Ingegneria dei Requisiti

Corso tenuto dal Professor Mariano Ceccato

Università degli Studi di Verona

Alessio Gjergji

Indice

Capitolo 1

Introduzione

1.1 Terminologia di base

Per avere un'implementazione corretta e completa del software dobbiamo fare in modo che il software risolva un problema concreto del mondo reale. Dobbiamo quindi comprendere correttamente il mondo reale, ovvero come funziona, come dovrebbe funzionare, quali sono i vincoli che governano, capire il contesto in cui il software deve operare. Questo è il compito dell'ingegneria dei requisiti.

Esempio

- **Problema**: la gestione del freno a mano in un'automobile moderna potrebbe essere automatizzato.
- **Contesto**: se l'automobile è in movimento, se si sta frenando, se è intenzione del guidatore fermarsi, se il guidatore ha premuto il pedale del freno, ...

Quando parliamo di **requisiti** dobbiamo inoltre definire il **mondo**. Quando definiamo il mondo, abbiamo a che fare con elementi molto complessi che contengono di fatto elementi umani (*lo staff, l'utente, il cliente, ...*), elementi fisici (*dispositivi, software legacy, la natura, ...*) e dovrà quindi adattarsi alla situazione esistente.

Dobbiamo definire anche il mondo delle **macchine**. Parliamo quindi di tutto ciò che dovrà essere implementato e/o acquistato, come ad esempio *database, server, client, ...* e i componenti hardware e software che dovranno essere implementati.

L'ingegneria dei requisiti non si limita solamente al mondo delle macchine, ma tiene in considerazione anche gli effetti che il software ha sul mondo reale, che dovrà modellare.

Il mondo e le macchine hanno i propri fenomeni, ma ne condividono anche alcuni. Ad esempio, il mondo ha il *rilascio del freno a mano*, mentre le macchine hanno `errorCode = 013`. Nell'intersezione potremmo avere ad esempio `motor.Regime =‘up‘`.

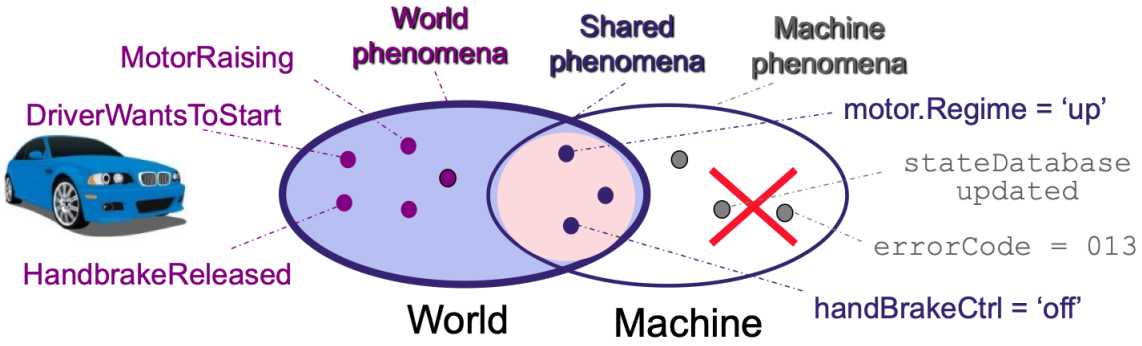


Figura 1.1.1: Il mondo e le macchine

Quando parliamo di requisiti, consideriamo solamente la parte relativa al mondo, che comprende quindi anche l'intersezione. È importante comprendere che non descrivono **nulla** riguardo alle macchine.

1.1.1 Le due versioni del mondo

Il sistema è l'insieme delle interazioni delle componenti che strutturano il mondo.

- Il sistema **as-is**: si tratta del sistema prima che il software venga implementato. Questo sistema è composto da solamente dal mondo.
- Il sistema **to-be**: si tratta del sistema come dovrebbe essere quando il software opererà. Questo sistema è composto dall'unione del mondo e delle macchine.

Definizione preliminare dell'ingegneria dei requisiti

Consiste in una serie di attività collegate tra loro che ci permettono di esplorare, valutare, documentare, consolidare, rivedere e adattare quelli che sono gli obiettivi, le capacità, i vincoli e le assunzioni in un sistema software. Basato sui problemi del sistema **as-is** e le opportunità date dalle nuove tecnologie.

1.1.2 I requisiti di sistema e i requisiti software

Il sistema (*unione del mondo reale e del sistema software*) sono i requisiti che fanno riferimento a tutto, si tratta degli statement che fanno parte del **system to-be**, mentre i requisiti software sono un sottoinsieme dei requisiti di sistema, fanno quindi riferimento solamente al **software to-be**, di fatto le funzionalità che il software che dovrà fornire per rispondere in maniera adeguata ai problemi del nostro utente.

1.2 Scope dell'ingegneria dei requisiti

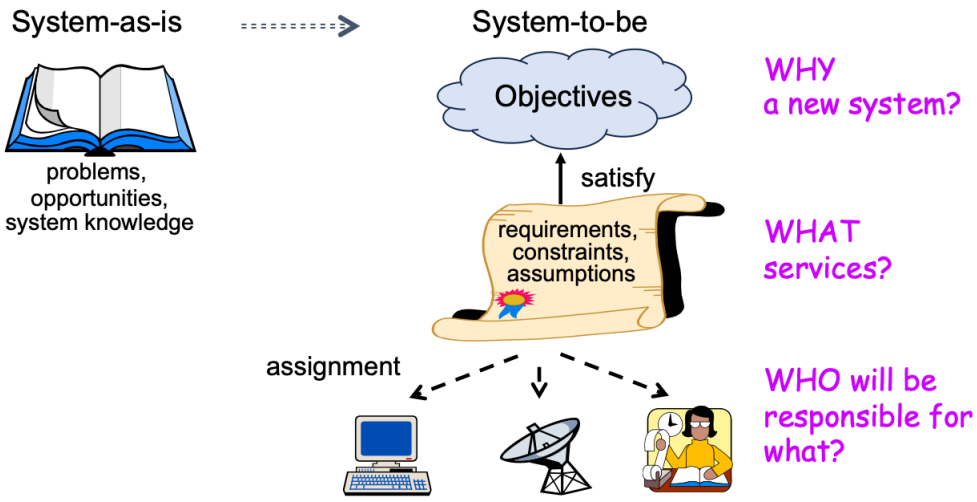


Figura 1.2.1: Scope dell'ingegneria dei requisiti

Prima di iniziare a lavorare sui requisiti, abbiamo il nostro system as-is, che comprenderà i problemi, le opportunità e le conoscenze sul sistema. Da qui mapperemo il tutto nel system to-be. Per comprendere al meglio il system to-be, dobbiamo ragionare in tre prospettive:

- Perché: quali sono gli obiettivi dell'abbinamento tra software e mondo reale. Quindi quali sono le necessità che il software dovrà soddisfare.
- Cosa: quali sono le funzionalità che il software dovrà fornire per soddisfare i bisogni del cliente, quali saranno i vincoli e le assunzioni che il software dovrà rispettare.
- Chi: chi sarà responsabile della fornitura delle funzionalità.

Questa è la chiave di lettura quando ragioniamo in termini di software: dobbiamo capire quali sono le funzionalità esposte e come queste danno una risposta agli obiettivi che ci siamo posti e chi ha la responsabilità di implementare tali funzionalità, se una persona, un componente interno al sistema, un componente esterno, ... Ci saranno quindi delle viste diverse che ci permetteranno di ragionare in questo spazio tridimensionale.

1.2.1 La dimensione del perché

Si tratta dell'identificazione degli obiettivi di cui il sistema software dovrà dare risposta. Vanno identificati, perciò l'interazione con gli stakeholder è essenziale, ma vanno inoltre analizzati. Tali obiettivi andranno poi rifiniti, perché tipicamente quello che viene raccolto in prima battuta è molto astratta e di alto livello. Vedremo in seguito come questi obiettivi verranno rifiniti e a che livello di dettaglio verranno portati.

Ci sono delle difficoltà intrinseche nel definire gli obiettivi:

- È importante capire il dominio del problema, ovvero capire il contesto in cui il software dovrà operare.
- Valutare opzioni alternative (*strade differenti per soddisfare lo stesso obiettivo*). Qui è opportuno limitare le scelte per poter fare una scelta più consapevole quando si avrà una conoscenza del dominio più approfondita per poter valutare in maniera più consapevole le opzioni alternative.
- Potremmo avere obiettivi in contrasto tra loro.

1.2.2 La dimensione del cosa

Sostanzialmente mira a identificare le funzionalità messe a disposizione all'interno del sistema software. L'obiettivo è date una risposta agli obiettivi identificati nella fase precedente in maniera adeguata. Ci saranno quindi delle metriche per misurare l'adeguatezza di tali risposte in modo che gli utenti siano soddisfatti a pieno delle funzionalità.

Le risposte dovranno essere fornite in maniera realistica, elaborando una soluzione che sia compatibile con quelli che sono i vincoli ambientali.

Anche qui ci sono delle difficoltà:

- Identificazione delle esatte funzionalità che dovranno essere fornite.
- Le funzionalità devono essere descritte in modo che siano comprensibili da tutti gli attori coinvolti, dove gli **attori** sono sia gli ingegneri del software e dei requisiti, sia gli stakeholder.
- Garantire una tracciabilità tra gli obiettivi rispetto alle funzionalità, ovvero un *link* tra gli obiettivi e le funzionalità che dovranno essere implementate. Il link è importante perché se ci accorgiamo che la motivazione cambia, probabilmente cambierà anche il "cosa".

1.2.3 La dimensione del chi

Si tratta di identificare chi sarà responsabile della fornitura delle funzionalità. Le varie responsabilità possono essere distribuite tra più attori, sia interni che esterni al sistema, possono essere quindi hardware o software o persone.

La difficoltà principale è quella di identificare il giusto grado di autonomia. Anche qui è opportuno documentare le alternative ma non adottarle subito, in modo da poter fare una scelta più consapevole in seguito. Solitamente si adotta un approccio iterativo raffinando le scelte in base alle informazioni che si acquisiscono nel tempo.

1.3 Tipologie di requisiti

Quando parliamo di requisiti, possiamo adottare due tipologie di registri:

- **Descrittivo**: raccontiamo le proprietà del sistema **indipendentemente** da come dovrebbe funzionare. Dipendendo quindi la leggi naturali, le leggi fisiche, i vincoli di sistema.

  Esempio: se le porte del treno sono chiuse, allora non sono aperte.

- **Percettivo**: raccontiamo le proprietà desiderabili nel sistema che potrebbero dipendere o meno dal comportamento del sistema.

  Esempio: le pote dovrebbero sempre rimanere chiuse quando il treno è in movimento.

La seconda tipologia di frasi sono quelle che possono essere negoziate, cambiate, modificate, mentre le prime sono inalterabili.

Gli statement formulati possono avere differenze per quanto riguarda lo scope. Quindi potrebbero riferirsi a fenomeni che riguardano solamente l'ambiente o tra l'ambiente e il sistema software.

1.3.1 Requisiti di sistema e requisiti software

I requisiti di sistema sono frasi prescrittive che si riferiscono ai fenomeni ambientali e dovranno essere soddisfatti dal sistema software. Tali frasi dovranno essere formulate con un vocabolario che sia comprensibile da tutti gli attori coinvolti. Ad esempio, `TrainMoving` $\rightarrow$ `DoorsClosed`.

I requisiti software sono frasi prescrittive che si riferiscono ai fenomeni condivisi tra il sistema software e l'ambiente, che sono supportate dal software-to-be. Tali frasi sono formulate nel vocabolario degli sviluppatori. Ad esempio, `measuredSpeed` $\geq 0 \rightarrow$ `doorState` = `'closed'`.

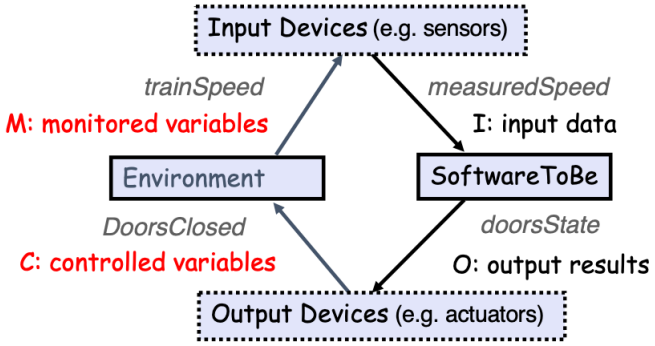
Proprietà di dominio, assunzioni e definizioni

Una proprietà di **dominio** è frase descrittiva che riguarda un fenomeno del mondo reale, che avviene indipendentemente dall'esistenza del sistema software. Ad esempio, "se l'accelerazione del treno è maggiore di 0, allora la velocità del treno è diversa da 0".

Le **assunzioni** sono frasi che devono essere soddisfatte dall'ambiente quando andiamo a considerare il software-to-be e sono formulate in termini dei fenomeni del mondo reale. Tipicamente sono prescrittive, ma non sempre. Ad esempio, "la velocità misurata è diversa da 0 se e solo se la velocità del treno è diversa da 0".

Le **definizioni** sono frasi che definiscono il significato preciso del concetto del sistema o della terminologia ausiliaria. Ad esempio, "la velocità misurata è la velocità misurata dal tachimetro del treno".

1.3.2 Relazione tra i requisiti software e i requisiti di sistema



Quando parliamo di requisiti software abbiamo questo modello quaternario. Possiamo avere degli *input device* e degli *output device* che sono i componenti che permettono di far si che il software possa interagire con il mondo reale.

Mappatura dei requisiti software e dei requisiti di sistema

Se i requisiti software sono soddisfatti, le assunzioni sono soddisfatte e le proprietà di dominio sono soddisfatte, allora i requisiti di sistema sono soddisfatti.

$$\texttt{SOFREQ}, \texttt{ASM}, \texttt{DOM} \models \texttt{SYSREQ}$$

1.4 Categorie di requisiti

I requisiti possono essere classificati in base a diverse categorie:

- **Funzionali**: sono frasi che raccontano quelle che sono le funzionalità che il software dovrà fornire per soddisfare i bisogni del cliente. Catturando il funzionamento del sistema software.
- **Non funzionali**: sono frasi che raccontano le proprietà del sistema software che non sono legate alle funzionalità. Sono quindi legati alla qualità del software. Ad esempio, la sicurezza, la performance, la scalabilità, ...

I requisiti non funzionali, essendo un po' trasversali, potrebbero non emergere direttamente dagli stakeholder, quindi è opportuno adottare delle tassonomie per poterli identificare. Possono essere raccolti mediante domande esplicite mediante interviste, questionari, ...

1.5 Il ciclo di vita dei requisiti

1.5.1 Comprensione del dominio

Prima di tutto è necessario far un passaggio di comprensione del dominio, in modo da comunicare con i committenti e gli stakeholder con la terminologia e la comprensione del dominio

stesso. Principalmente si studia il *system as-is* e si cerca di capire come funziona il mondo prima che il software venga implementato. Studiando quindi l'organizzazione, studiando i ruoli, le prassi e individuando i problemi e le opportunità. Si identificano gli stakeholder e si cerca di capire quali sono i loro bisogni e le loro aspettative.

Ciò che viene prodotto da questa prima fase è un primo glossario della terminologia del dominio.

1.5.2 Elicitazione dei requisiti

In questa fase si può iniziare ad esplorare il problema che il software dovrà risolvere. Capendo le specifiche esigenze degli stakeholder, le opportunità tecnologiche e le limitazioni.

1.5.3 Valutazione e negoziazione

A questo punto possiamo raggiungere il primo target dei requisiti concordati con gli stakeholder. Per raggiungere questo obiettivo, ci sarà, ovviamente una prima fase di negoziazione, dove emergeranno i punti di disaccordo tra le varie parti e iniziare a mediare tra essi trovando una possibile soluzione.

Tipicamente nella fase di raccolta dei requisiti, si raccolgono anche obiettivi contrastanti tra le diverse classi di stakeholder. Ad esempio alcune responsabilità potrebbero essere volute in capo a parti diverse.

Ciò che viene prodotto da questa fase è un primo documento di specifica dei requisiti.

1.5.4 Specifica e documentazione

La fase successiva è quella di documentare tali requisiti con le varie modalità per fissarli in maniera formale. Il documento dei requisiti deve essere un documento strutturato e deve essere redatto in modo che sia comprensibile da tutti gli attori coinvolti.

Ciò che viene prodotto da questa fase è un documento di specifica dei requisiti.

1.5.5 Consolidamento dei requisiti

La fase successiva è la fase di validazione e verifica, in modo da poter capire se è corretto e completo. In modo da capire se abbiamo lacune, inconsistenze e ambiguità. Per far ciò ci sono diverse tecniche per:

- Validare i requisiti: capire se i requisiti danno risposta concreta e completa a quelle che sono le necessità degli stakeholder.
- Verificare i requisiti: capire se sono presenti inconsistenze o lacune nei requisiti.
- Dovrà essere verificato rispetto a target qualitativi.

- I problemi trovati vanno risolti.

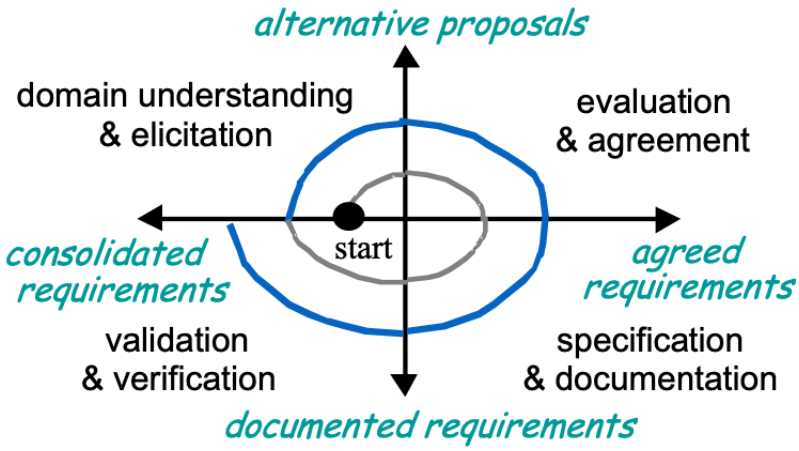
Ciò che viene prodotto da questa fase è un documento di specifica dei requisiti consolidato.

Il documento dei requisiti consolidato può fornire una base per la pianificazione del piano di test di **accettazione**. Definendo gli scenari in cui è possibile decidere se il software è completo o meno. Sono quindi **scenari** di alto livello. È possibile, inoltre, abbozzare un piano di test sviluppo.

Tale documento può essere utilizzato come base per la definizione di un contratto di fornitura del software.

La spirale dei requisiti

Nella prassi si adotta un approccio iterativo e incrementale, in modo da poter affinare i requisiti in base alle informazioni che si acquisiscono nel tempo.



Una visione evolutiva dei requisiti, che si adatta ai mutamenti dell'ambiente di lavoro, è caratteristica dello sviluppo agile del software.

1.6 Target qualitativi dei requisiti

I target di qualità, che verranno approfonditi in seguito, sono:

- **Completezza** degli obiettivi, dei requisiti e delle assunzioni
- **Coerenza** degli elementi del documento dei requisiti
- **Adeguatezza** dei requisiti, delle assunzioni e delle proprietà del dominio
- **Chiarezza** degli elementi del documento dei requisiti
- **Misurabilità** dei requisiti e delle assunzioni

- **Pertinenza** dei requisiti e delle assunzioni
- **Fattibilità** dei requisiti
- **Comprensibilità** degli elementi del documento dei requisiti
- **Buona strutturazione** del documento dei requisiti
- **Modificabilità** degli elementi del documento dei requisiti
- **Tracciabilità** degli elementi del documento dei requisiti

1.6.1 Errori comuni

Ci sono varie tipologie di errori che non devono essere commessi:

Tra questi troviamo:

- **Omissione**: l'assenza di requisiti critici rappresenta un errore grave.
- **Contraddizione**: la presenza di requisiti che si contraddicono tra loro costituisce un errore critico.
- **Inadeguatezza**: requisiti che non soddisfano le necessità o gli standard previsti sono considerati errori critici.
- **Ambiguità**: requisiti poco chiari o interpretabili in modi diversi sono errori gravi.

Altri problemi comuni includono:

- **Inmisurabilità**: requisiti che non possono essere misurati o quantificati.
- **Rumore e sovraspecificazione**: dettagli superflui o eccessivi che complicano il documento dei requisiti.
- **Infattibilità**: requisiti che non possono essere realizzati, spesso dovuti a pensieri desiderativi.
- **Inintelligibilità**: requisiti che non sono chiari o comprensibili.
- **Scarsa strutturazione**: problemi nella struttura del documento, come riferimenti anticipati o errori di organizzazione.
- **Opacità**: mancanza di trasparenza o chiarezza nei requisiti.

1.6.2 Il processo di ingegneria dei requisiti può variare a seconda del tipo di progetto

Il processo di ingegneria dei requisiti può variare significativamente in base al tipo di progetto. Alcuni dei principali fattori di variazione includono:

- **Progetti green-field** vs. **Progetti brown-field**: I progetti green-field partono da zero, mentre i progetti brown-field riguardano l'aggiornamento o l'espansione di sistemi esistenti.

- **Progetti guidati dal cliente** vs. **Progetti guidati dal mercato**: I progetti guidati dal cliente si concentrano sulle specifiche esigenze di un cliente particolare, mentre i progetti guidati dal mercato mirano a soddisfare le esigenze generali del mercato.
- **Progetti in-house** vs. **Progetti esternalizzati**: I progetti in-house sono sviluppati internamente all'organizzazione, mentre quelli esternalizzati vengono affidati a fornitori esterni.
- **Progetti single-product** vs. **Progetti product-line**: I progetti single-product riguardano un singolo prodotto, mentre i progetti product-line coinvolgono una linea di prodotti correlati.

I fattori di variazione includono:

- Il peso relativo di attività come l'elicitation, la valutazione, la documentazione, il consolidamento e l'evoluzione.
- L'intreccio tra ingegneria dei requisiti e design.
- Il peso relativo dei requisiti funzionali rispetto a quelli non funzionali.
- I tipi di stakeholder e sviluppatori coinvolti.
- Gli usi specifici del documento dei requisiti.
- L'uso di tecniche specifiche.

1.7 Ostacoli nell'ingegneria dei requisiti

Di fatto l'ingegneria dei requisiti è una disciplina molto complessa, che si trova ad affrontare diversi ostacoli. Il costo di identificare e correggere errori nei requisiti aumenta con il tempo, quindi è importante riuscire a identificarli il prima possibile. I costi in fasi avanzate possono arrivare anche a 200 volte il costo di identificazione in fase iniziale.

1.7.1 Ostacoli alla buona pratica dell'ingegneria dei requisiti

Ci sono diversi ostacoli che possono impedire una buona pratica dell'ingegneria dei requisiti. Tra questi troviamo:

- **Gli sforzi nell'ingegneria dei requisiti sono spesso spesi senza la garanzia che il contratto del progetto venga concluso**: Ciò significa che una quantità significativa di lavoro può essere svolta senza avere la certezza che il progetto andrà effettivamente avanti.
- **Pressione su scadenze strette, costi a breve termine e aggiornamenti tecnologici**: Le tempistiche ridotte, i vincoli economici e la necessità di mantenere il passo con la tecnologia possono mettere sotto pressione il processo di ingegneria dei requisiti, portando a compromessi nella qualità.
- **Scarsa disponibilità di studi sull'economia dell'ingegneria dei requisiti**: Esiste una mancanza di dati quantitativi sui benefici e sui risparmi dei costi derivanti

dall'ingegneria dei requisiti, rendendo difficile misurare il progresso rispetto alle fasi di progettazione e implementazione.

  - Mancanza di dati quantitativi sui benefici e sui risparmi dei costi dell'ingegneria dei requisiti.
  - Il progresso nel processo di ingegneria dei requisiti è più difficile da misurare rispetto alla progettazione e all'implementazione.

- **I documenti dei requisiti sono talvolta percepiti come**:
  - Grandi, complessi e rapidamente obsoleti.
  - Troppo distanti dal prodotto eseguibile che i clienti stanno pagando.

1.7.2 Sviluppo Agile e Ingegneria dei Requisiti

Lo sviluppo agile può superare alcuni degli ostacoli associati all'ingegneria dei requisiti. Alcuni dei modi in cui lo sviluppo agile riesce a farlo includono:

- **Un maggiore sviluppo agile può superare alcuni ostacoli**:
  - Fornendo in modo continuo e precoce funzionalità di valore per il cliente.
  - Riducendo la distanza tra i requisiti e il codice.
- **Cicli brevi di ingegneria dei requisiti nel processo a spirale, ciascuno seguito direttamente da un breve ciclo di implementazione**:
  - Gli incrementi funzionali utili vengono ricavati direttamente dall'utente.
  - Le fasi di valutazione, specifica e consolidamento sono spesso abbreviate (ad esempio, la specifica equivale a un caso di test sull'implementazione).
  - Gli incrementi vengono implementati e testati da un piccolo team nello stesso luogo, vicino all'utente per un feedback immediato, utilizzando regole rigorose.

1.7.3 Assunzioni forti per il successo dell'agilità

Perché l'agilità abbia successo, sono necessarie alcune assunzioni forti. Queste includono:

- **Tutti i ruoli degli stakeholder sono riducibili a un singolo ruolo**: Questo implica che le responsabilità e i compiti possono essere gestiti da un'unica figura.
- **Il progetto è sufficientemente piccolo da essere assegnabile a un singolo team di piccole dimensioni, localizzato in un unico luogo**: Questo team include programmatori, tester e manutentori.
- **L'utente può interagire prontamente ed efficacemente**: La comunicazione con l'utente deve essere rapida e produttiva.

- **La funzionalità può essere fornita rapidamente, costantemente e in modo incrementale, dalle parti essenziali a quelle meno importanti**: Non è necessaria la prioritizzazione delle funzionalità.
- **Gli aspetti non funzionali, le assunzioni sull'ambiente, gli obiettivi, le opzioni alternative e i rischi possono ricevere poca attenzione**: Questi elementi sono considerati meno critici.
- **Poca documentazione è richiesta per la coordinazione del lavoro e la manutenzione del prodotto**: La precisione dei requisiti non è essenziale; la verifica prima della codifica è meno importante di un rilascio anticipato.
- **Le modifiche ai requisiti non richiedono probabilmente un rifacimento significativo del codice**: Le modifiche ai requisiti sono gestibili senza necessità di grandi ristrutturazioni del codice.

Maggiore/minore agilità è raggiungibile con un minore/maggiore peso nelle fasi di elicitazione, valutazione, documentazione e consolidamento dei cicli di ingegneria dei requisiti.

Capitolo 2

Elicitazione dei requisiti

L'obiettivo comprendere il dominio in cui stiamo lavorando e raccogliere e capire i requisiti del software che dovrà essere implementato.

2.1 Acquisizione della conoscenza

L'acquisizione della conoscenza è una fase cruciale nel processo di ingegneria dei requisiti e include diverse attività importanti.

In primo luogo, è essenziale studiare il sistema attuale, noto come sistema *as-is*. Questo studio comprende la comprensione dell'organizzazione aziendale (*struttura, dipendenze, obiettivi strategici, politiche, flussi di lavoro e procedure operative*) e del dominio applicativo (*concetti, obiettivi, compiti, vincoli e regolamenti*). Inoltre, è necessario analizzare i problemi esistenti nel sistema attuale, identificandone sintomi, cause e conseguenze.

Un'altra attività chiave è l'analisi delle opportunità tecnologiche e delle nuove condizioni di mercato, per valutare le possibilità offerte dalle nuove tecnologie e comprendere come i cambiamenti nelle condizioni di mercato possano influenzare il sistema.

Identificare gli stakeholder del sistema è fondamentale. Gli stakeholder sono tutte le parti interessate che hanno un'influenza o sono influenzate dal sistema. Comprendere le loro esigenze e aspettative è vitale per il successo del progetto.

L'identificazione degli obiettivi di miglioramento per il sistema futuro, noto come sistema *to-be*, è un'altra attività importante. Questo include l'analisi dei vincoli organizzativi e tecnici, l'esplorazione di opzioni alternative, la definizione delle responsabilità e lo sviluppo di scenari ipotetici di interazione tra software e ambiente. Infine, è cruciale stabilire i requisiti specifici per il software e fare assunzioni sull'ambiente operativo.

Queste attività forniscono una base solida per il processo di ingegneria dei requisiti, assicurando che il sistema sviluppato soddisfi le esigenze degli stakeholder e funzioni efficacemente nel suo ambiente operativo.

Ci sono sostanzialmente due macro-approcci per l'acquisizione della conoscenza:

- **Artefact-driven**: Questo approccio si basa sull'analisi di documenti, modelli e altri artefatti esistenti per comprendere il sistema attuale e identificare i requisiti del sistema futuro. È utile quando il sistema attuale è ben documentato e i requisiti del sistema futuro sono chiari.
- **Stakeholder-driven**: Questo approccio si basa sull'interazione diretta con gli stakeholder per comprendere le loro esigenze e aspettative e identificare i requisiti del sistema futuro. È utile quando il sistema attuale è poco documentato o i requisiti del sistema futuro sono incerti.

2.1.1 Analisi degli stakeholder

Gli stakeholder sono tutte le parti interessate che hanno un'influenza o sono influenzate dal sistema.

La cooperazione degli stakeholder è essenziale per il successo dell'ingegneria dei requisiti. Il processo di elicitazione de requisiti può essere visto come un apprendimento cooperativo, dove la collaborazione tra tutte le parti coinvolte porta a una migliore comprensione dei requisiti del sistema.

Per garantire una copertura adeguata e comprensiva del mondo dei problemi, è necessario selezionare un campione rappresentativo degli stakeholder. Questa selezione deve essere dinamica, adattandosi man mano che si acquisiscono nuove conoscenze.

La selezione degli stakeholder si basa su diversi criteri, tra cui:

- La posizione rilevante nell'organizzazione.
- Il ruolo nella presa di decisioni e nel raggiungimento degli accordi.
- Il tipo di conoscenza contribuita e il livello di competenza nel dominio.
- L'esposizione ai problemi percepiti.
- Gli interessi personali e i potenziali conflitti.
- L'influenza nell'accettazione del sistema.

Selezionare gli stakeholder giusti è cruciale per assicurare che tutte le prospettive rilevanti siano considerate e che il sistema finale soddisfi le esigenze di tutti gli interessati.

2.1.2 Difficoltà nell'acquisizione della conoscenza dagli stakeholder

L'acquisizione della conoscenza dagli stakeholder presenta diverse difficoltà, tra cui fonti di informazione distribuite, punti di vista conflittuali e difficoltà di accesso alle persone chiave e ai dati. Gli stakeholder possono avere background, terminologie e culture differenti, e la conoscenza tacita o le esigenze nascoste possono complicare ulteriormente il processo. Inoltre, i dettagli irrilevanti, la politica interna, la competizione e la resistenza al cambiamento possono influire negativamente.

Il turnover del personale e i cambiamenti organizzativi e delle priorità possono creare ulteriori sfide.

Per superare queste difficoltà, sono essenziali abilità di comunicazione, costruzione di relazioni di fiducia e riformulazione continua della conoscenza attraverso riunioni di revisione.

2.1.3 Studio del background

Il processo di acquisizione della conoscenza inizia con la raccolta, la lettura e la sintesi dei documenti pertinenti. Questi documenti riguardano:

- **L'organizzazione**: include organigrammi, piani aziendali, rapporti finanziari, verbali di riunioni, ecc.
- **Il dominio**: comprende libri, indagini, articoli, regolamenti e rapporti su sistemi simili nello stesso dominio.
- **Il sistema attuale (as-is)**: include flussi di lavoro documentati, procedure, regole aziendali, documenti scambiati, rapporti di difetti/reclami, richieste di modifica, ecc.

Questo studio fornisce le basi necessarie per prepararsi prima di incontrare gli stakeholder e rappresenta un prerequisito per altre tecniche. Tuttavia, un problema comune è la gestione di una grande quantità di documentazione, dettagli irrilevanti e informazioni obsolete. La soluzione è utilizzare la meta-conoscenza per selezionare le informazioni rilevanti, sapendo cosa è necessario conoscere e cosa non lo è.

2.1.4 Raccolta dei dati

La raccolta dei dati è un'attività importante che si concentra sulla raccolta di fatti e cifre non documentati. Questi dati possono includere informazioni di marketing, statistiche di utilizzo, dati sulle prestazioni e costi. La raccolta può avvenire tramite esperimenti progettati o tramite la selezione di set di dati rappresentativi da fonti disponibili, utilizzando tecniche di campionamento statistico.

Questa attività può integrare lo studio del background, fornendo ulteriori informazioni utili per l'elicitazione dei requisiti non funzionali relativi a prestazioni, usabilità e costi.

Tuttavia, ci sono alcune difficoltà nella raccolta dei dati. Ottenere dati affidabili può richiedere tempo e i dati devono essere interpretati correttamente per essere utili.

2.1.5 Questionari

L'utilizzo dei questionari è un metodo efficace per raccogliere informazioni dagli stakeholder in modo rapido, economico e a distanza. I questionari consistono in una lista di domande inviate agli stakeholder selezionati, ognuna con una lista di possibili risposte. Possono includere:

- *Domande a scelta multipla*: dove si seleziona una risposta da una lista di opzioni.

- *Domande con pesatura*: dove si chiede di attribuire un peso a una lista di affermazioni, qualitativamente (ad esempio, "alto" o "basso") o quantitativamente (percentuali), per esprimere l'importanza percepita, le preferenze, i rischi, ecc.

I questionari sono utili per ottenere rapidamente informazioni soggettive da molte persone e possono essere d'aiuto nella preparazione di interviste più focalizzate.

Tuttavia, i questionari devono essere preparati con attenzione per evitare bias multipli (*dei destinatari, dei rispondenti, delle domande, delle risposte*) e informazioni inaffidabili dovute a fraintendimenti o risposte incoerenti.

Ecco alcune linee guida per la progettazione e la validazione dei questionari:

- Selezionare un campione rappresentativo e statisticamente significativo di persone, fornendo motivazioni per rispondere.
- Verificare la copertura delle domande e delle possibili risposte.
- Assicurarsi che le domande e le formulazioni siano imparziali e non ambigue.
- Aggiungere domande ridondanti implicitamente per rilevare risposte incoerenti.
- Far controllare il questionario da una terza parte.

Pro dei questionari

- Raccolta rapida di informazioni
- Economici e facili da distribuire
- Utili per preparare interviste focalizzate

Contro dei questionari

- Possibili bias dei destinatari e rispondenti
- Rischio di fraintendimenti nelle domande e risposte
- Difficoltà nell'assicurare risposte coerenti

2.1.6 Card sorting e repertory grids

L'obiettivo del card sorting e delle repertory grids è acquisire ulteriori informazioni sui concetti già elicitati. Nel card sorting, si chiede agli stakeholder di dividere un set di carte, ognuna rappresentante un concetto, in sottogruppi basati sui loro criteri. Per ogni sottogruppo, si indaga sulle proprietà condivise utilizzate per la classificazione. Questo processo è iterativo e può essere ripetuto per nuovi raggruppamenti e proprietà.

Ad esempio, nel contesto di un sistema di pianificazione delle riunioni, le carte "Riunione" e "Partecipante" potrebbero essere raggruppate insieme, suggerendo che *i partecipanti devono essere invitati alla riunione*. Nella successiva iterazione, lo stesso raggruppamento potrebbe indicare che *i vincoli dei partecipanti per la riunione devono essere conosciuti*.

Nel repertory grid, si chiede agli stakeholder di caratterizzare un concetto attraverso attributi e intervalli di valori, formando una griglia concetto-attributo. Ad esempio, per il concetto di "Riunione", gli attributi potrebbero essere *Data* (Lun-Ven) e *Luogo* (Europa).

Il conceptual laddering, invece, richiede agli stakeholder di classificare i concetti target lungo collegamenti di classe-sottoclasse, come ad esempio `RiunioneRegolare` e `RiunioneOccasionale` come sottoclassi di `Riunione`.

Questi metodi sono semplici, economici e facili da usare per l'elicitazione rapida di informazioni mancanti, ma i risultati possono essere soggettivi, irrilevanti o inaccurati.

Pro del card sorting e repertory grids

- Semplici ed economici
- Facili da usare
- Rapida elicitazione di informazioni

Contro del card sorting e repertory grids

- Risultati possono essere soggettivi
- Possibilità di informazioni irrilevanti
- Possibili inaccuratezze nei risultati

2.1.7 Scenari e storyboard

Gli scenari e gli storyboard aiutano ad acquisire o validare informazioni attraverso esempi concreti e narrazioni. Gli scenari illustrano sequenze tipiche di interazione tra i componenti del sistema per raggiungere un obiettivo implicito, utilizzati sia per spiegare il sistema attuale (*as-is*) che per esplorare il sistema futuro (*to-be*).

Gli storyboard raccontano una storia attraverso una sequenza di istantanee, che possono essere frasi, schizzi, diapositive o immagini, e possono includere annotazioni che spiegano chi sono i partecipanti, cosa accade loro, perché accade e cosa succede in caso di eventi alternativi.

Gli scenari possono essere:

- *Scenario positivo*: un comportamento che il sistema dovrebbe coprire.
- *Scenario negativo*: un comportamento che il sistema dovrebbe escludere.
- *Scenario normale*: tutto procede come previsto.
- *Scenario anomalo*: una sequenza di interazione desiderata in situazioni di eccezione.

Pro degli scenari e storyboard

- Esempi concreti e contro-esempi
- Stile narrativo (*appealing to stakeholders*)
- Producono sequenze di animazione, casi di test di accettazione

Contro degli scenari e storyboard

- Inerentemente parziali (*problema di copertura del test*)
- Esplosione combinatoria (*cf. tracce di programma*)
- Sovraspecificazione potenziale: sequenziamento non necessario, confini prematuri software-ambiente
- Possono contenere dettagli irrilevanti, granularità incompatibili tra diversi stakeholder
- Mantengono i requisiti impliciti

Nonostante ciò, sono preziosi come veicoli iniziali per l'elicitazione dei requisiti.

2.1.8 Prototipi e mock-up

L'obiettivo dei prototipi e dei mock-up è verificare l'adeguatezza dei requisiti attraverso il feedback diretto degli utenti, mostrando uno schizzo ridotto del software futuro in azione. Questo metodo si concentra su requisiti poco chiari e difficili da formulare per elicitare ulteriori dettagli.

Un prototipo è un'implementazione rapida di alcuni aspetti del sistema:

- *Prototipo funzionale*: si focalizza su requisiti funzionali specifici, come l'avvio di una riunione o la raccolta dei vincoli dei partecipanti.
- *Prototipo dell'interfaccia utente*: si concentra sulla usabilità, mostrando moduli di input-output e pattern di dialogo.

I prototipi possono essere implementati rapidamente utilizzando linguaggi di programmazione di alto livello, linguaggi di specifica eseguibile, e servizi generici.

Prototipazione dei requisiti include due approcci principali:

- *Mock-up*: il prototipo viene scartato una volta che ha soddisfatto il suo scopo di chiarire e validare i requisiti.
- *Prototipo evolutivo*: il prototipo viene trasformato e perfezionato fino a diventare parte del prodotto finale.

Pro dei prototipi e mock-up

- Forniscono un'idea concreta di come sarà il software
- Chiariscono i requisiti, elicitano quelli nascosti, migliorano l'adeguatezza, e aiutano a comprendere le implicazioni
- Utili anche per la formazione degli utenti e come stub per test di integrazione

Contro dei prototipi e mock-up

- Non coprono tutti gli aspetti del sistema
- Possono mancare funzionalità importanti
- Ignorano requisiti non funzionali rilevanti (*prestazioni, costi, ecc.*)
- Possono creare aspettative troppo alte e fuorvianti
- Codice "quick-and-dirty" difficile da riutilizzare per lo sviluppo del software
- Potenziali incongruenze tra codice modificato e requisiti documentati