UNIVERSITÀ DEGLI STUDI
DI GENOVA

Università degli Studi di Genova

# Fondamenti di Ingegneria del Software

*Lorenzo Vaccarecci*

2024-2025

# Indice

# Capitolo 1

# Modelli di processo di sviluppo software

## 1.1 Introduzione

| **Processo**: insieme strutturato e organizzato di attività che si svolgono per ottenere un risultato. |
|---|

Perchè modellare il processo? Per dare ordine, controllo e ripetibilità con l'intenzione di migliorare la produttività e la qualità del prodotto.

### 1.1.1 Processo prescrittivo e adattivo

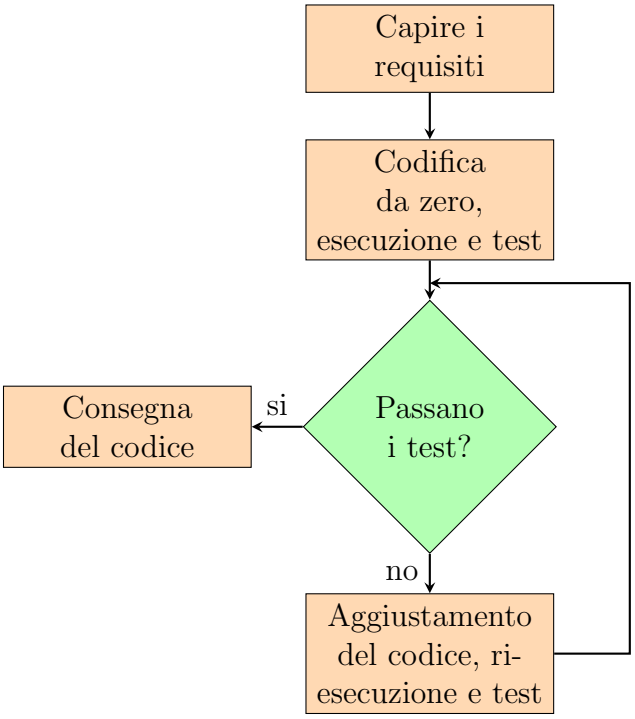
- **Processo prescrittivo**: un processo che segue un modello predefinito e rigido, con passaggi specifici e ben definiti.
- **Processo adattivo**: un processo che permette modifiche e adattamenti durante il suo svolgimento.

Perchè studiare i modelli di processo? Perchè uno dei compiti dei manager aziendali è quello di decidere il modello di processo da adottare considerando la tipologia del software da progettare e il personale disponibile.

## 1.2 Modelli di processo

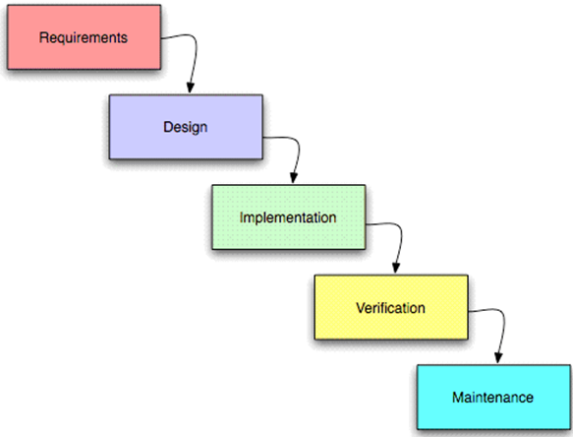
## 1.3 Code and Fix

- Si arriva al codice finale "per tentativi"
- Non adatto per progetti grandi con tanti sviluppatori
- Non è un modello di processo vero e proprio




## 1.4 Modello a cascata

- Storicamente il primo modello del processo di sviluppo software
- Ogni fase produce un prodotto che è l'input della fase successiva
- Con il modello waterfall abbiamo il passaggio dalla dimensione artigianale alla produzione industriale del software
- Molto rigido: non si può tornare indietro



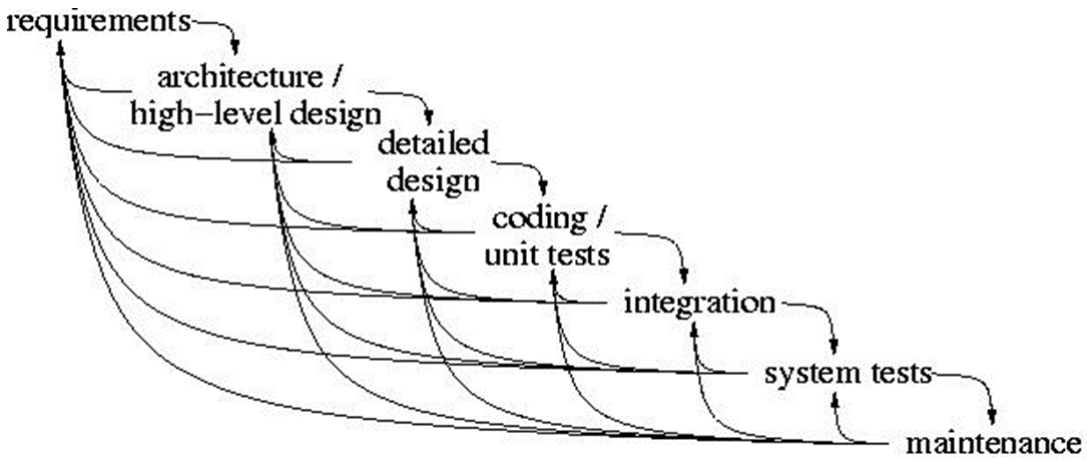

| Vantaggi | Svantaggi |
|---|---|
| Enfasi su aspetti come l'analisi dei requisiti e il progetto di sistema trascurati nell'approccio code & fix | Lineare, rigido, monolitico: no feedback tra fasi, no parallelismo, unica data di consegna |
| Pospone l'implementazione dopo avere capito i bisogni del cliente | La consegna avviene dopo anni, intanto i requisiti cambiano o si chiariscono: così viene consegnato software obsoleto |
| Introduce disciplina e pianificazione | Viene prodotta troppa documentazione poco chiara: l'utente spesso non conosce tutti i requisiti all'inizio dello sviluppo |
| E' applicabile se i requisiti sono chiari e stabili | Alcuni difetti superati da modello waterfall con feedback e iterazioni |

### 1.4.1 Studio di fattibilità

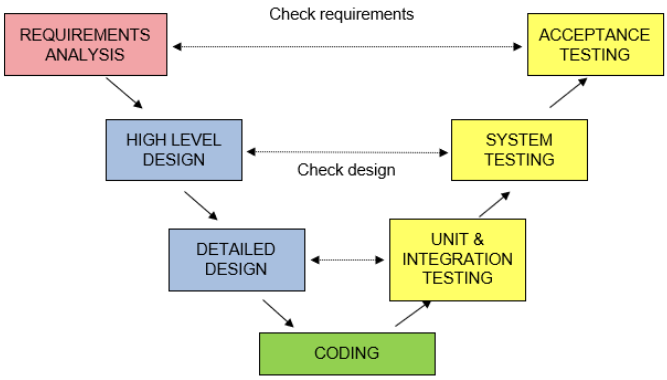
- Fase che precede lo sviluppo vero e proprio
- Viene analizzata la fattiblità e convenienza del progetto
- Stima dei costi
- Si valuta il Return Of Investment (ROI)

### 1.4.2 Varianti del modello a cascata

- Cascata con prototipazione: prima di iniziare lo sviluppo si costruisce un prototipo "usa e getta" con il solo scopo di fornire agli utenti una base concreta per meglio definire i requisiti.
- Cascata con feedback e iterazioni: posso tornare a una fase precedente.




- V-Model:
  - Enfasi sulle fasi di testing
  - Evidenzia come le attività di testing (parte destra della V) sono collegate a quelle di analisi e progettazione (parte sinistra della V)
  - Ogni controllo fatto a destra che non dia buon esito porta a un rifacimento/modifica di quanto fatto a sinistra
  - **Parallelismo**: creazione dei test e una volta che ho il codice li eseguo
  - Problemi (anche per Waterfall):
    * Versione funzionante solo alla fine!
    * Errore in fase iniziale può avere conseguenze disastrose



## 1.5 Modelli evolutivi

| |
|---|
| Idea: sviluppare un implementazione iniziale, esporla agli utenti e raffinarla attraverso successivi rilasci del SW (release) |

Sottocategorie:

- Prototyping
- Modelli incrementali
- Modelli iterativi

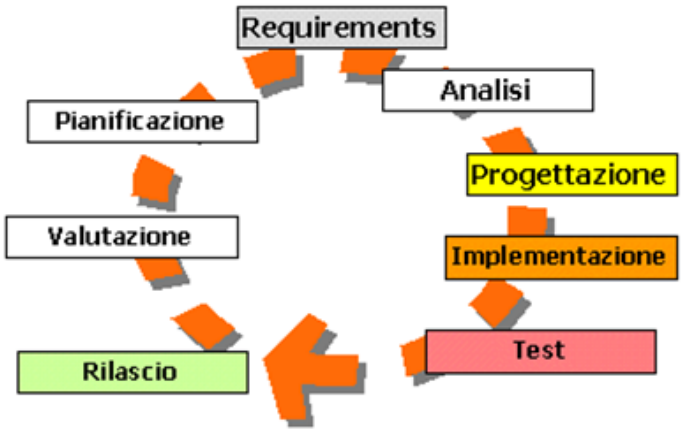
### 1.5.1 Modelli a Prototyping

- Realizzazione di un prototipo funzionante del sistema, su cui validare i requisiti (o l'architettura)
- Il prototipo ha meno funzionalità ed è meno efficiente

| Vantaggi | Svantaggi |
|---|---|
| Permette di raffinare requisiti definiti in termini di obiettivi generali e troppo vaghi | Il prototipo è un meccanismo per identificare i requisiti, spesso da "buttare": problema economico e psicologico, il rischio è di non farlo e così scelte non ideali diventano parte integrante del sistema |
| Rilevazione precoce di errori di interpretazione | |

### 1.5.2 Modelli Iterativi-Incrementali

- Sviluppo di varie release, di cui solo l'ultima è completa
- Dopo la prima release, si procede in parallelo
- Le fasi di sviluppo vengono percorse più volte




#### Modelli Incrementali

- Ogni release aggiunge nuove funzionalità
- Nella fase di pianificazione si decide il requisito/funzionalità da includere nella release successiva.
- Si trattano per prime le funzionalità ad alto rischio
- Si cerca di massimizzare il valore per gli utenti

**Modelli Iterativi**

- Da subito sono presenti tutte (o buona parte) delle funzionalità che sono via via raffinate, migliorate

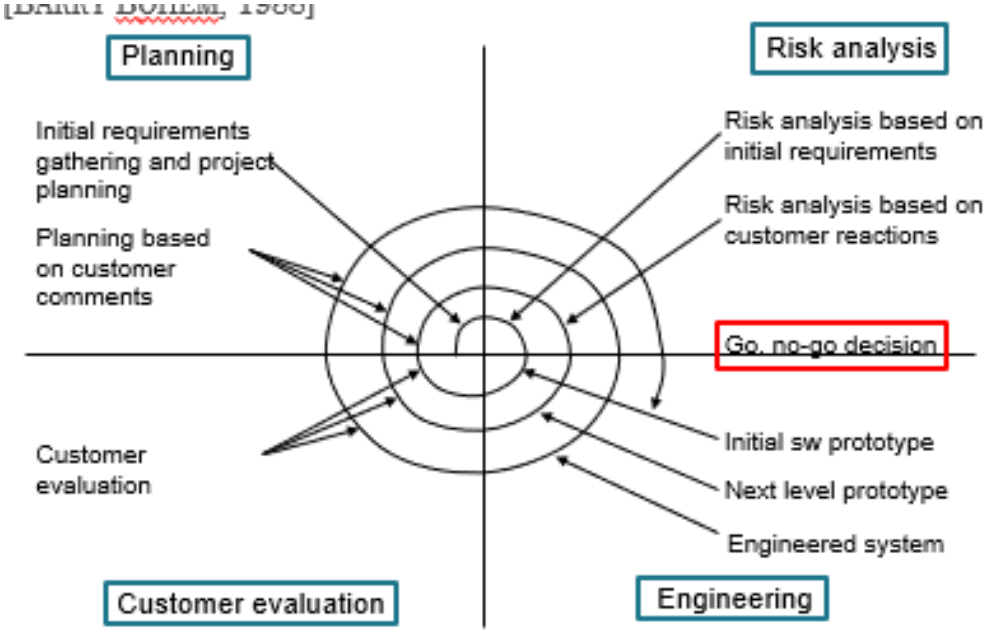
## 1.6 Modello a spirale

- Sistemi di grandi dimensioni
- Approccio "evolutivo" con interazioni continue fra cliente e developer
- Modello "risk-driver": tutte le scelte sono basate sui risultati dell'analisi dei rischi
- 'Meta-modello': dà un'idea generale ma quando si inizia a lavorare bisogna scegliere un modello esistente
  - Requisiti chiari e stabili → modello a cascata
  - Requisiti confusi → prototipo

**Rischio**: circostanza potenzialmente avversa in grado di pregiudicare lo sviluppo e la qualità del software

Ogni scelta/decisione ha un rischio associato, due caratteristiche importanti nella valutazione di un rischio sono:

- Gravità delle conseguenze
- Probabilità che si verifichi la circostanza




- **Planning**: determinazione di obbiettivi, alternative, vincoli
- **Risk Analysis**: analisi delle alternative e identificazione/risoluzione dei rischi
- **Engineering**: sviluppo del prodotto di successivo livello
- **Customer Evaluation**: valutazione dei risultati dell'engineering dal punto di vista del cliente

| Vantaggi | Svantaggi |
|---|---|
| Adatto allo sviluppo di sistemi complessi | Non è un rimedio universale (panacea) |
| Primo approccio che considera il rischio (risk-driver) | Necessita competenze di alto livello per la stima dei rischi |
| | Richiede un'opportuna personalizzazione ed esperienza di utilizzo |
| | Se un rischio rilevante non viene scoperto o tenuto a bada si inizia da zero |

## 1.7 Unified Process

- Specifico per sistemi ad oggetti, con uso di notazione UML per tutto il processo
- Guidato dagli **Use Case**
- Incorpora molte delle idee 'buone' dal modello a spirale
- Meta-modello
- Supportato da tool(visuali) in ogni fase
- Processo prescrittivo per eccellenza

### 1.7.1 Le iterazioni

- Possibili diverse iterazioni che terminano con il rilascio del prodotto
- Ogni iterazione consiste di quattro fasi (anche ripetute più volte) che terminano con una milestone (= rilascio di artefatti soggetti a controllo)
- Ogni fase è costituita da diverse attività:
  - Requisiti (R)
  - Analisi (A)
  - Design (D)
  - Codifica (C)
  - Testing (T)

### 1.7.2 Le fasi

- Inception: studio di fattibilità, requisiti essenziali del sistema, risk analysis
- Elaboration: sviluppa la comprensione del dominio e del problema, gli Use Case della release da rilasciare, l'architettura del sistema
- Construction: Design (in UML), codifica e testing del Sistema
- Transition: Messa in esercizio della release nel suo ambiente (deploy), training e testing da parte di utenti fidati

## 1.8 Sviluppo basato sui componenti

Modello che va nella direzione del **riutilizzo del software**

| Vantaggi | Svantaggi |
|---|---|
| Riduce la quantità di software da scrivere | Sono necessari dei compromessi: requisiti iniziali potrebbero differire da quelli che si possono soddisfare con le componenti disponibili |
| Riduce i costi totali di sviluppo e i rischi | Integrazione non sempre facile |
| Consegne più veloci | Spesso i componenti usati sono fatti evolvere dalla ditta produttrice senza controllo di chi li usa |

## 1.9 Metodi Plan-Driven e Agili

| Plan-Driven | Agile |
|---|---|
| Seguono un approccio classico dell'ingegneria dei sistemi fondato su processi ben definiti e ocn passi standard | Rispondere ai cambiamenti dei requisiti in modo veloce |
| | Filosofia del programmare come "arte" piuttosto che processo industriale |
| | Cosa più importante soddisfare il cliente e non seguire un piano (contratto) |

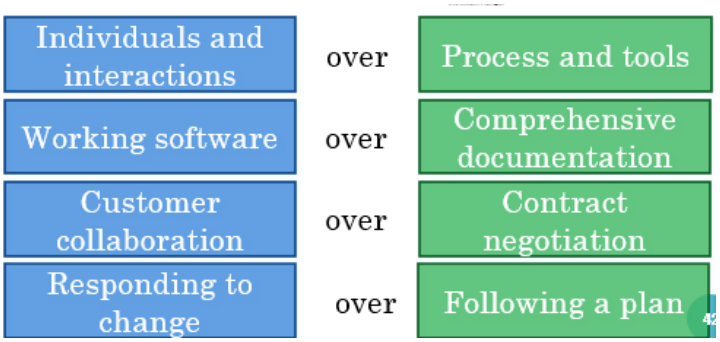



Figura 1.1: The Agile Manifesto

### 1.9.1 Come scegliere?

**Metodi plan-driven**:

- Sistemi grandi e comploessi, safety-critical o con forti richieste di affidabilità
- Requisiti stabili e ambiente predicibile

**Metodi agili**:

- Sistemi e team piccoli, clienti e utenti disponibili, ambiente e requisiti volatili
- Team con molta esperienza
- Tempi di consegna rapidi

## 1.10 DevOps

Metodo di sviluppo evolutivo

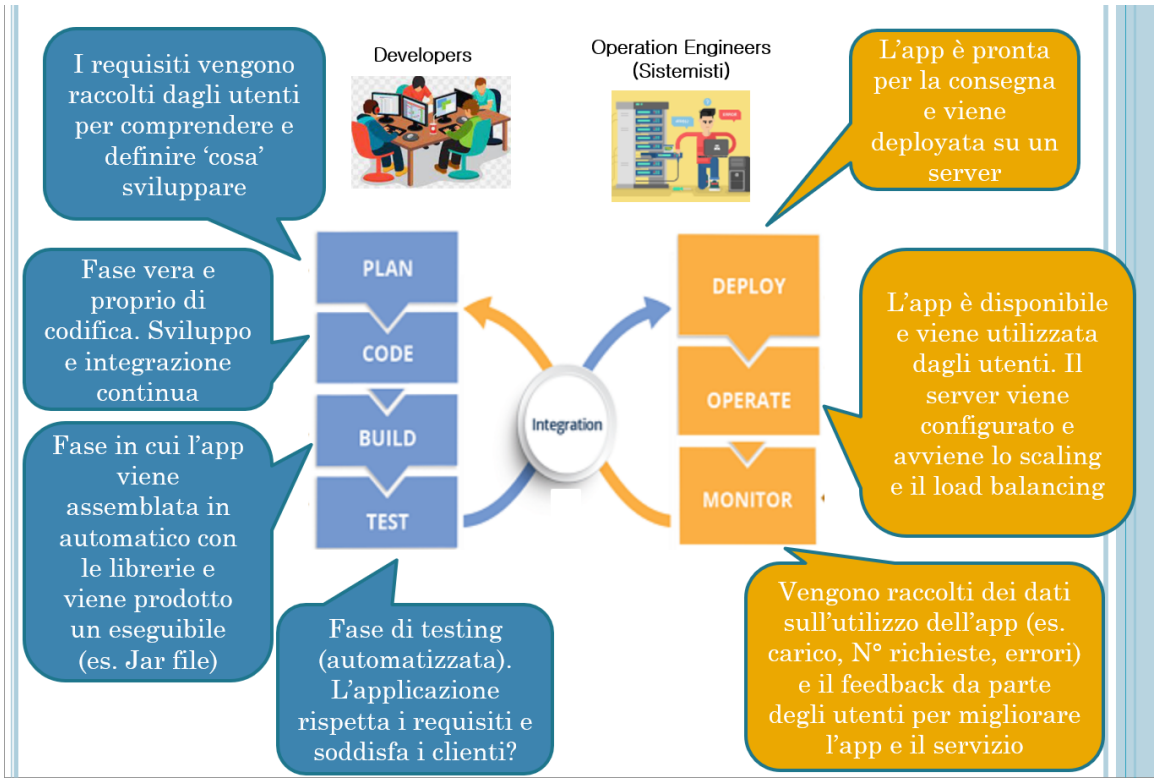



Figura 1.2: DevOps

### 1.10.1 Continuous Integration

La Continuos Integration (CI), o Integrazione Continua, è una pratica di sviluppo software in cui i programmatori integrano frequentemente il proprio lavoro (codice) nel repository condiviso del progetto, in genere diverse volte al giorno.

# Capitolo 2

# Ingegneria dei requisiti

## 2.1 Introduzione

Descrivere 'qualcosa' che il sistema dovrà fare (una funzionalità) o un vincolo a cui deve sottostare

- **Diversi livelli di astrazione**:
  - Descrizione astratta ed imprecisa del sistema
  - Descrizione dettagliata e matematica dello stesso

| Che cosa il sistema farà e non come! |
|---|

E' importante definire i requisiti in modo da evitare difetti in fasi avanzate del progetto, infatti i difetti dovrebbero essere scoperti il più presto possibile, ovvero a livello dei requisiti.

## 2.2 Classificazione dei requisiti

- **Requisiti utente**: descrizione in linguaggio naturale delle funzionalità che il sistema dovrà fornire e dei vincoli operativi (sono scritti per (e con) il cliente)
- **Requisiti di sistema**: descrive in modo dettagliato le funzionalità che il sistema dovrà fornire (sono scritti per gli sviluppatori)
- **Requisiti funzionali**: descrivono ciò che il sistema dovrà fare, non come ma cosa
- **Requisiti non-funzionali**: definiscono vincoli sul sistema e sullo sviluppo del sistema, in generale riguardano la scelta di linguaggi, piattaaforme, strumenti, tecniche d'implementazione, ma anche: prestazioni, questioni etiche, ...

Un requisito etico può essere ad esempio che nella realizzazione dell'applicazione verranno utilizzato solo strumenti e servizi 'non proprietari' (es. no Microsoft)

### 2.2.1 Esempio: Bancomat

In rosso i requisiti funzionali, in blu i requisiti non funzionali

- Il sistema deve mettere a disposizione le funzioni di prelievo, saldo e estratto conto
- Il sistema deve essere disponibile a persone portatori di Handicap, deve garantire un tempo di risposta inferiore al minuto, e deve essere sviluppato su architettura X86 con sistema operativo compatibile con quello della Banca

- Le operazioni di prelievo devono richiedere autenticazione tramite un codice segreto memorizzato sulla carta
- Il sistema deve essere facilmente espandibile, e adattabile alle future esigenze bancare

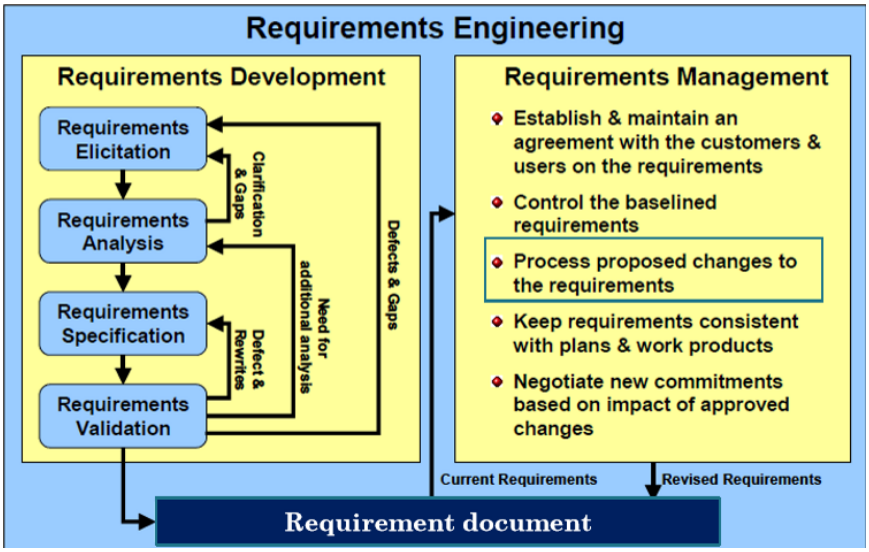
## 2.3 Requirements Engineering

E' il termine usato per descrivere le attività necessarie per raccogliere, documentare e tenere aggiornato l'insieme dei requisiti di un sistema software.

### 2.3.1 Scopo

Lo scopo primario del RE è la produzione di un documento (il requirement document) che definisca le funzionalità e i servizi offerti dal sistema da realizzare (anche tenerlo aggiornato)

### 2.3.2 Processo iterativo




- **Elicitation**:
  - Ottenere, estrarre, ricavare, tirar fuori i requisiti dal cliente e da altri partecipanti
  - Il primo passo è identificare gli stakeholders[1]
  - Intervise, osservazioni sul luogo di lavoro, questionari, analisi dei prodotti dei competitors, workshop (brainstorming)
  - Studio/analisi di leggi e regolamenti, help-desk reports, 'change requests' di prodotti analoghi, 'lessons learned' in progetti simili, ...
- **Analisi dei requisiti**:
  - I bisogni (user needs) degli stakeholders raccolti durante la fase di elicitation sono analizzati e raffinati
  - Si cerca di capire se i requisiti sono corretti
  - Si cercano di identificare i "missing requirements"
  - Si identificano requisiti poco chiari
  - Si risolvono i requisiti "contradditori o in conflitto"

[1]Stakeholder: persona veramente interessata allo sviluppo del progetto

  - Viene stabilità la priorità (prioritizzazione):
    * Per sapere cosa "tagliare" se non tutti potranno essere realizzati
    * Scala numerica
    * Scala MoSCoW:
      · **Must have**: requisiti obbligatori
      · **Should have**: requisiti importanti ma non indispensabili
      · **Could have**: requisiti desiderabili ma non necessari

- **Definizione e specifica**:
  - Definizione dei requisiti utente: costituisce un contratto fra le parti
  - Specifica dei requisiti di sistema: costituisce "starting point" per la fase di design
- **Validazione**:
  - Esame della definizione/specifica dei requisiti per valutarne la qualità
  - Di solito la convalida o validazione si effettua mediante 'formal peer reviewes'
  - Scrivere dei casi di test a partire dai requisiti
  - Sviluppare un prototipo
- **Requirements Management**:
  - Approvazione di alcune richieste di cambio dei requisiti
  - Negoziazione con il cliente
  - Impact analysis per i cambi richiesti
  - Tenere allineati i requisiti e il codice (e casi di test)
  - Tracciare il progresso di un progetto

## 2.4 Proprietà dei requisiti

- **Validità-correttezza**
- **Consistenza**: non ci sono requisiti contradditori
- **Completezza**: tutti gli aspetti che il cliente vuole sono coperti nei requisiti (in teoria)
- **Realismo**: non si chiede l'impossibile
- **Inequivocabilità (Unambiguos)**: ogni requisito dovrebbe avere solo un interpretazione
- **Verificabilità**: i requisiti vanno espressi in modo che siano testabili
- **Tracciabilità**:
  - Ogni funzionalità implementata nel sistema deve poter essere fatta risalire a dei requisiti in modo semplice
  - Ogni requisito nella requirement specification deve corrispondere ad uno nella requirement definition

## 2.5 Template/Schema dei requisiti

Conviene attenersi a questo Schema

`<id>` il `<sistema>` deve `<funzione>`

Es. `R1.`Il sistema deve gestire tutti i regitratori di cassa del negozio (non più di 20)

## 2.6 Analista software

L'analista software o di sistema è la persona che:

- si occupa dell'elicitazione dei requisiti
- analizza i requisiti
- scrive il documento dei requisiti (definizione e/o specifica)
- Comunica/spiega i requisiti a sviluppatori e altri stakeholder

Alcune competenze che un analista dovrebbe avere:

- Arte della negoziazione
- Stabilire una strategia (problem solving)
- Giusta capacità di imporsi
- Ascoltare attentantemente
- Dono della sintesi
- Padronanza del linguaggio naturale
- Buona conoscenza del dominio (ad esempio in ambito medico o automobilistico)

### 2.6.1 Consigli per un'intervista

1. Fare molte domande
2. Ascoltare bene
3. Mettere in discussione i quantificatori universali: 'tutto, ogni, sempre, ...'
4. Annotare tutte le risposte

### 2.6.2 Importanza della comunicazione

- Elicitation = Attività molto delicata perchè mette in comunicazione due o più persone di realtà anche molto diverse
- Frequenti incomprensioni, che si ripercuotono sulla qualità dei requisiti

Occore fare molta attenzione a:

- Diversità di significato che si attribuisce ai termini → possibile soluzione definizione del glossario:
  - Per la spiegazione dei termini tecnici
  - Per ridurre l'ambiguità dei termini usati
  - Per "espandere" gli acronimi
- Assunzioni nascoste (Hidden assumptions)
- Verbosità (= sovrabbondanza di parole)
- Mancanza di chiarezza/precisione

## 2.7 Consigli finali

- Riuso di (parte di) requisiti
- Utilizzo di un glossario comune tra clienti, utenti e analisti
- Utilizzo di un 'buon' template/form
- Utilizzo di un software per la geitone/raccolta e analisi dei requisiti