Ingegneria del Software

Introduzione

Perché studiamo ingegneria del software:

- ★ Aspetti economici: il software nel mercato è sempre più presente.
- ★ Sviluppo del software professionale (con adeguata documentazione).

Che cos'è un software? È un prodotto(es. programma) più tutta la documentazione.

Le caratteristiche di un buon software sono:

- ★ Manutenibilità
- ★ Usabilità
- ★ Affidabilità e sicurezza
  - Reliability
  - Safety: non crea danni fisici o economici a chi lo utilizza.
  - Security: protezione dei dati da altri accessi.
- ★ Efficienza: quanto velocemente e con quali costi si risolve il problema.
- ★ Efficacia: quanto il software è in grado di risolvere il problema per cui è stato creato.
- ★ Accettabilità: il software deve essere accettato e/o apprezzato dagli utenti.

Tipi di software:

- ★ Prodotti generici: software uguale per tutti gli utenti (es. Word).
- ★ Prodotti personalizzati: dedicati e costruiti su misure per un cliente preciso (es. software per il policlinico di Borgo Roma).

Aspetti che influenzano un sistema software:

- ★ Eterogeneità: prodotto adatto a diversi device.
- ★ Cambiamenti sociali e organizzativi.
- ★ Sicurezza e fiducia: l'utente deve potersi fidare del software.
- ★ Scala: le dimensioni del software.

Tipi di applicazioni:

- ★ Applicazioni Standalone: nate per funzionare su una singola macchina.
- ★ Applicazioni interattive transazionali: come ad esempio un e-commerce, dove avvengono transazioni di denaro e arrivo della merce.
- ★ Sistemi di controllo embedded: sono sistemi software a bordo di device specifici, come i navigatori satellitari.
- ★ Sistemi di elaborazione batch: elaborazione nascosta all'utente.
- ★ Sistemi di intrattenimento: come videogiochi, streaming, etc.
- ★ Sistemi per modellazione, progettazione e simulazione: per utenti specializzati in un certo campo.
- ★ Sistemi di collezione di dati
- ★ Sistemi di sistemi: sono sistemi che coordinano altri sistemi (es Amazon -> Paypal -> banca).
- ★ Sistemi Data-Intensive: raccolte dati per altri servizi (es. suggerimenti agli acquisti).

Fondamenti dell'ingegneria del software:

- ★ Processo di sviluppo: non posso creare software a caso, ma è necessario un approccio.
- ★ Affidabilità e prestazioni.
- ★ Comprensione delle specifiche del software: bisogna avere un'idea chiara su quello che si vuole realizzare.
- ★ Riuso del software: devo sapere se è possibile riutilizzare parti del software già scritto per futuri progetti.
- ★ Software web-based: adatto uno sviluppo incrementale e "agile".
    - Lo sviluppo passa da più soluzioni intermedie che verifico di volta in volta.
    - Basato su servizi e interfacce grafiche.

Etica:

- ★ Responsabilità: chi contribuisce alla scrittura del codice ha la piena responsabilità della scritture del software stesso.
- ★ Onestà e responsabilità:
    - Onestà nelle proprie competenze.
    - In caso di errori bisogna dirli e assumersi la responsabilità.
- ★ Rispetto della legge: per quali fini verrà sviluppato il software?

ACM/IEEE: associazioni internazionali che si occupano di definire standard, codice etico e di informare le persone.

Bisogna prestare attenzione a (responsabilità verso):

- ★ Clienti e impiegati: non si schiavizzano i colleghi.
- ★ Prodotto: utilizzare al meglio i più recenti strumenti.
- ★ Giudizio: avere un giudizio imparziale.
- ★ Gestione.
- ★ Professione: far rispettare il pensiero che la gente ha sulla nostra professione.

Processi Software

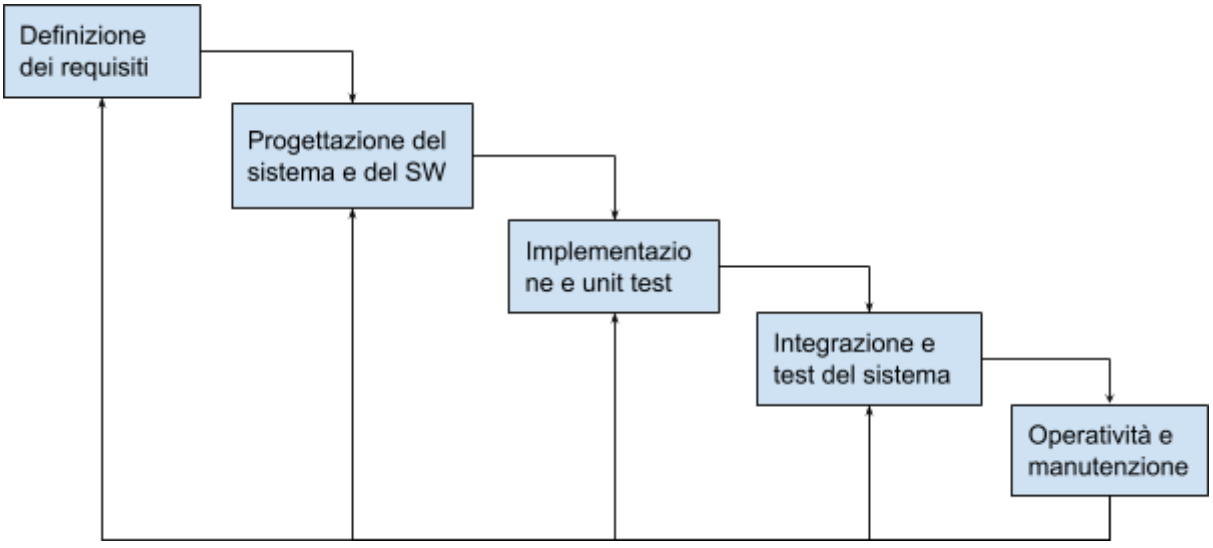
Che cosa è un processo? È un insieme di operazioni opportunamente strutturato per arrivare a un risultato.

I processi software sono un insieme strutturato di attività per realizzare un sistema software.

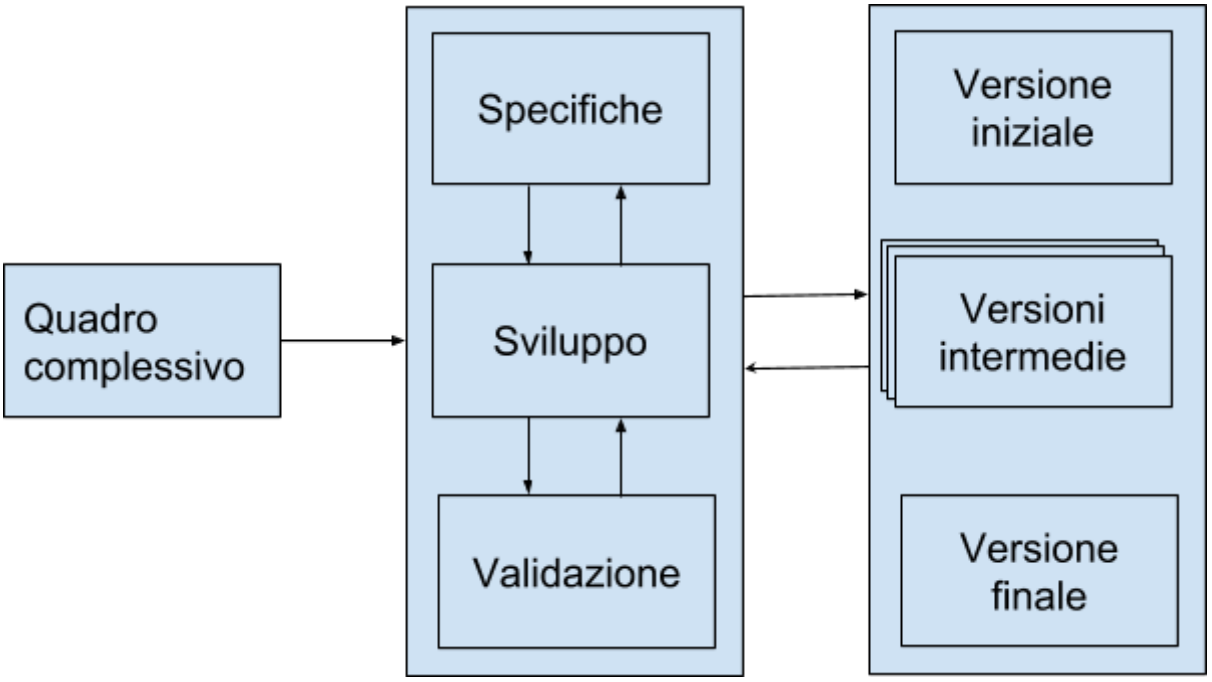
- ★ Specifica: una o più attività che dicono cosa il software dovrà fare (viene coinvolto l'utente finale).
- ★ Progettazione e implementazione: descrivo le parti del software e come devono interagire e infine si scrive il codice delle parti descritte.
- ★ Validazione: assicurarsi che i componenti funzionino singolarmente e successivamente il sistema funzioni nel complesso.
- ★ Evoluzione: una volta finito di sviluppare un software in futuro saranno necessarie evoluzioni e cambiamenti.

Modello di processo software

- ★ Rappresentazione astratta di un processo:
  - Attività che lo compongono, quello che va fatto.
  - Ordine delle attività: dipendenze, quali in parallelo, etc.
  - Prodotti di una attività: che risultati bisogna ottenere al termine di ogni attività.
  - Ruoli: chi fa cosa.
  - Pre e post condizioni: Cosa bisogna fare prima che inizi o al termine di un'attività.
- ★ Un processo può essere:
  - Plan-driven: tutte le attività, la durata, e la divisione dei compiti sono pianificate all'inizio del progetto.
  - Agile: maggiore flessibilità nei tempi e nella pianificazione delle attività.
- ★ **Modello a cascata**:
  - plan-driven
  - Si compone di:



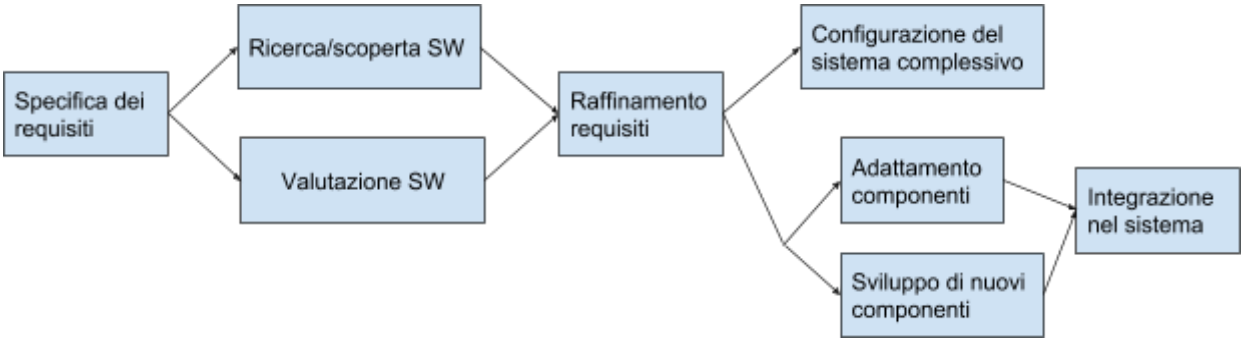
  - Non si torna indietro fino a che non si passa per tutti i punti in ordine.
- ★ **Sviluppo incrementale**:
  - Plan-driven o agile
  - Composto da:



- Non si perde il contatto con l'utente.

★ **Modello basato su riuso del codice**:

- Plan-driven o agile
- Integrazione e configurazione



- \+ Parto da componenti già esistenti
- \- Compromesso

Attività dei processi

➔ Tecnici, collaborativi e gestionali

➔ Specifica (*vedi dopo*)

- ◆ Elicitazione e analisi dei requisiti (l'utente non sa esattamente cosa vuole o cosa vuole viene dato per scontato/sottinteso) ⇒ Descrizione del sistema
- ◆ Specifica dei requisiti ⇒ Requisiti di sistema e utenti
- ◆ Validazione dei requisiti ⇒ Documento dei requisiti

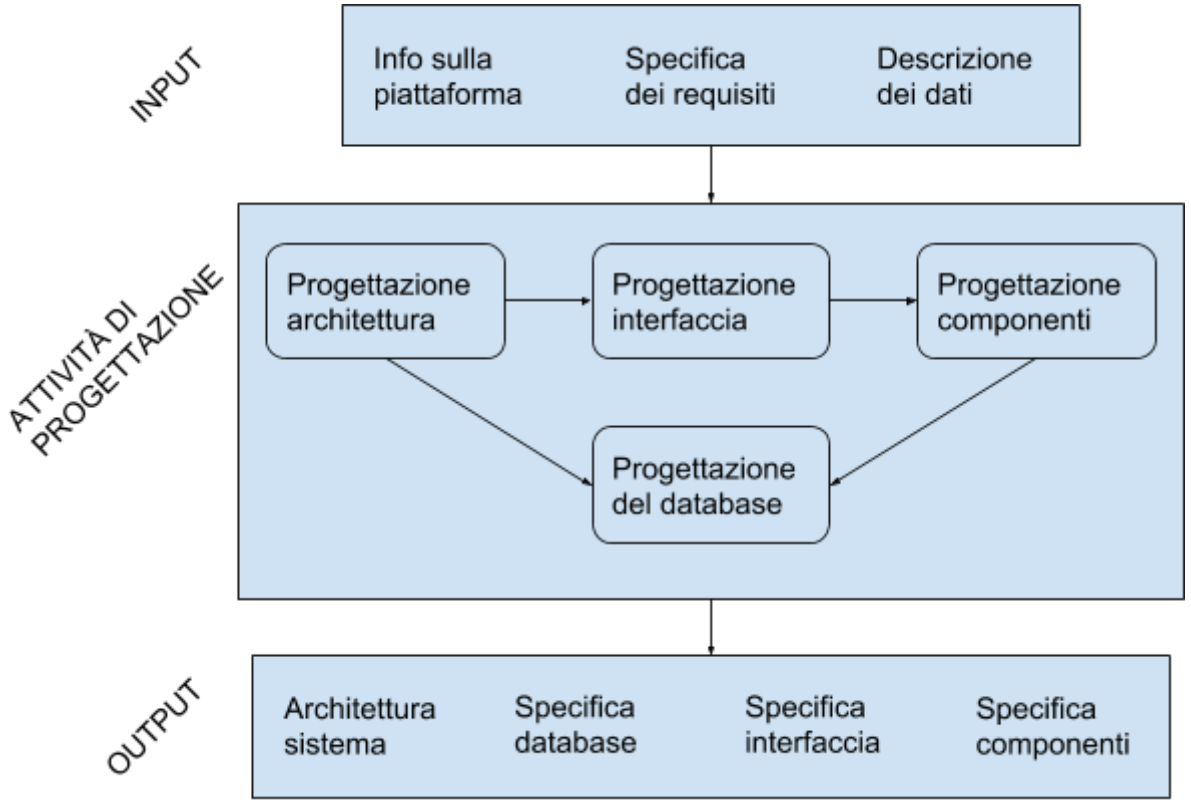
➔ Sviluppo

- ◆ Progettazione
  - Struttura del software
  - Moduli/interfacce
- ◆ Implementazione
  - Traduzione della struttura del software in codice eseguibile (debugging)

➔ Verifica e validazione

➔ Evoluzione

Modello generale del processo di progettazione



Validazione del software

- ➔ Verifica e validazione (V&V)
  - ◆ Software testing
    - Test dei componenti (JUnit)
    - Test del sistema
    - Test sull'utente (Alpha & Beta)

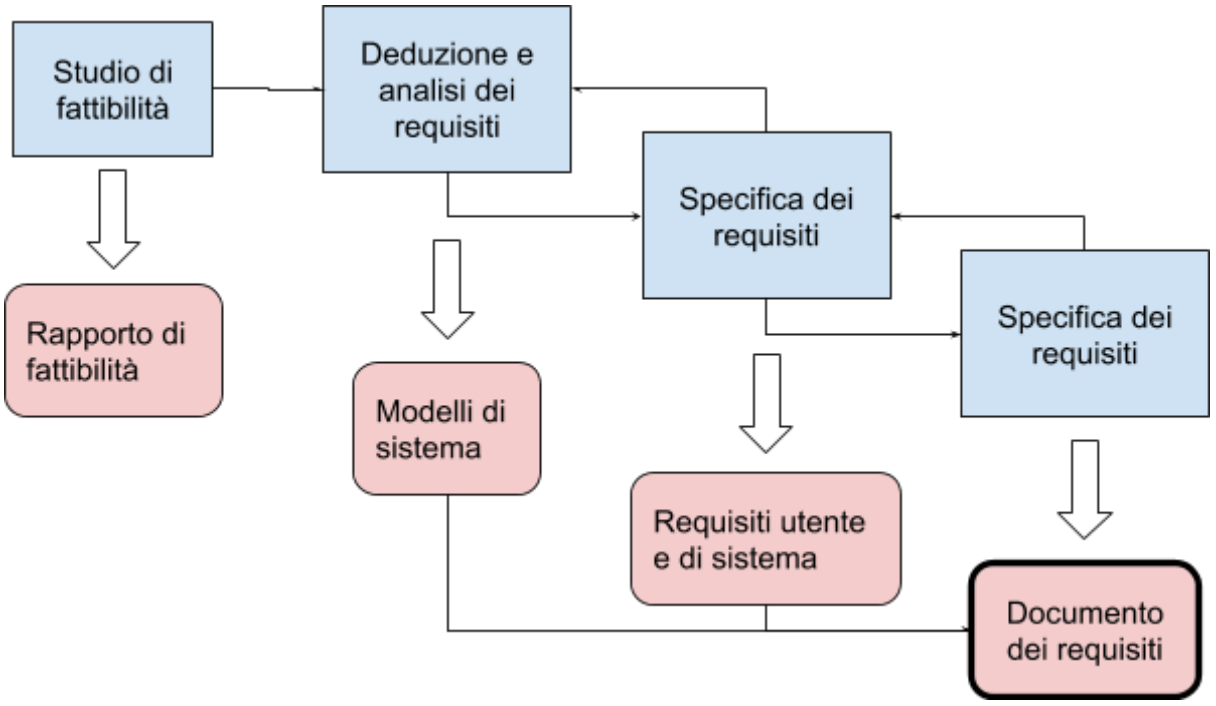
Specifica dei requisiti

- ➔ Successo:
  - ◆ il software che vado a costruire soddisfa i requisiti.
  - ◆ I requisiti soddisfano i bisogni dell'utente.
  - ◆ I bisogni dell'utente rappresentano le necessità reali.

Ingegneria dei requisiti: processo di ricerca, analisi, documentazione e verifica dei requisiti di sistema.

- ➔ Requisiti di sistema: descrizione dei servizi forniti e dei vincoli operativi.

Processo di ingegneria dei requisiti



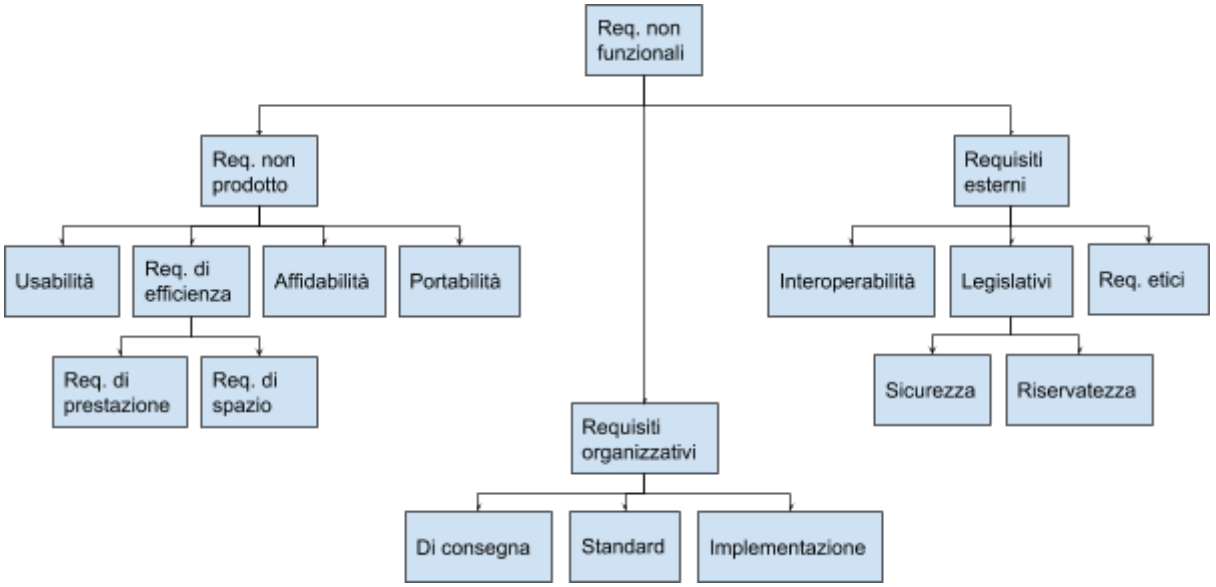
Requisito

- ➔ Formulazione astratta ad alto livello di un servizio che il sistema dovrebbe fornire.
- ➔ Vincolo di sistema.
- ➔ Definizione formale e dettagliata di una funzione del sistema.

Tipi di requisiti

- ➢ **Requisiti utente**: dichiarano in linguaggio naturale i servizi richiesti (possono essere corredati da diagrammi).
  - Alto livello di astrazione
- ➢ **Requisiti di sistema**: sono una formulazione dettagliata e strutturata di funzioni, servizi e vincoli.
  - Definiscono cosa dovrebbe essere implementato scritto in linguaggio naturale.
  - Notazioni semi-formali
  - Livello di astrazione più basso.
- ➢ Un requisito utente può essere espanso in più requisiti di sistema.
- ➢ **Requisiti funzionali**: descrivono i servizi che il sistema deve fornire, possono anche affermare esplicitamente cosa il sistema non deve fare.
  - Dipendono dal tipo di sistema richiesto
  - Possono avere diversi livelli di dettaglio.
- ➢ **Requisiti non funzionali**:
  - Vincoli sui servizi e sul processo di sviluppo.
  - Si applicano al sistema completo.
  - Si riferiscono al sistema nel suo complesso.
  - Non riguardano direttamente le specifiche del sistema relativamente alla sue funzionalità
    - Requisito funzionale
  - Si riferiscono a proprietà del sistema come:
    - Affidabilità
    - Tempi di risposta

        - Occupazione spazio
        - Vincoli



- ➢ **Requisiti di dominio**:
    - Derivano dal dominio applicativo del sistema.
        - Legati al linguaggio specifico del dominio.
    - Impongono nuovi requisiti sul sistema
        - Requisiti funzionali
        - Vincoli su requisiti funzionali già esistenti
        - Particolari requisiti legati a calcoli da effettuare

ESEMPIO:
Il sistema deve essere facile da usare ⇒
Il sistema deve essere facile da usare per i controllori del traffico esperti; e organizzato in modo che gli errori degli utenti siano minimizzati ⇒
I controllori esperto dovrebbero essere capaci di usare tutte le funzioni del sistema dopo un addestramento di 2 ore. Dopo questo addestramento non dovrebbero superare in media 2 errori al giorno.
→ È diventato un requisito non funzionale verificabile.

Documento dei requisiti:
- ➔ Dichiarazione ufficiale di quello che poi gli sviluppatori dovrebbero implementare
- ➔ Deve includere:
    - ◆ Requisiti utente:
        - Dovrebbe descrivere i requisiti funzionali e non funzionali in modo comprensibile per gli utenti del sistema senza entrare nel dettaglio tecnico.
        - Dovrebbero specificare solo il comportamento esterno del sistema.
        - Non dovrebbero descrivere caratteristiche di progettazione.
        - Dovrebbero essere scritti in linguaggio semplice
            - Non devono usare linguaggio tecnico.
            - Non devono far riferimento all'implementazione.
        - Il linguaggio naturale può causare diversi problemi:
            - È possibile mancare di chiarezza.
            - Confusione sui tipi di requisiti.

        - Mescolanza dei requisiti.
    - Requisiti di sistema:
        - Sono versioni espanse dei requisiti utente e sono utilizzati dagli ingegneri del software come base di partenza per la progettazione.
        - Dovrebbero descrivere il comportamento del sistema e i vincoli operativi, NON come il sistema dovrebbe essere implementato o progettato.
    - Specifica delle interfacce:
        - Il sistema dovrà operare insieme a sistemi già esistenti.
            - Necessità di specificare con precisione le interfacce.
- ➔ Ha un insieme di utenti molto vario a cui deve essere comprensibile.
- ➔ Per minimizzare le incomprensioni ⇒ Accorgimenti:
    - Scrivere tutte le definizione dei requisiti usando un certo formato standard (opportunamente scelto).
        - Unificare il formato rende le omissioni meno probabili e i requisiti diventano più facili da verificare.
    - Usare il linguaggio in maniera coerente:
        - Distinguere requisiti obbligatori → Il sistema DEVE …
        - Distinguere requisiti desiderati/graditi → Il sistema DOVREBBE …
    - Usare diversi stili di testo.
    - Evitare l'uso del gergo informatico.
- ➔ Utenti del documento
    - Clienti
        - Specificano i requisiti.
        - Leggono il documento dei requisiti per verificare.
        - Modificano i requisiti.
    - Manager
        - Pianificano l'offerta.
        - Pianificano lo sviluppo.
    - Ingegnere di sistema
        - Sviluppo del sistema.
        - Testing.
        - Manutenzione.

ESEMPIO:

2.6.1 **FUNZIONALITÀ DELLA GRIGLIA**

L'editor deve fornire una griglia, dove …
MOTIVAZIONE: Una griglia aiuta l'utente nella creazione di diagrammi.
SPECIFICA: Maggiori dettagli
SORGENTE: Chi ha proposto il requisito.

Alcuni standard per la scrittura di documenti dei requisiti: IEEE/ANSI 830-1998. Stesura del documento:

1) Prefazione: informazioni di partenza (potenziali lettori, riferimenti alla versione attuale).
2) Introduzione: descrizione (in breve le funzioni, integrazione con altri sistemi e come si inserisce all'interno degli obiettivi strategici) e necessità del sistema.
3) Glossario: definizione dei termini tecnici utilizzati nel documento.
4) Definizione dei requisiti utente: descrizione dei servizi forniti agli utenti e i requisiti di sistema non funzionali.
    a) Uso del linguaggio naturale corredato da diagrammi.

5) Architettura del sistema: descrizione ad alto livello dell'architettura prevista.
    a) Suddivisione delle funzioni nei vari moduli.
6) Specifiche dei requisiti di sistema: requisiti funzionali e non funzionali (eventualmente interfacce)
7) Modelli di sistema: delimitare uno o più modelli di sistema mostrando le relazioni del sistema con l'ambiente.
    a) Modelli di oggetti.
    b) Diagrammi di flusso.
8) Evoluzione del sistema
9) Appendici
10) Indici

Processi di ingegneria dei requisiti

Attività comuni a tutti i processi:
- ➔ Deduzione dei requisiti.
- ➔ Analisi dei requisiti.
- ➔ Validazione dei requisiti.
- ➔ Gestione dei requisiti.

Deduzione e analisi dei requisiti ⇒ STAKEHOLDERS (figure diverse)
Dedurre i requisiti dagli stakeholders non è facile
- Hanno un'idea generale dei requisiti
- Esprimono i requisiti sulla base delle loro competenze.
- Interessi diversi.

Attività di deduzione e analisi dei requisiti:
1. Scoperta dei requisiti
2. Classificazione e organizzazione dei requisiti
3. Negoziazione e definizione priorità dei requisiti
4. Specifica dei requisiti

Scenari: esempi di interazioni
- ➔ Casi d'uso ⇒ Tecnica basata su scenari per la deduzione di requisiti
    - ◆ Notazione UML
- ➔ I diagrammi di sequenza ⇒ usati per completare le informazioni

Casi d'uso:
[SCHEMA QUI]

Etnografia: tecnica di osservazione
- ➔ Scoperta dei requisiti impliciti
- ➔ Esterna i processi reali e non formali
    - ◆ Come la gente lavora realmente
    - ◆ Requisiti che derivano dalla cooperazione

Convalida dei requisiti: dimostrare che i requisiti raccolti definiscono in maniera corretta il sistema che il cliente vuole

➔ Importante per evitare errori nei requisiti

Tipi di controllo

➔ Validità: descrizione corretta delle funzionalità richieste.
➔ Consistenza: tra i requisiti non dovrebbero esserci conflitti o contraddizioni.
➔ Completezza: i requisiti devono includere tutte le funzionalità e i vincoli descritti.
➔ Realismo: i requisiti possono essere implementati considerando:
  ◆ Tecnologia
  ◆ Tempo
  ◆ Budget
➔ Verificabilità: i requisiti sono verificabili?

Tecniche di validazione:

- Revisione dei requisiti: i requisiti vengono sistematicamente analizzati da un team di revisori.
- Prototipazione
- Generazione di casi di test
- Risoluzione dei problemi

Gestione dei requisiti: gestione dei cambiamenti dei requisiti

- Evoluzione nel tempo
  - Raffinando la descrizione
- Cambiamenti veri e propri
  - Valuto l'impatto

Rispetto all'evoluzione nel tempo

- Requisiti Duraturi: relativamente stabili ⇒ dominio
- Requisiti Volatili: requisiti che possono cambiare durante lo sviluppo o dopo ⇒ Politiche di governo
  - Mutabili: possono cambiare a causa della dinamicità ambientale.
  - Emergenti: emergono durante lo sviluppo.
  - Consequenziali: derivano dall'introduzione del sistema
  - Compatibilità

Politiche di tracciabilità

- Definiscono, registrano e indicano come conservare la relazione tra i requisiti.

Tool a supporto della gestione/elaborazione di grandi quantità di dati

- Tracciabilità dei requisiti
- Tracciabilità del progetto

Processi software

- ➔ Gestione del cambiamento
  - ◆ Anticipare il cambiamento
    - Sviluppo dei prototipi
      - Specifica (analisi dei requisiti)
      - Progettazione
      - Test
    - Miglioramento usabilità
    - Miglioramento rispetto alle attese dell'utente
    - Miglioramento qualità del progetto
    - Miglioramento della manutenibilità
    - Riduzione sforzo di sviluppo
  - ◆ Tollerare il cambiamento ⇒ Tecniche di sviluppo incrementale

Processo di sviluppo prototipi

- ➢ Obiettivi prototipo ⇒ Funzionalità del prototipo ⇒ Sviluppo Prototipo ⇒ Valutazione del prototipo
- ➢ I prototipi dovrebbero essere usa e getta

Sviluppo e consegna incrementali

- Sistema ⇒ Incrementi
- Requisiti utenti ⇒ con priorità
- Sviluppo incremento ⇒ requisiti congelati
- Sviluppo incrementale
  - Ogni incremento deve essere completato prima del successivo
  - Valutazione fatta dall'utente finale
- Consegna incrementale
  - Consegna utente finale (deploy)

[... schema ...]

Miglioramento del processo

- Ottimizzare risorse
- Gestire qualità software
- Maturità del processo
- Agile

[... schema ...]

Sviluppo Agile del Software

- Specifica, progettazione e implementazione sono “mischiate”
- Il software è sviluppato in versioni successive (incrementi) con il coinvolgimento dell’utente (stakeholder)
- Rilascio frequente delle versioni per valutazione
- Uso estensivo di strumenti automatici di test
- Minimalità della documentazione

[... schema ...]

- Individui e interazioni <==> Processi e strumenti
- Software funzionante <==> Documentazione completa
- Collaborazione dell’utente <==> Negoziazione di un contratto
- Risposta al cambiamento <==> Seguire un piano

Principi

- Coinvolgimento Cliente
- Consegna incrementale
- Persone non processi
- Abbracciare il cambiamento
- Mantenere la semplicità

Extreme Programming

- Nuove versioni più volte al giorno
- Incrementi consegnati al cliente ogni due settimane
- Tutti i test devono essere superati da una release prima di procedere
- Principi
    - Pianificazione incrementale
    - Rilasci piccoli
    - Progettazione semplice
    - Sviluppo anticipato dei test
    - Refactoring
    - Pair programming
    - Proprietà collettiva
    - Continua integrazione
    - Ritmo sostenibile
    - Cliente on site

[... schema ...]

Modellazione orientata agli oggetti

UML: strumenti per la definizione di diagrammi

- Activity diagram
- Sequence diagram
- Class diagram

Oggetto: unità di informazione che rappresenta un ente, concetti astratti, oggetti fisici relativi al nostro sistema.

- OID: Object ID.
- Stato: rappresentato tramite valori degli attributi.
- Comportamento: che cosa può fare l'oggetto.

Information Hiding (encapsulation): nascondere le informazioni e mostrare all'esterno solo i comportamenti.
Gli oggetti hanno un **tipo**, che è un'astrazione per descrivere stato e comportamento.

- Tipi atomici.
- Costruttori di tipo.
    - Record (Tupla)
    - Set
    - Bag
    - List
- Segnatura dei metodi.

Type Docente:
    Record (CF: string, name: string, cognome: string, studenti: set(studente))

Gli oggetti vengono raccolti (e implementati) in classi.

- Classe ⇒ Estensionale
- Tipo ⇒ Intensionale
- Oggetto

IDENTITÀ !== UGUAGLIANZA

- Identità basata su Object ID
- Uguaglianza basata sul valore dello stato dell'oggetto.
    - Superficiale: Stessi valori e ma riferimenti ad altri oggetti uguali
    - Profonda: Stessi valori e stessi valore dei riferimenti degli altri oggetti.

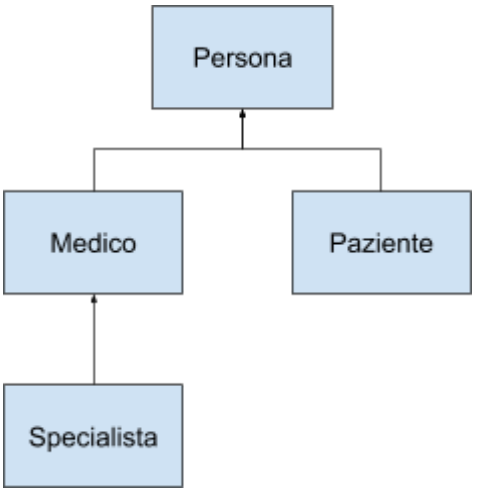
METODI

- Segnatura (nome, parametri e tipi)
- Body
- Possono essere:
    - Costruttori
    - Distruttori
    - Accessori ⇒ getter
    - Trasformatori ⇒ setter

[.. schema ..]

Gerarchie di generalizzazione

- Inheritance ⇐⇒ Ereditarietà
- Subtyping
- Aggiungo metodi e attributi
- Ridefinisco l'implementazione dei metodi e raffinamento di tipo
    - Overriding
        - Late binding
    - Overloading
    - Covarianza
        - Proprietà
        - Parametri in ingresso
        - Parametri in uscita
    - (Contro varianza)




```
CLASS MEDICO
    TYPE TUPLE(DATI_ANAGRAFICI: TUPLE(COGNOMENOME: STRING))
    METHOD SET_DATIA
END;

METHOD BODY SET_DATA IN CLASS MEDICO {(SELF-> DATI_ANAGRAFICI) = TUPLE
(COGNOMENOME: 'ROSSI')}

CLASS CARDIOLOGO INHERITS MEDICO
    TYPE TUPLE(DATI_ANAGRAFICI: TUPLE(COGNOMENOME: STRING, INDIRIZZO:
STRING))
    METHOD READ_INDIRIZZO: STRING
END;

METHOD BODY READ_INDIRIZZO:STRING IN CLASS CARDIOLOGO {
    RETURN (SELF->DATI_ANAGRAFICI).INDIRIZZO
}

//----

M = NEW CARDIOLOGO
M->SET_DATIA  // non viene settato l'attributo indirizzo
M->READ_INDIRIZZO // non è in grado di leggere l'attributo indirizzo ⇒
ERRORE!
```

```
CLASS PAZIENTE
    TYPE TUPLE (COGNOME: STRING, TERAPIA: FARMACO)
END;
CLASS PAZ_CARDIO INTHERITS PAZIENTE
    METHOD STESSA_TERAPIA(PC: PAZ_CARDIO): BOOLEAN
END;

METHOD PAZIENTI_SIMILI(P:PAZIENTE) IN CLASS PAZIENTE

METHOD BODY PAZIENTI_SIMILI(P:PAZIENTE) IN CLASS PAZIENTE {
    RETURN (P->STESSA_TERAPIA(SELF) AND ....)
}

//----

P1: PAZIENTE
PC1: PAZ_CARDIO
P1->PAZIENTI_SIMILI(PC1) // ERRORE!
```

UML (Unified Modelling Language)

Classe:

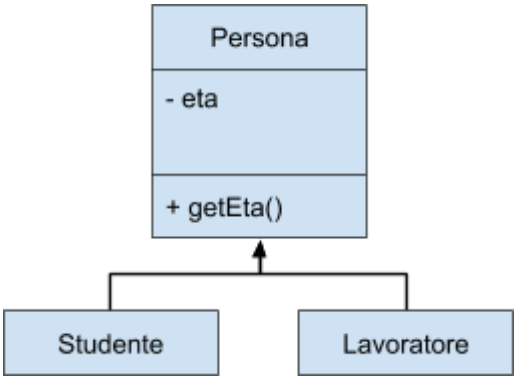
- UpperCamelCase

Attributi:

- Visibilità Nome: tipo molteplicità = valoreIniziale
- lowerCamelCase

Metodi:

- Visibilità Nome(NomeP: TipoP, NomeP TipoP, ..): tipoRestituito
- lowerCamelCase



Simboli di visibilità:

+	pubblica
-	privata
#	protected
~	package

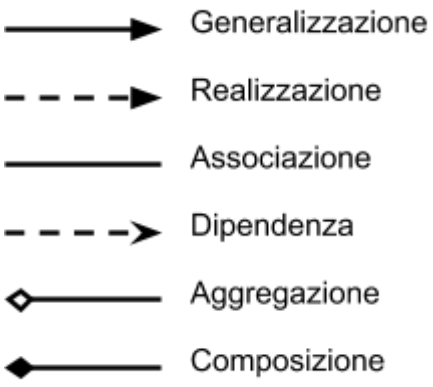
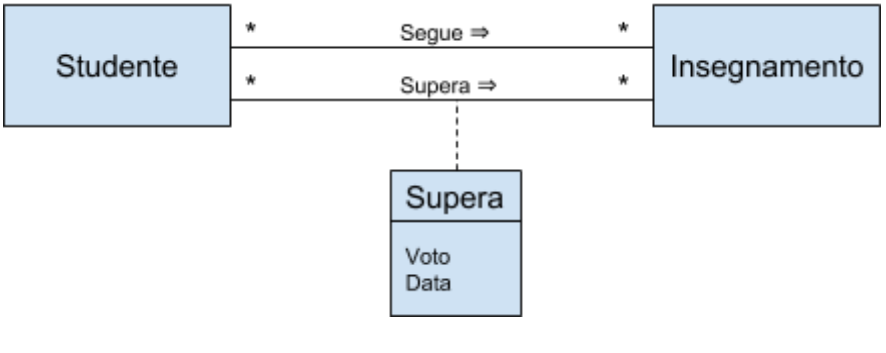


Diagramma delle classi

- Associazione
- Aggregazione e composizione

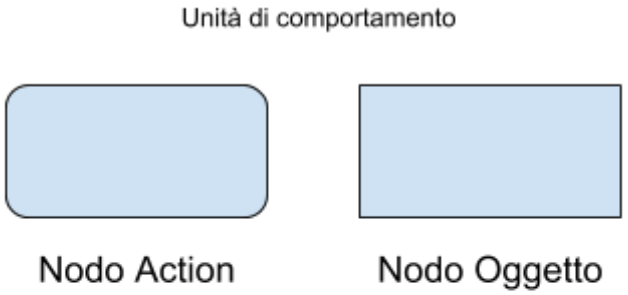


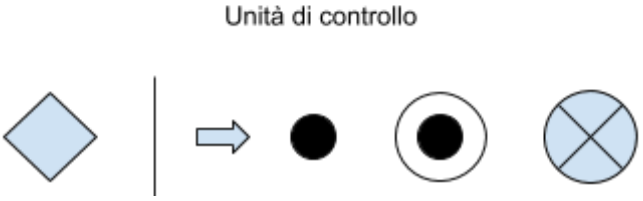
- Classi con attributi principali
- Associazione con nome e cardinalità
- Gerarchie di generalizzazione

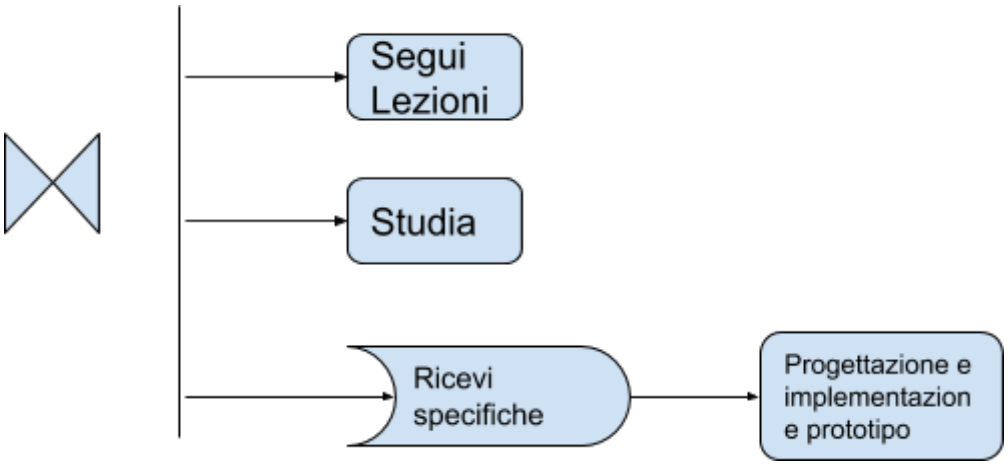
Diagramma delle Attività

- Flusso di esecuzione di un caso d'uso.
- Operazione di una classe.
- Rappresentare un algoritmo.

- Semantica a token.
- Decision ⇒ Fusion
- Fork ⇒ Join

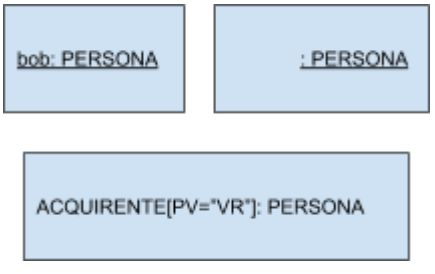


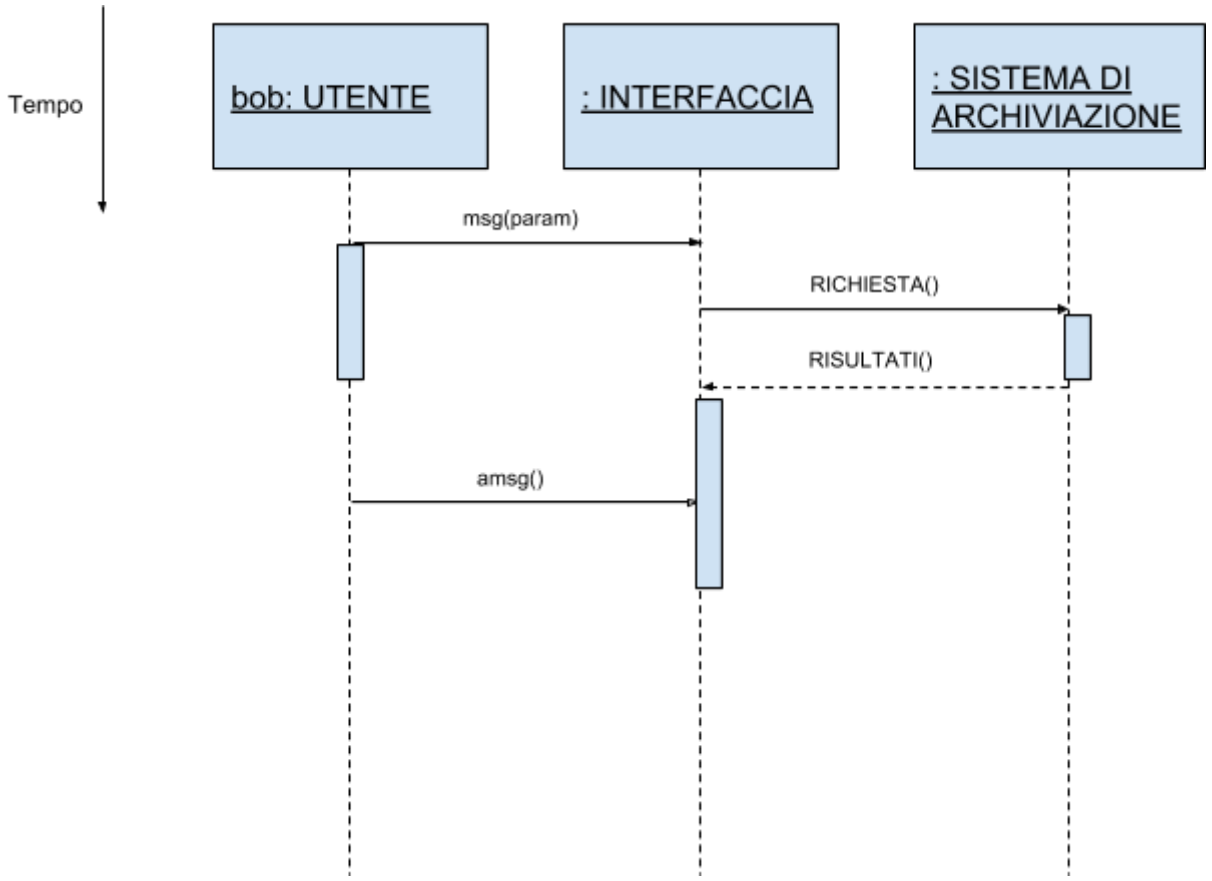




Diagrammi di sequenza

- Scambio di messaggi nel tempo fra entità-attori (istanze di classi)
  - Realizzazione di un comportamento
- Un comportamento tratto da un classificatore
  - Casi d'uso
  - Metodi di classi
- Istanze di attori, Istanze di classi, (linee di vita)





- Frecce piene ⇒ chiamate sincrone
- Frecce vuote ⇒ chiamate asincrone
- Frecce tratteggiate ⇒ ritorno di chiamata
- Creazione
- Distruzione
- Messaggio trovato (dall'esterno)
- Messaggio perso (interno)

Frammento Combinato

- Operatore
- Operandi
- Condizioni di guardia
- ALT ⇒ case/select
- OPT ⇒ if/then
- LOOP ⇒ for/while
    - BREAK ⇒ break
- REF

[...schema...]

Progettazione Architetturale

- Template?
- Distribuzione di SW e HW
- Pattern Architetturali: soluzioni che vengono riusate in diversi contesti.
- Strategie per controllare le operazioni?
- Migliore organizzazione architetturale per i requisiti non funzionalità?
- Come scomporre in sotto-componenti?

Architettura e Caratteristiche del Sistema

- Performance
- Sicurezza
- Safety
- Disponibilità
- Manutenibilità
- Viste Architetturali (+1 Vista Concettuale)
    - Logica
    - Di Sviluppo
    - Di Processo
    - Fisica

Pattern Architetturali

- “Soluzioni Predefinite”
- Buone Pratiche
- Patterns:
    - MVC: Model View Controller.
    - A strati.
    - Repository
    - Client-Server
    - Pipe and Filter: Elaborazione dati.

Architetture di applicazioni

- Applicazioni di elaborazione dei dati
- Applicazioni transazionali
    - Web-Based Information Systems
    - Sistemi di E-commerce
- Applicazioni orientate ai processi
- Applicazioni basate su eventi

Test del Software

- Mostrare che il software fa quello per cui è realizzato
- Scoprire eventuali difetti
- Dati artificiali
- Rivela la presenza di errori ma NON la loro assenza.

Validazione e verifica del software

- Scopo del software
- Aspettative dell'utente
- Contesto commerciale
- Ispezione: analisi statica del software
    - Requisiti utenti
    - Architettura software
    - Modelli UML
    - (Schema Base Dati)
    - Programmi
    -
- Test: analisi dinamica del software

Fasi di test

- Test di sviluppo
    - Test di unità
    - Test di interfaccia/architettura
    - Test di sistema
- Test del sistema rilasciato
- Test utente
    - Alpha & Beta test

Sviluppo guidato da test

- Sviluppo di codice e test ⇒ Interlacciati
- Test prima del codice
- Sviluppo incrementale
- Metodi agili ma possibili anche per plan driven
- Copertura del codice (Code Coverage)
- Regressività dei test
- Debugging semplificato
- Documentazione del software

[...schema...]

Test da utenti

- α-test
- β-test
- Accettabilità

Evoluzione del Software

- Nuovi requisiti
- Cambi nel contesto organizzativo
- Errori
- Nuovi strumenti
- Performance e affidabilità

[...schema...]
[...schema...]
[...schema...]

Legacy Systems

- Sistemi vecchi e (spesso) non documentati
- Sistema socio-tecnico

[...schema...]

Manutenzione del codice

- Modificare codice dopo la messa in funzionamento

Scrum

<slides>