

Ingegneria del Software

Angelo Passarelli

September 25, 2023



Appunti basati sulle lezioni e dispense della professoressa Laura Semini ¹

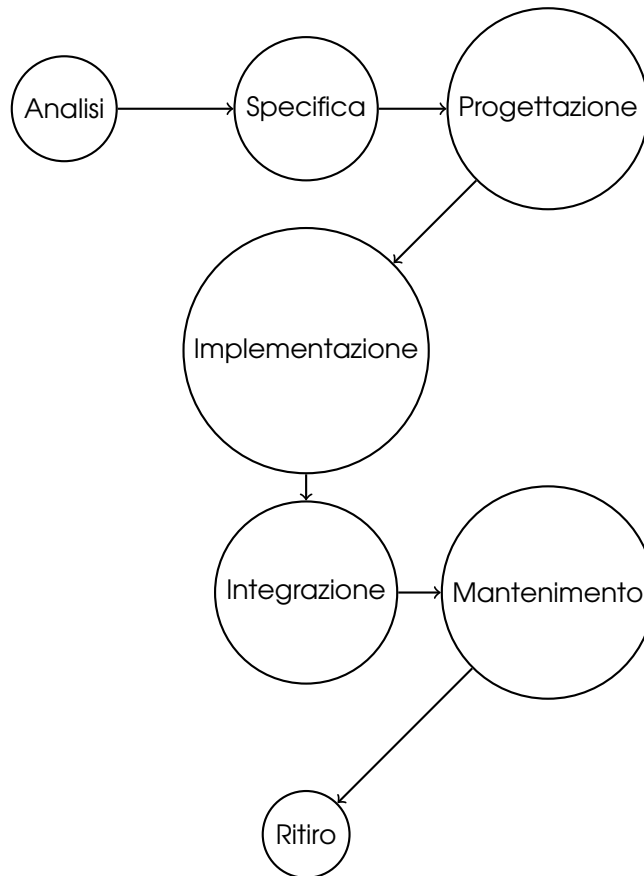
¹<http://didawiki.cli.di.unipi.it/doku.php/informatica/is-a/start>

Contenuti

| | | |
|----------|---|-----------|
| 0 | Introduzione | 3 |
| 1 | Modelli di Ciclo di Vita | 4 |
| 1.1 | Modelli Sequenziali | 4 |
| 1.1.1 | Build-and-Fix | 4 |
| 1.1.2 | Modello a Cascata | 5 |
| 1.1.3 | Modello a V | 5 |
| 1.2 | Modelli Iterativi | 6 |
| 1.2.1 | Rapid Prototyping | 6 |
| 1.2.2 | Modello Incrementale | 6 |
| 1.2.3 | Modello a Spirale | 8 |
| 1.3 | Unified Process | 9 |
| 1.4 | Processi Agili | 9 |
| 1.4.1 | Il Manifesto di Snowbird | 10 |
| 1.4.2 | eXtreme Programming | 10 |
| 1.4.3 | SCRUM | 11 |
| 2 | Analisi dei Requisiti | 13 |
| 2.1 | Studio di Fattibilità | 13 |
| 2.2 | Dominio | 13 |
| 2.3 | Requisiti | 13 |
| 2.4 | Documento dei Requisiti | 14 |
| 2.5 | Fasi dell'Analisi dei Requisiti | 14 |
| 2.5.1 | Acquisizione | 14 |
| 2.5.2 | Elaborazione | 14 |
| 2.5.3 | Convalida | 15 |
| 2.5.4 | Negoziiazione | 16 |
| 2.5.5 | Gestione | 16 |
| 2.6 | Casi d'uso | 17 |
| 2.7 | User Stories | 17 |

0 Introduzione

Fasi del progetto



Specificità del Software

1. **Fault tolerance**: capacità del software di essere tollerante ai guasti.
2. **Difetto latente**: difetto nascosto che si trova difficilmente in fase di testing; e anche nel caso comparisse è quasi impossibile da ritrovare.
3. **Robustezza**: capacità di funzionare anche con input non previsti e/o non testati.
4. Il software non presenta **costi materiali** e nemmeno **costi marginali**, ovvero il costo di un'unità del prodotto.

5. Infine il software non si consuma nel tempo, ma potrebbe diventare **obsoleto**.

La Manutenzione

Costi La fase di manutenzione è quella che richiede costi più alti. Per evitare uno spreco durante questa fase è necessario studiare bene l'analisi dei requisiti, in quanto un errore in questa fase si propagherà in modo esponenziale, in termini di costi, nelle fasi successive.

La manutenzione si divide in:

- **Manutenzione Correttiva**: rimuove gli errori, lasciando invariata la specifica.
- **Manutenzione Migliorativa**: consiste nel cambiare quella che è la specifica, e a sua volta può dividersi in:
 - **Perfettiva**: modifiche per migliorare e/o introdurre nuove funzionalità.
 - **Adattiva**: modifiche indotta da cambiamenti esterni, come leggi o modifiche all'hardware o al sistema operativo.

Stakeholders

- **Fornitore**: colui che sviluppa il software.
- **Committente**: chi lo richiede e paga.
- **Utente**: chi lo usa.

1 Modelli di Ciclo di Vita

Definizione (Processo Software). Con processo software si indica il percorso da seguire per sviluppare un prodotto o più nello specifico un software. Fanno parte del processo sia gli strumenti e le tecniche per lo sviluppo che i professionisti coinvolti.

1.1 Modelli Sequenziali

1.1.1 Build-and-Fix

Il prodotto è sviluppato senza alcuna fase di progettazione preliminare, lo sviluppatore scrive il software e poi lo modifica ogni volta che non soddisfa il committente.

Contro Diventa improponibile per progetti grandi e la manutenzione diventa difficile senza documentazione nè specifica.

1.1.2 Modello a Cascata

Questo modello è stato il primo a distinguere il processo software in più fasi, evidenziando l'importanza della progettazione e dell'analisi.

Viene chiamato anche modello *document driven* dato che ogni fase produce un documento, e per passare alla successiva occorre aver approvato il documento della fase precedente.

Contro Troppo pesante da seguire, inoltre non si può tornare indietro, e mancando l'interazione con il cliente, se non è soddisfatto, va tutto ripetuto dall'inizio.

1.1.3 Modello a V

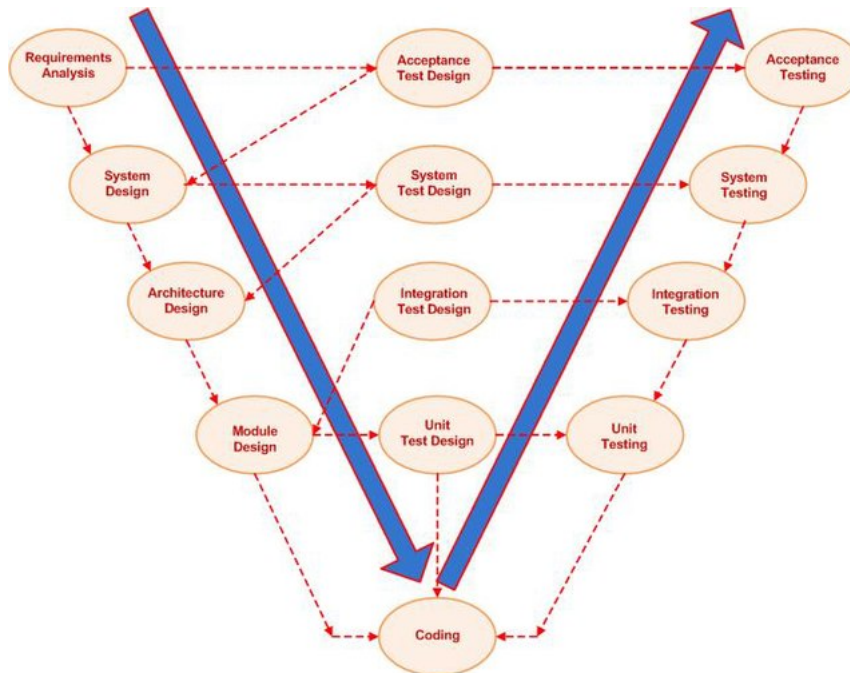


Figure 1: Le frecce blu rappresentano il *tempo*, mentre quelle tratteggiate le *dipendenze*

Questo modello evidenzia come sia possibile progettare i **test** durante le fasi di sviluppo (quelle a sinistra, prima della fase di *coding*). Mentre sulla destra sono presenti i test veri e propri che devono verificare e convalidare l'attività in corrispondenza sulla sinistra.

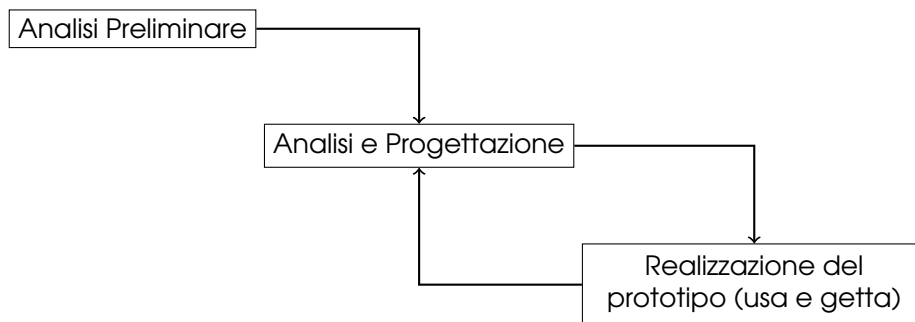
Standard SQA Questo modello è uno degli standard SQA (Software Quality Assurance), usato per descrivere le attività di test durante il processo di sviluppo.

1.2 Modelli Iterativi

1.2.1 Rapid Prototyping

L'obiettivo è quello di costruire rapidamente un prototipo del software per permettere al committente di sperimentarlo.

Questo modello diventa utile quando i requisiti non sono chiari, quindi ogni prototipo aiuterà il cliente a descriverli meglio.



1.2.2 Modello Incrementale

Il software viene costruito in modo iterativo, aggiungendo di volta in volta nuove funzionalità.

I requisiti e la progettazione vengono definiti inizialmente, per questo è possibile applicarlo solo in caso di requisiti stabili.

Contro Se non viene realizzata una buona progettazione, questo modello sfocia in un *Build-and-Fix*.

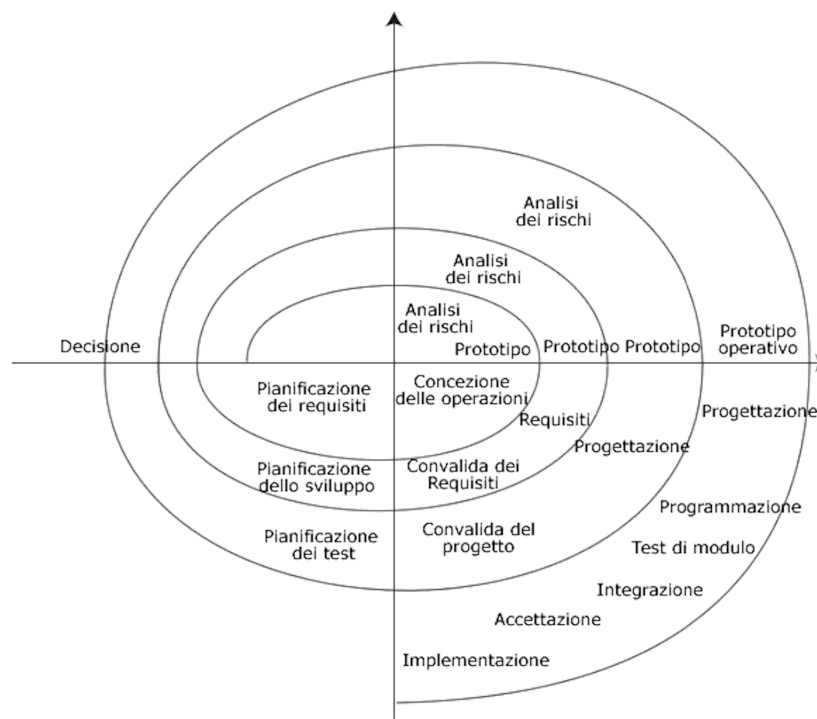


1.2.3 Modello a Spirale

In questo caso ogni iterazione è formata da 4 fasi che corrispondono ai quadranti del piano:

1. *Quadrante in alto a sinistra*: definizione degli obiettivi e dei vincoli.
2. *Quadrante in alto a destra*: analisi e risoluzione dei rischi.
3. *Quadrante in basso a destra*: sviluppo e verifica del prossimo livello.
4. *Quadrante in basso a sinistra*: pianificazione della fase successiva.

Questo modello viene anche chiamato *risk driven* in quanto è incentrato principalmente sull'analisi dei rischi. Inoltre si ispira profondamente al metodo iterativo *plan-do-check-act cycle*²



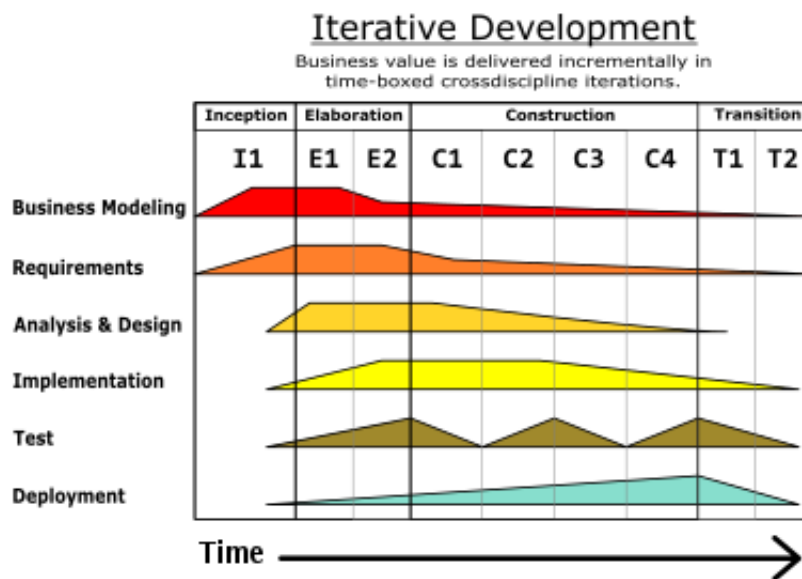
²https://it.wikipedia.org/wiki/Ciclo_di_Deming

1.3 Unified Process

In questo modello vengono distinte quattro fasi chiamate *Inception*, *Elaboration*, *Construction* e *Transition*. Ogni fase può presentare un numero variabile di iterazioni anche in base alla dimensione del progetto.

Questo modello viene definito *iterativo incrementale*, *incrementale* perchè alla fine di ogni iterazione si ottiene un rilascio del sistema con funzionalità in più o migliorate rispetto al rilascio precedente.

Inoltre viene data molta importanza all'architettura del sistema, infatti già dalle prime fasi ci si concentra soprattutto sull'architettura anche se a livello molto superficiale, lasciando i dettagli alle fasi successive. In questo modo è molto facile avere una visione generale del sistema che sarà facilmente modellabile sulla variazione dei requisiti. Per questo, piuttosto che dai requisiti, ci si fa guidare principalmente dai *casi d'uso* e dall'*analisi dei rischi*.



1.4 Processi Agili

Definizione (Metodo Agile). Con *metodo agile* si intende un metodo per lo sviluppo del software che si basa principalmente sul coinvolgimento del committente. Questa metodologia si riferisce ai principi del *Manifesto di Snowbird* del 2001.

I concetti chiave di questi processi sono:

- **Continuous Integration**: rendere il più automatico possibile la consegna e l'integrazione dei singoli moduli.
- **Continuous Delivery**: rilascio frequente e supportato delle nuove versioni del software.
- **DevOps**: *Development e Operations*, ovvero maggiore collaborazione tra sviluppatori e responsabili della manutenzione, della sicurezza e dell'infrastruttura dell'azienda.

1.4.1 Il Manifesto di Snowbird

Il *Manifesto di Snowbird* si fonda su quattro punti fondamentali:

1. **Comunicazione**: la comunicazione fra tutti gli attori del progetto è centrale, soprattutto le interazioni e la collaborazione con i clienti.
2. **Semplicità**: si mantiene il codice sorgente il più semplice possibile, ma comunque avanzato tecnicamente, in questo modo si riduce la documentazione al minimo indispensabile.
3. **Feedback**: sin dal primo giorno di sviluppo il codice viene testato, in modo da poter rilasciare versioni ad intervalli molto frequenti.
4. **Coraggio**: dare in uso il sistema il prima possibile ed implementare i cambiamenti richiesti man mano.

Di seguito sono riportati due modelli che si basano sui *processi agili*.

1.4.2 eXtreme Programming

Si basa su un insieme di consuetudini:

- *Pianificazione flessibile*: è basata su un insieme di scenari proposti dagli utenti e i programmatori vengono coinvolti direttamente.
- *Rilasci frequenti*: più o meno ogni 2-4 settimane, e alla fine si ricomincia con una nuova pianificazione.
- *Progetti semplici*: comprensibili a tutti.
- *Testing*: test basati sui singoli scenari e con supporto automatico.
- *Test Driven Development*: i casi di test vengono definiti prima della scrittura del codice.
- *Cliente sempre a disposizione*
- *Programmazione a coppie*: viene usato un solo terminale, una persona svolge il ruolo di *driver* che scrive il codice, mentre un'altra fa il *navigatore*, ovvero controlla il lavoro del *driver* attivamente.

- *No al lavoro straordinario*
- *Collettivizzazione del codice*: accesso libero e continua integrazione.
- *Code Refactoring*: modificare il codice senza cambiare il suo comportamento e commentarlo il più possibile.
- *Daily Stand Up Meeting*

1.4.3 SCRUM

Definizione (SCRUM). Con *SCRUM* si intende un processo *iterativo* ed *incrementale*, dove alla fine di ogni iterazione vengono rilasciate un insieme di funzionalità potenzialmente rilasciabili.

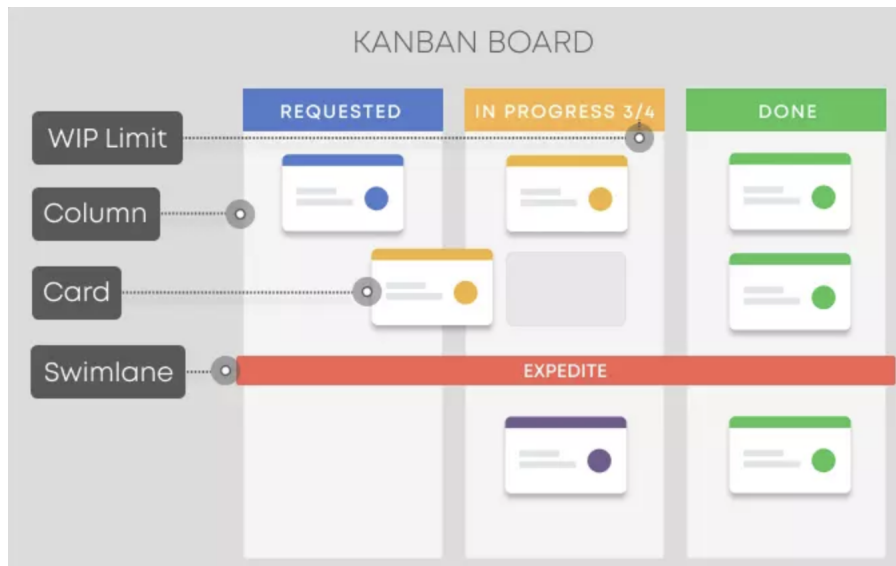
Il processo è diviso in tre fasi:

1. **Pre-game phase**:
 - (a) **Planning sub-phase**: viene creata una *Product Backlog List* che contiene tutti i requisiti conosciuti.
 - (b) **Architecture sub-phase**: viene già pianificato il design di alto livello e l'architettura del sistema.
2. **Development phase**: in questa fase il sistema viene sviluppato attraverso una serie di *Sprint*, ovvero cicli iterativi nei quali vengono sviluppate o migliorate una serie di funzionalità, e ogni sprint può durare circa 1-4 settimane. Lo *Sprint* ovviamente include le classiche fasi di sviluppo del software.
3. **Post-game phase**: il prodotto viene preparato per il rilascio, ovvero si prepara l'*integrazione*, i *test*, la *documentazione* per l'utente e la preparazione del materiale di *marketing*.

I ruoli principali durante l'esecuzione di un processo *SCRUM* sono tre:

- **Product Owner**: ci si riferisce a quella persona responsabile di accettare o rifiutare i risultati di un lavoro e di poter terminare uno *Sprint*, inoltre fa da raccordo fra tutti soggetti interessati nel progetto.
- **Membri del Team**: i membri decidono cosa fare in ogni *Sprint*, ogni team è indipendente e i membri non fanno capo ad alcun project manager. Ogni membro ha diverse specializzazioni (*cross-functional*), in modo tale da non avere persone con troppo carico di lavoro e ognuno si occupa di un singolo lavoro alla volta.
- **Scrum Master**: non ha alcuna autorità sul team, ma si occupa di supportarlo e motivarlo, garantendo anche le condizioni ambientali per lavorare al meglio.

Kanban Board Questa lavagna permette di gestire al meglio il flusso del lavoro. Come mostrato in figura è presente un *Work In Progress Limit* che definisce un limite alla quantità di post-it che possono essere presenti in ogni colonna. Questo limite permette di completare più velocemente i singoli lavori, in modo tale di dare qualcosa al cliente il prima possibile e di individuare facilmente i *colli di bottiglia* che possono rallentare gli altri lavori. Inoltre permette di ridurre il *task switching*, ovvero il lavoro su più task contemporaneamente.



Gli eventi che fanno parte di uno *Sprint* sono i seguenti:

1. **Sprint planning**: il *product owner* gestisce l'evento di pianificazione dello *Sprint*.
2. **Daily meeting**: i membri del team e gli SCRUM master si ritrovano davanti la *kanban* e discutono delle difficoltà che hanno riscontrato.
3. **Review**: alla fine di una modifica concreta al software, questo viene ispezionato in collaborazione con gli utenti per ottenere un feedback e per discutere su cambiamenti o nuove idee.
4. **Retrospettiva**: questa fase permette di riflettere, studiare e adattarsi per lo *Sprint* successivo.

2 Analisi dei Requisiti

Definizione (Analisi dei Requisiti). Si intende il processo di studio e analisi delle esigenze del committente e dell'utente per giungere alla produzione di un documento che definisce il *dominio* del problema e i *requisiti* del software. In alcuni casi si definiscono anche i *casi di test* e il *manuale utente*.

Prima di passare alla fase vera e propria di *analisi dei requisiti* occorre seguire una fase preliminare per stabilire la realizzabilità del progetto software.

2.1 Studio di Fattibilità

Si basa principalmente sulla descrizione del software e delle necessità dell'utente. In seguito vengono svolte due analisi:

- **Analisi di Mercato:** si fa un confronto con il mercato attuale e si stimano i costi di produzione e quanto l'investimento può essere remunerativo.
- **Analisi Tecnica:** si studiano tutti gli strumenti per la realizzazione del progetto, come i software, le architetture, gli hardware e gli algoritmi. Inoltre si studia come deve essere fatta la prototipazione del software e la futura ricerca.

2.2 Dominio

Il **dominio** è il contesto in cui il software opera. Per definirlo occorre costruire un **glossario**, ovvero una collezione di definizioni di termini rilevanti in quel dominio specifico e che può essere riusato in progetti successivi nello stesso dominio. Inoltre occorre definire un **modello statico**, quindi come interagiscono fra loro gli elementi del dominio staticamente, e un **modello dinamico**, ovvero come si comporta il dominio in base all'avvenire di un determinato evento che può coinvolgere gli utenti. Questi due modelli possono essere descritti sia tramite l'uso del linguaggio UML ³, sia usando la semplice descrizione testuale.

2.3 Requisiti

Definizione (Requisito). Il requisito è una proprietà che deve essere garantita dal sistema per soddisfare una qualsiasi necessità dell'utente.

I requisiti possono dividersi in due categorie:

³https://it.wikipedia.org/wiki/Unified_Modeling_Language

- **Requisiti funzionali:** quelli che descrivono le funzionalità e il comportamento del software.
- **Requisiti non funzionali:** descrivono le proprietà del software o del processo di sviluppo. Ad esempio le caratteristiche di efficienza e affidabilità, l'interfaccia, il linguaggio di programmazione e l'ambiente di sviluppo scelti, i vincoli legislativi e i requisiti hardware o di rete.

I requisiti possono essere descritti mediante l'uso di diversi linguaggi, formali o meno. In questo caso si vede la descrizione dei requisiti mediante la produzione di un documento scritto in linguaggio naturale.

2.4 Documento dei Requisiti

Questo documento è un contratto tra lo sviluppatore e il committente, che elenca i requisiti e i vincoli che il software deve soddisfare, e specifica anche una *deadline* per la consegna del progetto.

2.5 Fasi dell'Analisi dei Requisiti

L'*analisi dei requisiti* viene svolta in cinque passi:

1. **Acquisizione**
2. **Elaborazione**
3. **Convalida**
4. **Negoziazione**
5. **Gestione**

2.5.1 Acquisizione

Il team di analisti incontra i membri dell'organizzazione del committente e si procede con la raccolta dei requisiti che può avvenire tramite: semplici interviste, questionari, costruzione di prototipi (anche su carta), studio di documenti o l'osservazione di possibili utenti mentre lavorano.

2.5.2 Elaborazione

Viene scritta la prima bozza del *documento dei requisiti*, dove quest'ultimi vengono trattati in modo più approfondito. La struttura del documento deve essere la seguente:

| |
|---|
| <i>Introduzione</i> |
| <i>Glossario</i> |
| <i>Definizione dei Requisiti Funzionali</i> |
| <i>Definizione dei Requisiti Non Funzionali</i> |
| <i>Architettura</i> : la strutturazione del software in sottosistemi. |
| <i>Specifica dettagliata dei Requisiti Funzionali</i> |
| <i>Modelli astratti</i> : descrivere il sistema in base a ciascun punto di vista. |
| <i>Evoluzione del sistema</i> : successivi cambiamenti. |
| <i>Appendici</i> : descrizione della piattaforma hardware, database, manuale utente e i piani di test. |
| <i>Indici</i> : costruire un lemmario, quindi una lista di termini che puntano ai requisiti che li usano. |

Nota Bene Nella descrizione dei requisiti occorre sempre usare la forma assertiva. Esempio:

Il <sistema> deve <funzionalità>/<proprietà>

2.5.3 Convalida

Nella fase di convalida occorre revisionare il documento per far sì che vengano evitati i seguenti difetti:

- **Omissioni**: requisiti mancanti.
- **Inconsistenze**: contraddizione tra i requisiti o tra un requisito e il contesto.
- **Ambiguità**: vaghezze o requisiti che possono avere più significati. Le ambiguità all'interno del linguaggio naturale possono essere portate da **quantificatori**, **disgiunzioni** oppure possono presentarsi ambiguità di **coordinazione** (nel caso si usano sia la o che la e nella stessa frase) oppure **referenziale** nell'uso non chiaro di pronomi. Inoltre occorre sempre evitare **verbi deboli**, **forme passive**, ovvero verbi senza un soggetto esplicito ed anche **negazione** e **doppie negazioni**.
- **Sinonimi** e **Omonimi**: termini diversi con lo stesso significato e termini uguali con significato diverso.
- **Presenza di dettagli tecnici**
- **Ridondanza**: può esserci, ma solo tra sezioni diverse del documento.

Le principali tecniche di convalida dei requisiti sono:

- **Deskcheck**
 - **Walkthrough**: ovvero il documento viene analizzato per intero sequenzialmente.
 - **Ispezione**: la lettura del documento è strutturata, utilizzando per esempio la tecnica del lemmario.
- Uso di **strumenti di analisi** del linguaggio naturale.
- Uso di **prototipi**.

Una volta trovati i difetti, è importante ricordarsi che vanno sempre risolti con il committente.

2.5.4 Negoziazione

In questa fase vengono assegnate delle priorità ai requisiti in base alle **esigenze del committente** e ai **costi** e **tempi** di produzione.

Questa fase è importante per decidere se alcuni requisiti possono essere **cancellati** oppure **sviluppati** successivamente.

MoSCoW Questa è una tecnica per dare priorità ai requisiti, i quali vengono divisi in quattro classi:

- **Must have**: requisiti irrinunciabili per il cliente.
- **Should have**: non necessari ma utili.
- **Could have**: non molto utili, da realizzare solo se c'è tempo.
- **Want to have**: da sviluppare in successive versioni.

2.5.5 Gestione

Questa fase si occupa di tre aspetti principali:

- **Identificazione**: ad ogni requisito viene assegnato un identificatore univoco.
- **Attributi**: ad ogni requisito vengono assegnati attributi relativi allo **stato**, **priorità**, **sforzo** in termini di giorni e/o personale da impiegare, **rischio**, e la **versione destinazione** per quanto riguarda lo sviluppo incrementale.
- **Tracciabilità**: ovvero la capacità di descrivere e seguire la vita di un requisito. Viene costruita una mappa tra i requisiti e le **componenti del sistema**, il **codice** ed i **test**.

Contratto Il documento dei requisiti precede la stipula del contratto.

2.6 Casi d'uso

I **casi d'uso** sono un altro modo per acquisire i requisiti, valutando le interazioni degli utenti col sistema e indicando al committente i risultati attesi. I casi d'uso oltre ad includere la sequenza corretta di eventi attesi devono anche presentare i comportamenti inattesi, ovvero le **eccezioni**.

2.7 User Stories

Sono un'altra tecnica di raccolta dei requisiti utilizzata principalmente nei **processi agile** e consiste nell'utilizzo di **user story cards** per scrivere i requisiti, che in linea generale hanno questa forma:

*Nel mio ruolo di <ruolo utente>, ho bisogno che il sistema
<funzione>, al fine di <beneficio>*

Contro Il problema principale delle *user stories* è la **scalabilità**, ovvero sono difficili da trasporre su grandi progetti e problematiche se il team è distribuito geograficamente. Inoltre, essendo informali e brevi non sono adatte per raggiungere degli accordi legali e raramente includono i *requisiti non funzionali*.