USA e Canada)

6 incidenti in 2 anni

- Dose di radiazioni 100 volte superiore ai limiti consentiti
- Avvelenamento da radiazioni (3 decessi su 6 pazienti)

Sistema **mal progettato** e **poco robusto, difetto latente**

La commissione di inchiesta rilevò che il software era stato progettato e sviluppato

- rendendo praticamente impossibili i test automatici,
- senza considerare come avrebbe dovuto funzionare e comportarsi in caso di errore
  - quando il sistema rilevava un malfunzionamento, il terminale video mostrava un messaggio di errore, ma l'emissione di raggi veniva arrestata solamente in alcuni casi
  - in caso di malfunzionamento, l'operatore aveva la possibilità di premere il tasto "P" per ignorare l'errore e proseguire il trattamento

SISTEMA ANTI-MISSILE PATRIOT (1991)

Un campo base a Dhahran (Arabia Saudita) fu colpito da un missile iracheno causando la morte di 28 soldati

Sistema **poco robusto**

Concepito per lavorare ininterrottamente per 14 ore, fu usato per 100 ore continuative

- Il tempo era modellato in secondi dall'accensione: la conseguenza è stato un errore di approssimazione nel calcolo della traiettoria
- A causa dell'elevata velocità del missile (rilevato), questo portò ad un errore di circa 600 metri

LONDON AMBULANCE SERVICE (1992)

Sistema per gestire il servizio ambulanze di Londra

- Ottimizzazione dei percorsi
- Guida vocale degli autisti

Risultati:

- 3 versioni, per un costo totale di € 11.000.000
- L'ultima versione abbandonata dopo soli 3 giorni d'uso

Analisi errata del problema

Alcuni problemi:

- interfaccia utente inadeguata
- poco addestramento utenti
- sovraccarico non considerato
- nessuna procedura di backup
- scarsa verifica del sistema

ARIANE 5 (1996)

Lanciatore sviluppato sotto autorizzazione di ESA (European Space Agency)

Primo lancio

- quattro satelliti per lo studio della magnetosfera terrestre
- autodistruzione del razzo dopo 40 secondi (3,5 km di altezza)[1]

Errori durante **sviluppo** e **test insufficienti**

Il software (progettato per Ariane 4) tenta di convertire la velocità laterale del missile dal formato a 64 bit al formato a 16 bit

- l'Ariane 5 vola molto più velocemente dell'Ariane 4, e il valore della velocità laterale è più elevato di quanto possa essere gestito dalla routine di conversione
- **overflow**, spegnimento del sistema di guida, e trasferimento del controllo al secondo sistema di guida, che però essendo progettato allo stesso modo è andato in tilt nella medesima maniera
- test con dati vecchi.

Fu necessario quasi un anno e mezzo per capire quale fosse stato il malfunzionamento che aveva portato alla distruzione del razzo

1. https://www.youtube.com/watch?v=5tJPXYA0Nec

IL CASO TOYOTA (2000-2013)

«The National Highway Traffic Safety Administration […] received more than 6,200 complaints involving sudden acceleration in Toyota vehicles. The reports include 89 deaths and 57 injuries» (*Toyota "Unintended Acceleration" Has Killed 89*, CBS News, 2010)

Con **accelerazione inattesa** si intende l'accelerazione involontaria, imprevista e incontrollata di un veicolo, spesso accompagnata da un'apparente perdita di efficacia della frenata (la macchina accelerata richiede di applicare ai freni una forza pari a 80 kgp invece che 7-20 kgp)

NB: Casi simili si sono verificati anche con altre automobili, per esempio Honda e Tesla

Non il primo caso di problema software che ha causato morti ed è andato in giudizio, tuttavia

- è stato significativo per portata dei **risarcimenti** (milioni di dollari) e attenzione **mediatica**,
- contribuendo ad aumentare la consapevolezza dei problemi legati al software nell'**industria automobilistica**

La giuria, sopportata da esperti, ha ritenuto il software Toyota **mal progettato** e **non conforme** agli standard del settore

METEOR (1998)

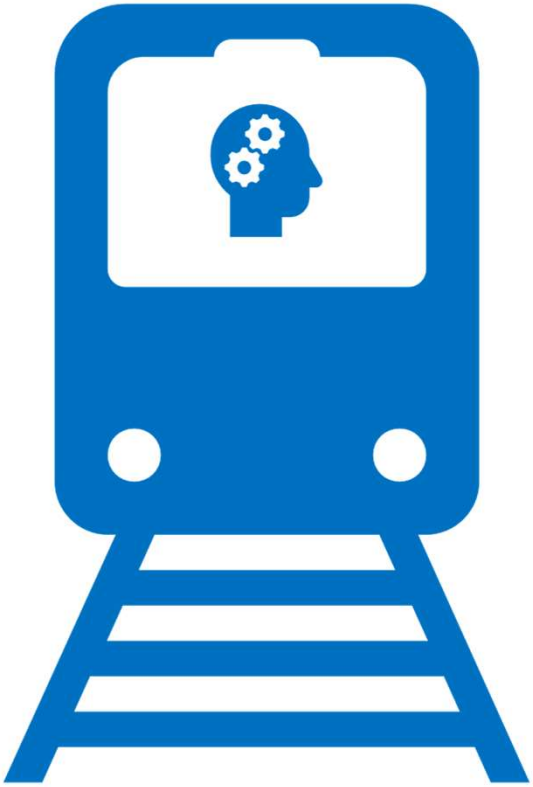
Meteor: Metro Est-Ovest Rapide

- Linea 14 della metropolitana di Parigi
- Prima linea integralmente automatizzata
- 8 km, 7 stazioni, 19 treni, un treno ogni 85 secondi

Progettato e realizzato da Siemens Transportation Systems

- B-method di Abrial
- Abstract machines
- Generazione di codice (ADA, C, C++)

Un successo!
(meritato, poiché **ben progettato**)



ALTRO?

Per approfondimenti e molti altri casi

https://en.wikipedia.org/wiki/List_of_software_bugs

E QUINDI?

I sistemi software sono sempre più presenti nella vita di tutti i giorni

Necessità di **software affidabile**, prodotto **rapidamente** e in modo **sostenibile** (sia economicamente, sia dal punto di vista ambientale)

L'**ingegneria del software** è un approccio sistematico per sviluppo, operatività, manutenzione e ritiro di software[1]

Nel lungo periodo, **conviene** utilizzare metodi/tecniche di **ingegneria del software**, invece di sviluppare software come se si trattasse di un "progetto personale"

⇒ Nella maggior parte dei sistemi, i costi principali si hanno **dopo** che il software è stato rilasciato in produzione e viene utilizzato

1. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. https://ieeexplore.ieee.org/document/159342

UN PO' DI STORIA

1963-1964: Durante lo sviluppo dei sistemi di guida e navigazione per le missioni Apollo, l'informatica americana **Margaret Hamilton** conia il termine ***software engineering***

> *«I fought to bring the software legitimacy so that it (and those building it) would be given its due respect and thus I began to use the term "software engineering" to distinguish it from hardware and other kinds of engineering.»*
> M. Hamilton

UN PO' DI STORIA (continua)

1968: NATO **Software Engineering Conference**, Garmish

- **Crisi**: qualità del software era in generale **inaccettabilmente bassa**
- **Soluzione**: La produzione di software deve usare tecniche e paradigmi come le consolidate discipline ingegneristiche (aka. **ingegneria del software**)

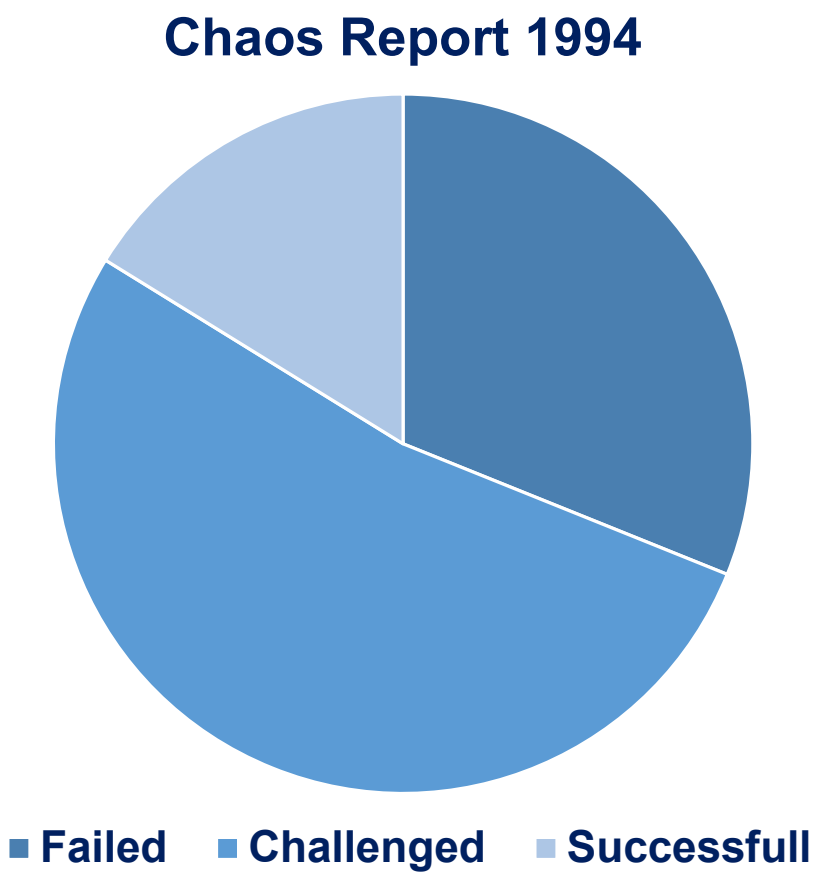
Si passa quindi

- da software sviluppato informalmente (es. risoluzione di equazioni) a grandi sistemi commerciali (es. sistemi informativi aziendali, per gestire tutte le informazioni delle funzioni aziendali)
- dalla programmazione individuale alla programmazione di squadra

UN PO' DI STORIA (CONT.)

1994: Analisi dei progetti software (Standish Group)

- Completati **in tempo**: 16.2 %
- Completati **in ritardo** (almeno il doppio del tempo): 52.7 %
  - Difficoltà nelle fasi iniziali dei progetti
  - Cambiamento di piattaforma e tecnologia
  - Difetti nel prodotto finale
- **Abbandonati**: 31.1 %
  - Per obsolescenza prematura
  - Per incapacità di raggiungere gli obiettivi
  - Per esaurimento dei fondi



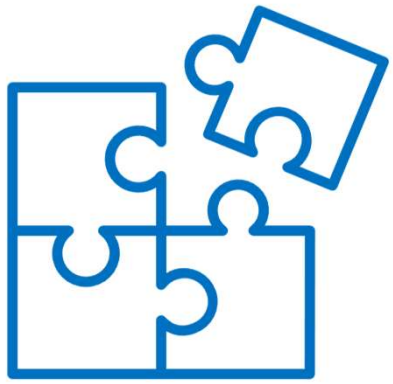
STANDISH GROUP: CAUSE DI ABBANDONO

Project Challenged Factors	% of Responses
1. Lack of user input	12.8
2. Incomplete requirements & specifications	12.3
3. Changing requirements & specifications	11.8
4. Lack of executive support	7.5
5. Technology incompetence	7.0
6. Lack of resources	6.4
7. Unrealistic expectations	5.9
8. Unclear objectives	5.3
9. Unrealistic time frames	4.3
10. New technologies	3.7
Other	23

SPECIFICITÀ DEL SOFTWARE

Un sistema software è **diverso** da altri prodotti ingegnerizzabili

- Non è necessariamente vincolato da materiali
- Non è governato da leggi fisiche o da processi manufatturieri
- Non ha costi marginali (costo di un'unità aggiuntiva prodotta)
- Non si «consuma»
- Spesso si «assembla»



SPECIFICITÀ DEL SOFTWARE: UN ESEMPIO

Altri prodotti dell'ingegneria (es. edifici, macchine, ecc.)

- Quando si **rompono non** è possibile **aggiustarli** come fossero arabe fenici

Quando un **sistema software** (es. sistema operativo) **«crasha»**lo facciamo ripartire

- Questo perché è stato progettato per minimizzare gli effetti di un fallimento (es. evitando di perdere i documenti su cui si stava lavorando)
- La **fault tolerance** è una qualità (ed una specificità) del software

Questa (ed altre specificità) devono essere considerate quando si analizzano i requisiti e gli obiettivi di un sistema software (e durante tutte le altre fasi del processo software)

STANDISH GROUP: CAUSE DI ABBANDONO

Project Challenged Factors	% of Responses
1. Lack of user input	12.8
2. Incomplete requirements & specifications	12.3
3. Changing requirements & specifications	11.8
4. Lack of executive support	7.5
5. Technology incompetence	7.0
6. Lack of resources	6.4
7. Unrealistic expectations	5.9
8. Unclear objectives	5.3
9. Unrealistic time frames	4.3
10. New technologies	3.7
Other	23

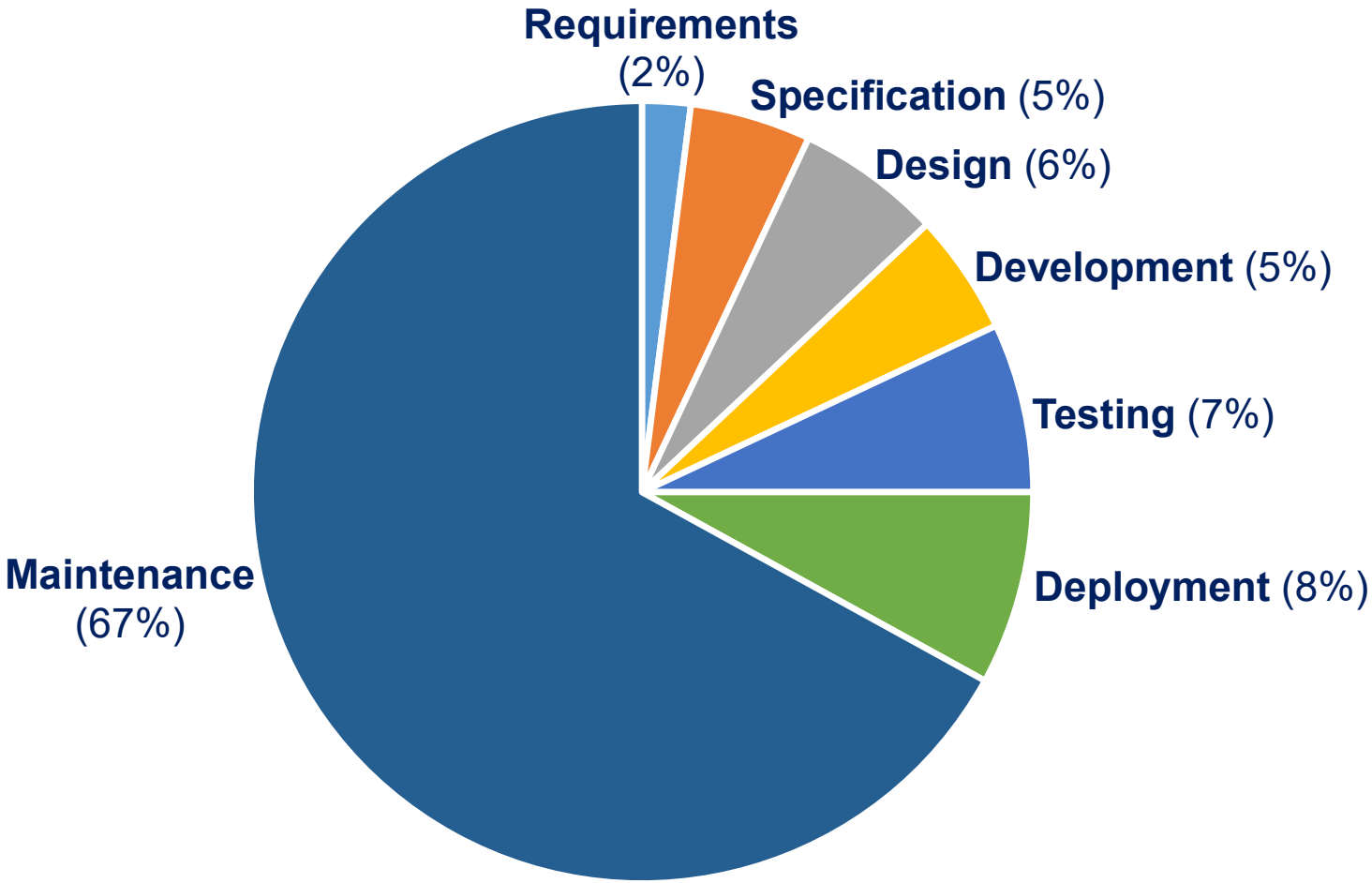
ASPETTI ECONOMICI

L'ingegneria del software si occupa anche di valutare e/o introdurre soluzioni ritenute **economicamente vantaggiose**

Ad esempio:

- L'azienda **AZSW** utilizza una tecnica $\mathbf{T_{old}}$ per la progettazione e lo sviluppo dei sistemi software che vende
- Viene proposta una nuova tecnica $\mathbf{T_{new}}$ che (rispetto a $\mathbf{T_{old}}$) consente di velocizzare la scrittura del codice di un fattore 10
- Non è detto che **AZSW** scelga di addottare $\mathbf{T_{new}}$
  - Costi dell'introduzione di $\mathbf{T_{new}}$ nel processo software
  - Costi per il training del personale all'utilizzo di $\mathbf{T_{new}}$
  - Costi di manutenzione dei prodotti software

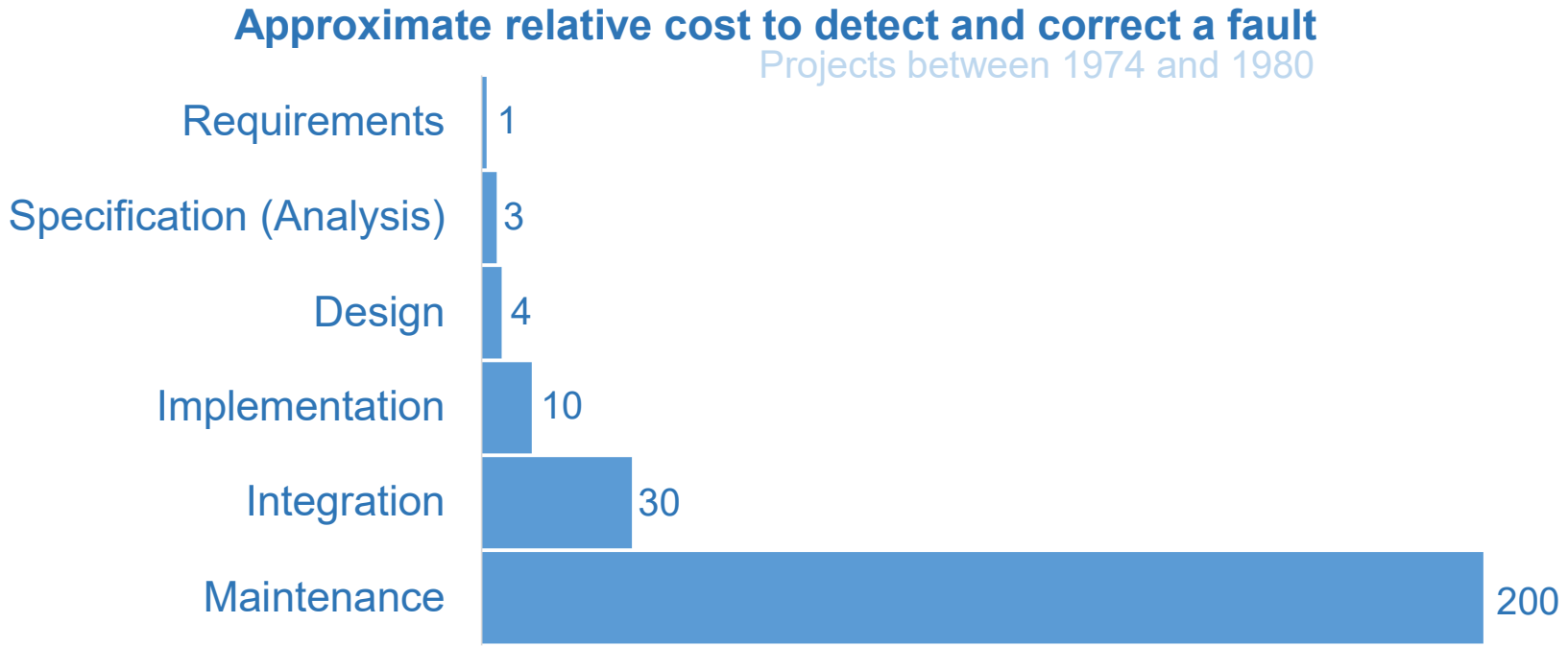
I COSTI DEL SOFTWARE



L'IMPORTANZA DELL'ANALISI DEI REQUISITI

Se si introduce un errore durante l'analisi dei requisiti, l'errore apparirà nella specifica, nella progettazione e nel codice del sistema software

Prima individuiamo l'errore meglio è!



LA MANUTENZIONE DEL SOFTWARE

La **manutenzione** include tutti i cambiamenti ad un prodotto software, anche dopo che è stato rilasciato in produzione

Diversi **tipi** di manutenzione:

- **correttiva** (~20%): rimuove gli errori/bug lasciando invariata la specifica
- **migliorativa**: consiste in cambiamenti di specifica e implementazione, ovvero
  - manutenzione **perfettiva** (~60%): modifiche applicate per migliorare le qualità del software, fornire nuove funzionalità, o migliorare funzionalità esistenti
  - manutenzione **adattativa** (~20%): modifiche a seguito di cambiamenti nel contesto del software (ad esempio, cambiamenti legislativi, nel hardware, o nel sistema operativo)

Esempio: IVA dal 20% al 22%

⇒

```
float aliquota=22;
…
prezzotot = prezzo + (prezzo*aliquota)/100
```

STANDISH GROUP: CAUSE DI ABBANDONO

Project Challenged Factors	% of Responses
1. Lack of user input	12.8
2. Incomplete requirements & specifications	12.3
3. Changing requirements & specifications	11.8
4. Lack of executive support	7.5
5. Technology incompetence	7.0
6. Lack of resources	6.4
7. Unrealistic expectations	5.9
8. Unclear objectives	5.3
9. Unrealistic time frames	4.3
10. New technologies	3.7
Other	23

LAVORO IN TEAM

La maggior parte del software è oggi prodotto da **team** di programmatori

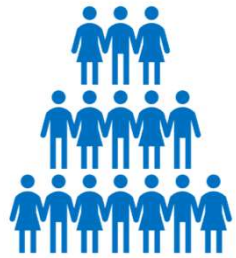
Il lavoro in team pone dei **problemi**:

- interfaccia tra le diverse componenti del codice
- comunicazione tra i membri del team

⇒ molto tempo da dedicato a relazioni intra-/inter-team (es. riunioni tra i vari componenti)

L'ingegnere del software deve essere anche capace di:

- gestire i rapporti umani e organizzare un team
- gestire gli aspetti economici e legali



E QUINDI?

Per essere in grado di affrontare le problematiche elencate finora, nel corso ci occuperemo di:

- **processo software**, inclusi
  - organizzazione e gestione dei progetti
  - definizione e correlazione delle attività
  - metodi di composizione dei gruppi di lavoro
  - strumenti di pianificazione, analisi, controllo
  - modelli ideali di processo di sviluppo

E QUINDI? (CONT.)

Per essere in grado di affrontare le problematiche elencate finora, nel corso ci occuperemo di:

- **processo software**, inclusi {…}
- **realizzazione** di sistemi software, inclusi
  - strategie di **analisi e progettazione**
    - Tecniche per la comprensione e la soluzione di un problema
    - Top-down, bottom-up, progettazione modulare, OO
  - Linguaggi di **specifica** e **progettazione**
    - Strumenti per la definizione di sistemi software
    - Reti di Petri, Z, OMT, UML
  - **Ambienti** di sviluppo
    - Strumenti per analisi, progettazione e realizzazione
    - Strumenti tradizionali, CASE, CAST

E QUINDI? (CONT.)

Per essere in grado di affrontare le problematiche elencate finora, nel corso ci occuperemo di:

- **processo software**, inclusi {…}
- **realizzazione** di sistemi software, inclusi {…}
- **qualità** del software, inclusi
  - **Modelli** di qualità
    - Definizione di caratteristiche della qualità
  - **Metriche** software
    - Unità di misura, scale di riferimento, strumenti
    - Indicatori di qualità
  - Metodi di **verifica** e **controllo**
    - Metodi di verifica, criteri di progettazione delle prove
    - Controllo della qualità, valutazione del processo di sviluppo

E QUINDI? (CONT.)

Per essere in grado di affrontare le problematiche elencate finora, nel corso ci occuperemo di:

- **processo software**, inclusi {…}
- **realizzazione** di sistemi software, inclusi {…}
- **qualità** del software, inclusi {…}

Tenendo in considerazione i vari **stakeholder**, ovvero:

- **fornitore**, aka. chi sviluppa un software
- **committente**, aka. chi richiede (e paga) per un software
- **utente**, aka. chi usa un software


Q
A

RIFERIMENTI

Contenuti

- **Capitolo 1** di “Software Engineering” (D. C. Kung), McGraw Hill, 2023.

Approfondimenti

- https://www.bcg.com/publications/2020/increasing-odds-of-success-in-digital-transformation
- https://www.codemag.com/Article/2303091/Architects-The-Case-for-Software-Leaders

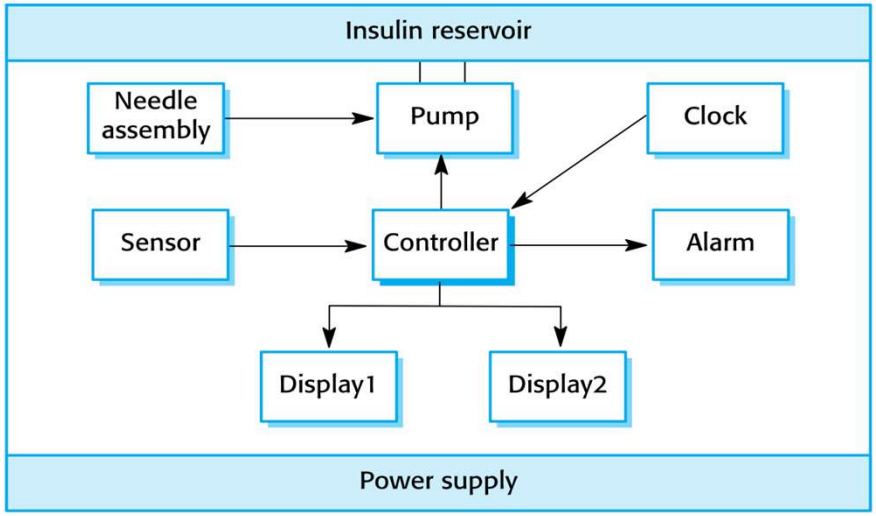
CASI DI STUDIO

Come riferimento, spesso utilizzeremo i seguenti casi (presi dal Sommerville)

- **Insulin pump**
  - Sistema embedded in una pompa di insulina usata da diabetici per tenere i livelli di glucosio nel sangue sotto controllo
- **Mentcare**
  - Sistema per la gestione di pazienti affetti da problemi mentali
  - Mantiene informazioni sui pazienti e sulle cure che stanno ricevendo
- **Wilderness weather station**
  - Sistema di collezionamento di dati riguardanti le condizioni meteo in aree remote
- **iLearn**
  - Ambiente di apprendimento digitale
  - Supporto all'apprendimento nelle scuole

INSULIN PUMP

- Colleziona dati da un sensore che misura il livello di glucosio nel sangue e calcola la quantità di insulina da iniettare
- Il calcolo è basato sulla frequenza con cui i livelli di glucosio cambiano
- Invia segnali ad una micro-pompa per iniettare la dose di insulina corretta



Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

NB: Si tratta di un sistema **safety-critical**

- livelli di glucosio troppo bassi possono indurre problemi cerebrali, coma e morte
- livelli di glucosio troppo alti possono indurre danni a lungo termine (problemi alla vista o ai reni)

INSULIN PUMP: ALCUNI REQUISITI

Sistema **safety-critical**

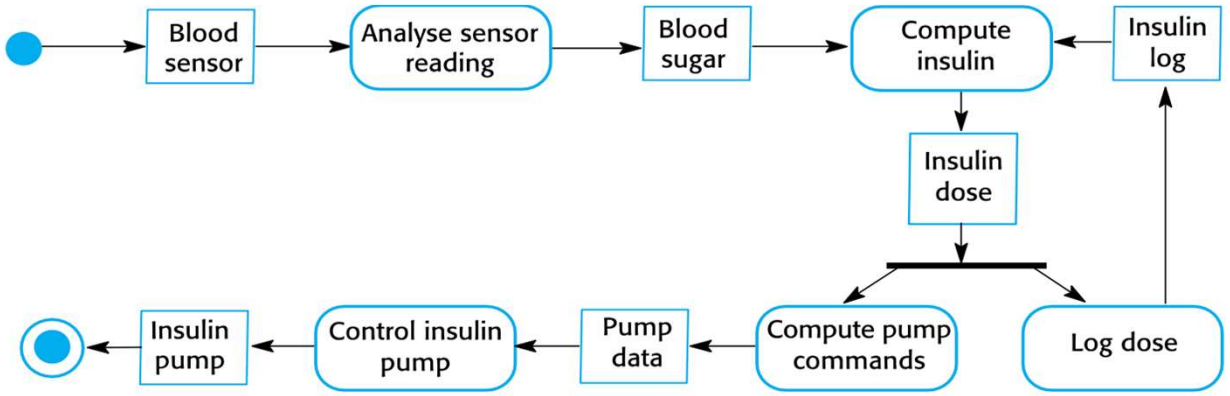
⇒ Il sistema deve essere in grado di **rilasciare insulina** quando richiesto

⇒ Il sistema deve rilasciare la **giusta quantità** di insulina (e non più di quella) per mantenere i giusti livelli di glucosio nel sangue

⇒ Il sistema deve funzionare in modo **affidabile**, rilasciando la giusta quantità di insulina per mantenere i giusti livelli di glucosio nel sangue

Il sistema deve essere progettato e sviluppato in modo da garantire questi (e altri) requisiti siano **sempre soddisfatti**

INSULIN PUMP: MODELLO DI FUNZIONAMENTO



Da: I. Sommerville. **Ingegneria del Software** (10ed), Pearson, 2017

CASI DI STUDIO (CONT.)

Come riferimento, spesso utilizzeremo i seguenti casi (presi dal Sommerville)

Insulin pump

- Sistema embedded in una pompa di insulina usata da diabetici per tenere i livelli di glucosio nel sangue sotto controllo

Mentcare

- Sistema per la gestione di pazienti affetti da problemi mentali
- Mantiene informazioni sui pazienti e sulle cure che stanno ricevendo

Wilderness weather station

- Sistema di collezionamento di dati riguardanti le condizioni meteo in aree remote

iLearn

- Ambiente di apprendimento digitale
- Supporto all'apprendimento nelle scuole

I dettagli li trovate sul libro (e li riprenderemo comunque ogni volta che serve)