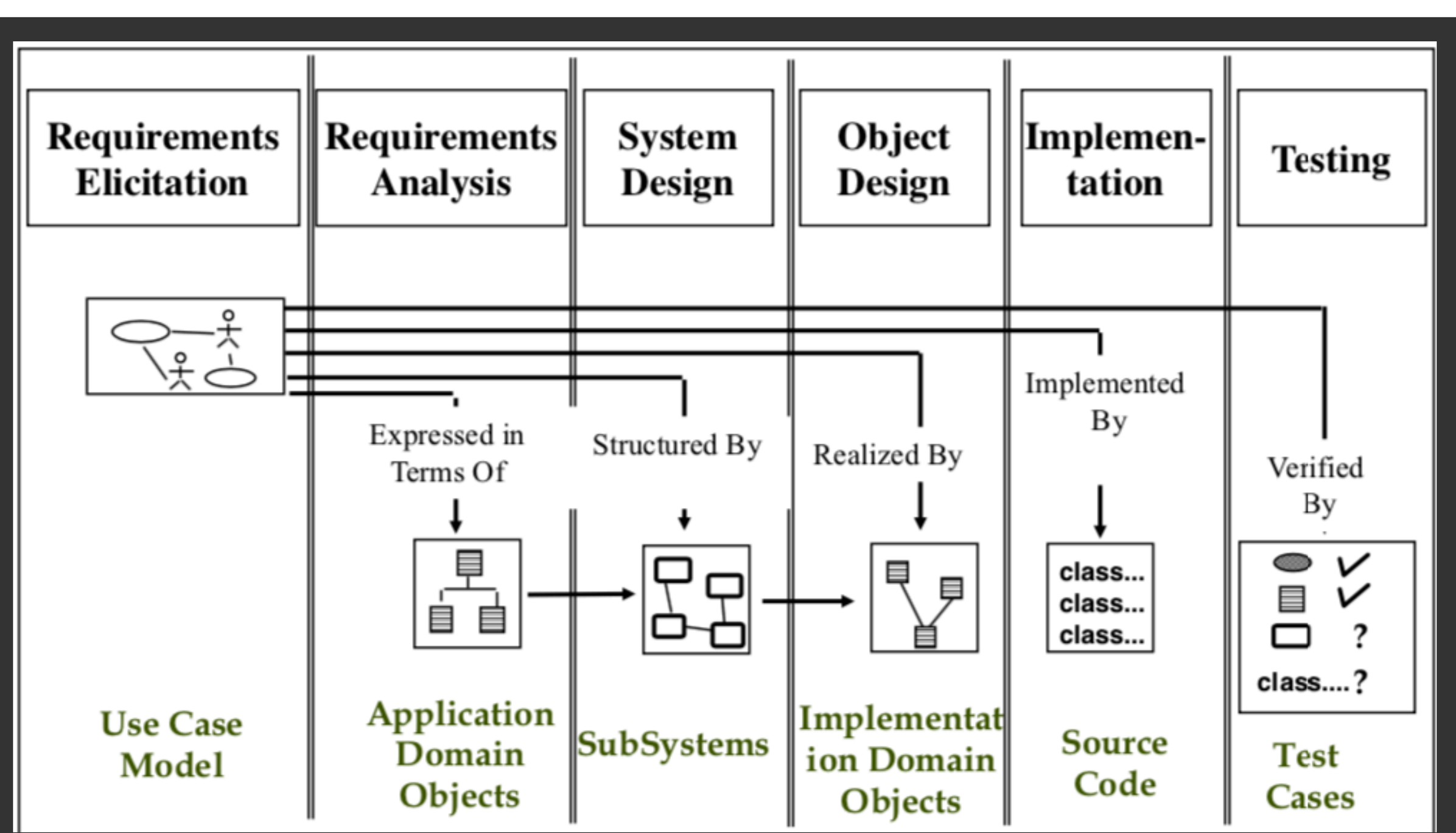


Il modello di riferimento è diverso da quello a cascata. Prende in considerazioni le fasi come:

1. Requirement Elicitation: l'output principale di questa fase è il modello degli "use-case"
2. Requirement Analysis: insieme alla precedente, forma l'ingegneria dei requisiti dove l'output è il RAD (Requirement Analysis Document)
3. System Design: si tratta di una progettazione ad alto livello in modo architetturale. Il sistema viene suddiviso in unità che possono essere sviluppati in modo indipendente, purché le interfacce rimangano invariate ed essere modificati indipendentemente senza rompere altri parti del sistema.
4. Object Design: si tratta di una progettazione ad basso livello quindi di dettaglio realizzato per mezzo di Implementation Domain Objects. Si tratta di una fase in cui si definiscono le scelte finali prima dell'implementazioni infatti l'analista deve scegliere tra i differenti modi di implementare i modelli di analisi, rispettando requisiti non funzionali e criteri di design.
5. Implementation: fase in cui si provvede a implementare i vari codici
6. Testing: fase in cui si provvede a verificare la completezza dei vari codici per mezzo di "Test Cases"

L'ingegneria dei requisiti mira a definire i requisiti del sistema in costruzione.
Lo sviluppo di un sistema non è fatto semplicemente prendendo una scena di un dominio qualsiasi. E' necessario rispondere a due domande:

1. Come possiamo identificare lo scopo di un sistema?
2. Fondamentale è la definizione del confine del sistema: cosa c'è dentro e cosa c'è fuori dal sistema?

A queste due domande viene data risposta nel processo dei requisiti: Requirements Elicitation e Requirements Analysis.

Elicitation significa "sollecitazione", "raccolta", ma ha un significato più profondo , in quanto sottintende la stimolazione di esprimere esigenze da raccogliere per procedere all'analisi dei requisiti. Per cui in questa fase c'è una collaborazione tra cliente , utente finale e analista, in una comunicazione bidirezionale, per cercare di far sbocciare quelli che sono i requisiti.

Un requisito è una "funzionalità" che il sistema deve avere o un "vincolo" che deve soddisfare per essere accettato dal cliente.
Quindi Requirements Elicitation si concentra sulla definizione dello scopo del sistema (raccolta dei requisiti). Il cliente, gli sviluppatori e gli utenti identificano un'area problematica e definiscono un sistema che risolve il problema. Tale definizione viene definita Requirements Specification e funge da contratto tra il cliente e gli sviluppatori.
La requirements specification è strutturata e formalizzata durante la fase di Requirements Analysis per produrre un "modello di analisi". Sia requirements specification sia il modello di analisi rappresentano le stesse informazioni. Differiscono soltanto nel come vengono strutturati: requirements specification è scritta in un linguaggio naturale comprensibile dal cliente e modello di analisi è scritta in un linguaggio semiformale o formale da poter consentire una comunicazione ottimale tra i vari sviluppatori. Dato che entrambi i modelli rappresentano gli stessi aspetti del sistema, i requisiti di richiesta e analisi avvengono contemporaneamente e in modo iterativo.
Requirements elicitation comprende le seguenti attività:

1. Identificare gli attori: Durante questa attività, gli sviluppatori identificano i diversi tipi di utenti che il futuro sistema supporterà.
2. Identificazione di scenari: Durante questa attività, gli sviluppatori osservano gli utenti e sviluppano una serie di scenari dettagliati per le funzionalità tipiche fornite dal sistema futuro. Gli scenari sono esempi concreti del futuro sistema in uso. Gli sviluppatori utilizzano questi scenari per comunicare con l'utente e approfondire la propria conoscenza del dominio dell'applicazione.
3. Identificazione degli use cases: Una volta che gli sviluppatori e gli utenti concordano su una serie di scenari, gli sviluppatori derivano dagli scenari una serie di use cases che rappresentano completamente il sistema futuro. Mentre gli scenari sono esempi concreti che illustrano un singolo caso, gli use cases sono astrazioni che descrivono tutti i casi possibili. Nel descrivere i casi d'uso, gli sviluppatori determinano l'ambito del sistema.
4. Raffinazione degli use cases: Durante questa attività, gli sviluppatori si assicurano che la specifica dei requisiti sia completa descrivendo dettagliatamente ogni caso d'uso e descrivendo il comportamento del sistema in presenza di errori e condizioni eccezionali.
5. Individuazione delle relazioni tra gli use cases: Durante questa attività, gli sviluppatori identificano le dipendenze tra i casi d'uso. Consolidando anche il modello del caso d'uso prendendo in considerazione funzionalità comuni. Ciò garantisce che le specifiche dei requisiti siano coerenti.
6. Individuazione dei requisiti non funzionali: Durante questa attività, sviluppatori, utenti e client concordano aspetti che sono visibili all'utente, ma non direttamente correlati alla funzionalità. Questi includono vincoli sulle prestazioni del sistema, sulla sua documentazione, sulle risorse che consuma, sulla sua sicurezza e sulla sua qualità.

Functional Requirements
Descrivono le interazioni tra il sistema e il suo ambiente indipendentemente dalla sua implementazione. L'ambiente include l'utente e qualsiasi altro sistema esterno con il sistema interagisce. Un esempio di requisiti funzionali sarebbe l'orologio "SatWatch", un orologio che si ripristina da solo senza l'intervento dell'utente.
Non Functional Requirements
Descrivono aspetti del sistema che non sono direttamente correlati al comportamento funzionale del sistema. Esso include un'ampia varietà di requisiti che si applicano a molti aspetti diversi del sistema, dall'usabilità alle prestazioni.
Il modello FURPS +2 fornisce le seguenti categorie di requisiti non funzionali:

1. L'usabilità è la facilità con cui un utente può imparare a operare, prepara input e interpretare output di un sistema o componente.
2. L'affidabilità è la capacità di un sistema o componente di eseguire le funzioni richieste nelle condizioni indicate per un periodo di tempo specificato. Esso comprende la robustezza (il grado in cui un sistema o un componente può funzionare correttamente in presenza di input non validi o condizioni ambientali stressanti) e sicurezza(una misura dell'assenza di conseguenze catastrofiche per l'ambiente)
3. I requisiti di prestazione riguardando gli aspetti quantificabili del sistema, come il tempo di risposta(la velocità con cui il sistema reagisce all'input dell'utente), il throughput(quanto lavoro può compiere il sistema entro un determinato periodo di tempo), la disponibilità(il grado di quale sistema o componente è operativo e accessibile quando richiesto per l'uso) e precisione
4. I requisiti di sopportabilità riguardano la facilità delle modifiche al sistema dopo l'implementazione, tra cui ad esempio l'adattabilità (la capacità di cambiare il sistema per gestire ulteriori concetti del dominio dell'applicazione), la manutenibilità (la capacità di cambiare il sistema per gestire le nuove tecnologie o per correggere i difetti) e l'internazionalizzazione (la capacità di cambiare il sistema per gestire ulteriori convenzioni internazionali, come lingue, unità e formati numerici).

Contrainsts o Pseudo requirements:
Imposto dal cliente o dall'ambiente in cui opera il sistema. Esempio: il linguaggio di implementazione deve essere Java.
Requirements Validation
I requirements sono continuamente validati dal cliente e dall'utente. La validazione è uno step critico nel processo di sviluppo, dato che sia il cliente e lo sviluppatore dipendono sulla requirements specification. La requirements validation implica che la verifica sia completa, coerente, inequivocabile e corretta, realistico e tacciabile.

1. E' completo se vengono descritti tutti gli scenari possibili attraverso il sistema
2. E' coerente (consistenza) se la specifica dei requisiti non si contraddice.
3. E' inequivocabile (chiarezza) se la specifica dei requisiti definisce esattamente un sistema (non vi è il caso in cui è possibile interpretare la specifica in due o più modi diversi)
4. E' corretta se rappresenta accuratamente il sistema di cui il cliente ha bisogno.
5. E' realistica se il sistema può essere implementato entro un limite di tempo e con attuali tecnologie.
6. E' rintracciabile se ogni requisito può essere rintracciato durante lo sviluppo del software alle sue corrispondenti funzioni di sistema e se ciascuna funzione di sistema può essere ricondotta al relativo set di requisiti.

La correttezza e completezza sono spesso difficili da stabilire, quindi la specifica dei requisiti deve essere attentamente esaminato da entrambe le parti. Inoltre parti del sistema che rappresentano un rischio elevato devono essere prototipale per dimostrare la loro fattibilità o per ottenere un feedback da parte dell'utente. Tutto questo servirà per il RAD affinché sia completo.
Proritizing Requirements
Ad ogni requirements gli viene dedicato una priorità, priorità che sono suddivise in tre categorie:

1. High priority: affrontato durante l'analisi, la progettazione e l'implementazione. Una funzione ad alta priorità deve essere dimostrata
2. Medium priority: affrontato durante l'analisi e il design. Di solito dimostrato nella seconda iterazione
3. Low priority: affrontato solo durante l'analisi. Illustra il modo in cui il sistema verrà utilizzato in futuro con una tecnologia non ancora disponibile

Tipi di requirements elicitation
Le attività di sollecitazione dei requisiti possono essere classificate in tre categorie.

1. Greenfield Engineering:
   - lo sviluppo inizia da zero, non esiste un sistema precedente, quindi i requisiti vengono estratti dagli utenti e dal client.
2. Re-engineering:
   - è la riprogettazione e la reimplementazione di un sistema esistente innescato da attivatori tecnologici o da processi aziendali. A volte la funzionalità del nuovo sistema viene estesa, ma lo scopo essenziale del sistema rimane lo stesso. I requisiti del nuovo sistema vengono estratti da un sistema esistente.
3. Interface Engineering:
   - E' la riprogettazione dell'interfaccia utente di un sistema esistente. Il sistema legacy non viene toccato, tranne che per la sua interfaccia che viene ridisegnata e reimplementata.
4. Forward engineering:
   - Produce un modello di codice sorgente che corrisponde a un modello a oggetti. Molti costrutti di moderazione, come le specifiche degli attributi e delle associazioni, possono essere mappati meccanicamente ai costrutti del codice sorgente supportati dal linguaggio di programmazione selezionato, mentre gli sviluppatori aggiungono corpi e metodi privati aggiuntivi.

Requirements elicitation è la più impegnativa in quanto richiede una continua collaborazione di diversi gruppi di partecipanti. Da un lato, il client e gli utenti sono esperti nel loro dominio e hanno un'idea generale di cosa dovrebbe fare il sistema ma spesso hanno poca esperienza nello sviluppo del software. D'altra parte gli sviluppatori hanno esperienza nella costruzione di sistemi, ma spesso hanno una scarica conoscenza dell'ambiente quotidiano degli utenti.
Scenari e casi d'uso forniscono strumenti per ridurre questo gap. Uno scenario descrive un esempio d'uso del sistema in termini di una serie di interazioni tra il l'utente e il sistema. Un caso d'uso è un'astrazione che descrive una classe di scenari. Entrambi sono scritti con un linguaggio naturale, quindi comprensibile al cliente.
Scenario
Gli scenari sono delle descrizioni dei vari possibili usi e sperimentazioni da parte delle persone nei confronti di sistemi e applicazioni informatici. Uno scenario è una descrizione concreta e informale di una singola caratteristica del sistema dal punto di vista di un singolo attore. Gli scenari non possono sostituire i casi d'uso in quanto si concentrano soltanto su istanze specifiche. Tuttavia, gli scenari migliorano la richiesta di requisiti fornendo uno strumento comprensibile per utenti e clienti.
Esempio: scenario per il sistema FRIEND, un sistema informativo per la risposta agli incidenti. In questo scenario, un ufficiale di polizia segnala un incendio e un Dispatcher avvia la risposta all'incidente.

Scenario warehouseOnFire per il caso d'uso ReportEmergency.

Nome dello scenario	wareHouseOnFire
Attori partecipanti	bob,alice:FiledOfficer john:Dispatcher
Flusso di eventi	1. bob percorrendo la strada principale con la sua auto di pattuglia nota che esce fumo da un magazzino. La sua compagna Alice, attiva la funzione "Segnala emergenza" dal suo laptop "amico" 2. Alice si dirige verso l'edificio, una breve descrizione della collocazione(ovvero, angolo nord-ovest) e un livello di emergenza. Oltre a un'unità antincendio, richiede diverse unità paramediche sulla scena, dato che l'area sembra essere relativamente occupata. Conferma il suo contributo e attende un riconoscimento 3. John, il dispatcher, viene avvisato dell'emergenza dalla stazione di lavoro. Esamina le informazioni inviate da alice e riconosce il rapporto. Assegna un'unità antincendio e due unità paramediche al sito dell'incidente e invia l'orario di arrivo previsto(ETA) ad Alice 4. Alice riceve il riconoscimento e l'ETA

Questo scenario descrive soltanto una situazione specifica e non in generale quindi se vi sono, in questo scenario, dei vari possibili ramificazioni (decisioni) è necessario rappresentare due scenari.
Gli scenari possono avere molti usi diversi durante la richiesta di requisiti.
Esistono diversi tipi di scenari come:

1. Scenari As-Is: descrivono una situazione attuale. Durante la reingegnerizzazione l'attuale sistema viene compreso osservando gli utenti e descrivendo le loro azioni come scenari. Questi scenari possono quindi essere validati per correttezza e accuratezza con gli utenti.
2. Scenari Visionari: descrivono un sistema futuro. Gli scenari visionari vengono utilizzati sia come punto di riferimento nello spazio di modulazione dagli sviluppatori che perfezionano le loro idee sul sistema futuro sia come mezzo di comunicazione per ottenere requisiti dagli utenti. Possono essere visti come un prototipo economico. Sono utili per la greenfield engineering e reengineering;
3. Scenari di valutazione: descrivono le attività dell'utente rispetto alle quali il sistema deve essere valutato. Lo sviluppo collaborativo di scenari di valutazione da parte di utenti e sviluppatori migliora anche la definizione della funzionalità testata da questi scenari. Utile per la validazione del sistema con il test di accettazione effettuata dal cliente, infatti questo tipo di test viene descritto attraverso questi scenari.
4. Scenari di formazione: sono esercitazione utilizzate per introdurre nuovi utenti nel sistema. Queste sono istruzioni dettagliate progettate per tenere in mano l'utente attraverso attività comuni.

Inizialmente ogni scenario è di alto livello e incompleto quindi per poter migliorarli ulteriormente vengono posti tali domande a se stesso o al cliente:

1. Quali sono i compiti che l'attore vuole che il sistema esegua?
2. A quali informazioni accede l'attore? Chi crea questi dati? Può essere modificato o rimosso? Da chi?
3. Di quali cambiamenti esterni ha bisogno l'attore per informare il sistema? Quante volte? Quando?
4. Di quali eventi ha bisogno il sistema informare l'attore? Con quale latenza?

Gli sviluppatori utilizzano documenti esistenti sul dominio dell'applicazione per rispondere a queste domande. Questi documenti includono manuali utente di sistemi precedenti, interviste a utenti e clienti. Gli sviluppatori dovrebbero sempre scrivere scenari utilizzando i termini del dominio dell'applicazione, anziché i propri termini. Man mano che gli sviluppatori acquisiscono ulteriori informazioni sul dominio dell'applicazione e sulle possibilità della tecnologia disponibile, perfezionano in modo iterativo e incrementale gli scenari per includere quantità crescenti di dettagli. Disegnare modelli di interfaccia utente spesso aiuta a trovare omissioni nelle specifiche e a costruire un quadro più concreto del sistema.
Identificare Use Cases
Un caso d'uso specifica tutti gli scenari possibili per una determinata funzionalità. Un caso d'uso viene avviato da un attore. Dopo la sua apertura, un caso d'uso può interagire anche con altri attori. Un caso d'uso rappresenta un flusso completo di eventi attraverso il sistema.
La descrizione di un caso d'uso comporta la specifica di quattro campi.

1. Attori Partecipanti
2. La descrizione delle condizioni di entrata e uscita di un caso d'uso consente agli sviluppatori di comprendere le condizioni in cui viene invocato un caso d'uso e l'impatto del caso d'uso sullo stato dell'ambiente e del sistema. Esaminando le condizioni di entrata e uscita dei casi d'uso, gli sviluppatori possono determinare se potrebbero mancare casi d'uso.
3. Descrivere il flusso di eventi di un caso d'uso consente a sviluppatori e clienti di discutere l'interazione tra attori e sistema. Ciò permette di definire il confine del sistema, definendo quali azioni vengono compiute dall'attore e quali azioni dal sistema.
4. Descrizione dei requisiti di qualità associati a un caso d'uso consente agli sviluppatori di ottenere requisiti non funzionali nel consenso di una funzionalità specifica.

Un esempio di un caso d'uso, ReportEmergency.

use case nome	ReportEmergency
Partcipating actor	Iniziato da FieldOfficer Comunicazione con Dispatcher
Flow of events	1. Il FieldOfficer attiva la funzione "Segnala emergenza" del suo terminale 2. Friend risponde presentando un modulo al FieldOfficer 3. L'ufficiale di campo completa il modulo selezionando il livello di emergenza, il tipo, la posizione e una breve descrizione della situazione. Il FieldOfficer descrive anche le possibili risposte alla situazione di emergenza. Una volta completato il modulo, FieldOfficer invia il modulo. 4. Friend riceve il modulo e notifica al Dispatcher 5. Il Dispatcher esamina le informazioni inviate e crea un incidente nel database invocando Open Incident use case. Il Dispatcher seleziona una risposta e riconosce il rapporto. 6. Friend riceve il riconoscimento e la risposta selezionata al FieldOfficer
Entry condition	Il FieldOfficer è loggato nel sistema Friend
Exit conditions	FieldOfficer ha ricevuto un riconoscimento e la risposta selezionata dal Dispatcher, OR FieldOfficer ha ricevuto una spiegazione che indica perché non è stato possibile elaborare la transazione.
Quality requirements	Il rapporto del FieldOfficer viene riconosciuto entro 30 secondi La risposta selezionata arriva entro e non oltre 30 secondi dopo essere stata inviata da Dispatcher.

Esempio appena elencato, mostra il caso d'uso ReportEmergency di cui lo scenario warehouseOnFire è un'istanza. L'attore FieldOfficer avvia questo caso d'uso attivando la funzione "Segnala emergenza" di Friend (il sistema). Si completa quando l'attore FieldOfficer riceve un riconoscimento dell'avvenuta creazione di un incidente. I passaggi 1 e 3 vengono avviati dall'attore, mentre i passaggi 2 e 4 vengono avviati dal sistema.
Inizialmente, gli sviluppatori nominati i casi d'uso, li collegano agli attori e forniscono una descrizione di alto livello del caso d'uso. Il nome di un caso d'uso dovrebbe essere una frase verbale che indica ciò che l'attore sta cercando di realizzare. La frase del verbo "Segnala emergenza" indica che un attore sta tentando di segnalare un'emergenza al sistema(al dispatcher).
Questo caso d'uso non si chiama "Record Emergency" perché il nome dovrebbe riflettere la prospettiva dell'attore, non il sistema. Inoltre, non viene chiamato "Tentativo di segnalare un'emergenza" perché il nome dovrebbe riflettere l'obiettivo del caso d'uso, non l'attività effettiva.

Raffinamento Use Cases

use case nome	ReportEmergency
Partcipating actor	Iniziato da FieldOfficer Comunicazione con Dispatcher
Flow of events	1. Il FieldOfficer attiva la funzione "Segnala emergenza" del suo terminale 2. Friend risponde presentando un modulo all'ufficiale. Il modulo include un menu di tipo emergenza (emergenza generale, incendio, trasporto) e posizione, descrizione dell'incidente, richiesta di risorse e campi di materiale pericoloso 3. Il FieldOfficer completa il modulo specificando minimamente il tipo di emergenza e i campi descrittivi. Il FieldOfficer può anche descrivere le possibili risposte alla situazione di emergenza e presentare le risorse specifiche. Una volta completato il modulo, FieldOfficer invia il modulo. 4. Friend riceve il modulo e notifica al Dispatcher una finestra di dialogo a comparsa. 5. Il Dispatcher esamina le informazioni inviate e crea un incidente nel database invocando Open Incident use case. Tutte le informazioni fornite nel modulo del FieldOfficer sono automaticamente incluse nell'incidente. Il Dispatcher seleziona una risposta allocando le risorse all'incidente(con il caso d'uso AllocateResuorces) e riconosce il rapporto di emergenza inviando un breve messaggio a FieldOfficer. 6. Friend visualizza il riconoscimento e la risposta selezionata su FieldOfficer
Entry condition	....
Exit conditions	....
Quality requirements	....

Una versione raffinata del caso d'uso ReportEmergency. E' stato esteso per includere dettagli sul tipo di incidenti noti a FRIEND e interazioni dettagliate che indicano come il Dispatcher riconosce il FieldOfficer.
Gli sviluppatori e gli utenti dovrebbero sforzarsi di affrontare la maggior parte dei problemi requisiti in anticipo in modo da poter ridurre il costo di modifica delle specifiche dei requisiti impreviste. Ci sono delle situazioni in cui ad un caso d'uso subisce molte modifiche e molte convalide quindi riscritti più volte o perfezionati o completamente caduto.

Euristica per lo sviluppo di scenari e casi d'uso

1. Utilizzare gli scenari per comunicare con gli utenti e per convalidare la funzionalità.
2. Perfeziona un singolo scenario per comprendere i presupposti dell'utente sul sistema. L'utente può avere familiarità con sistemi simili, nel qual caso l'adozione di convenzioni specifiche dell'interfaccia utente renderebbe il sistema più utilizzabile.
3. Definire moli scendi non molto dettagliati per definire l'ambito del sistema. Convalida con l'utente.
4. Utilizzare i mock-up solo come supporto visivo.
5. Presentare all'utente alternative multiple e molto diverse. La valutazione di diverse alternative amplia l'orizzonte dell'utente. Generare diverse alternative costringe gli sviluppatori a "pensare fuori dagli schermi"
6. Dettagliare un ampia sezione verticale (esempio—> scenario) quando l'ambito del sistema e le preferenze dell'utente sono ben comprese. Convalida con l'utente.

Come specificare un caso d'uso

1. Nome del caso d'uso
2. Attori: descrizione degli attori coinvolti in questo caso d'uso
3. Entry condition: usare delle sintassi predefinite come : "Questo caso d'uso inizia quando…."
4. Flow of events: una serie di eventi che descrivono le varie interazioni tra l'utente e il sistema
5. Exit condition: iniziare con: "questo caso d'uso termina quando…"
6. Exceptions: descrivere che cosa succede se le cose vanno male
7. Special Requirements: una lista non funzionale e contrasti.

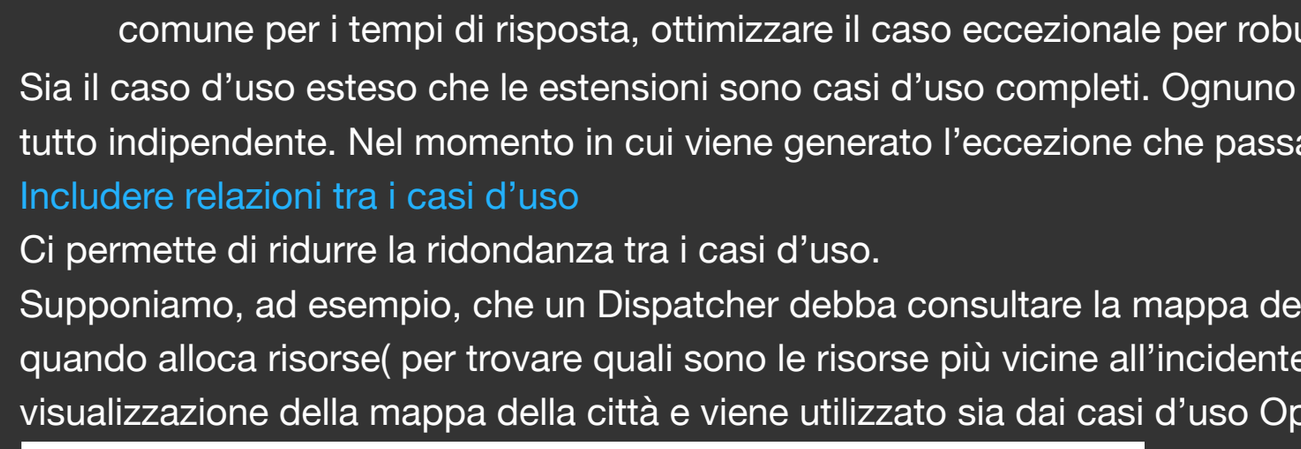
Il focus del raffinamento dei use cases è sulla completezza e correttezza. Gli sviluppatori identificano funzionalità non coperte dagli scenari e le documentano perfezionando i casi d'uso o scrivendone di nuovo. Il perfezionamento dei casi d'uso fornisce sempre più dettagli sulle caratteristiche fornite dal sistema e sui vincoli ad essi associati.
Prendiamo la tabella appena elencata:

1. Abbiamo aggiunto dettagli sugli attributi del modulo di segnalazione di emergenza e sui tipi di incidenti.
2. Sono specificate la sequenza di interazioni di basso livello tra l'attore e il sistema.
3. Abbiamo aggiunto informazioni su come il Dispatcher genera un riconoscimento selezionato le risorse.
4. Sono specificati i diritti d'accesso (quali attore possono invocare quali casi d'uso)
5. Le eccezioni mancanti vengono identificate e viene specificato il loro trattamento.
6. Vengono prese in considerazioni le funzionalità comuni tra casi d'uso.

Identificazione delle relazioni tra attori e casi d'uso
Le relazioni tra attori e casi d'uso consentono agli sviluppatori e agli utenti di ridurre la complessità del modello e aumentarne la comprensibilità.
Per modellare relazioni tra i casi usiamo invece:

- Relazioni "estese" per separare flussi di eventi eccezionali e comuni.
- Relazioni "inclusione" per ridurre la ridondanza tra i casi d'uso.

Tali relazioni sono da definirsi dipendenze.
Le relazioni di comunicazione tra attori e casi d'uso permettono di rappresentare il flusso di informazioni durante il caso d'uso. Specifichiamo quali attori comunicano con un caso d'uso specifico e quali attori possono accedere a informazioni specifiche e quali no. In questo modo siamo in grado di aver un controllo degli accessi per il sistema a un livello astratto.

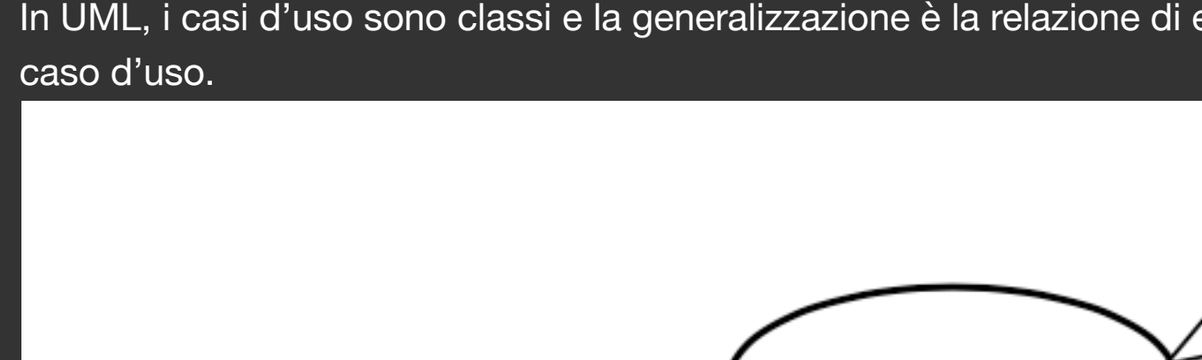


La frase <initiate> indica l'inizio del caso d'uso da parte di un attore e la frase <participate> indica che un attore comunica con il caso d'uso.
Estendere le relazioni tra i casi d'uso
Un caso d'uso estende un altro caso d'uso se il caso d'uso esteso può includere il comportante dell'estensione in determinate condizioni. Considerando l'esempio di sopra , supponiamo che la connessione tra la stazione FieldOfficer e la stazione Dispatcher sia interrotta mentre FieldOfficer sta compilando il modulo (ad esempio, l'auto del FieldOfficer entra in un tunnel). La stazione FieldOfficer deve informare il FieldOfficer che il suo modulo non è stato consegnato e quali misure dovrebbe prendere. Il caso d'uso di ConnectionDown è modellato come un'estensione di ReportEmergency, dove viene avviato in base alle condizioni del caso medesimo

La separazione di flussi eventi eccezionali e facoltativi dal caso d'uso di base presenta due vantaggi:

1. Il caso base diventa più semplice da capire.
2. Il caso base si distingue dal caso eccezionali, permettendo quindi agli sviluppatori di trattare ogni funzionalità in modo diverso(ad esempio, ottimizzare il caso comune per i tempi di risposta, ottimizzare il caso eccezionale per robustezza).

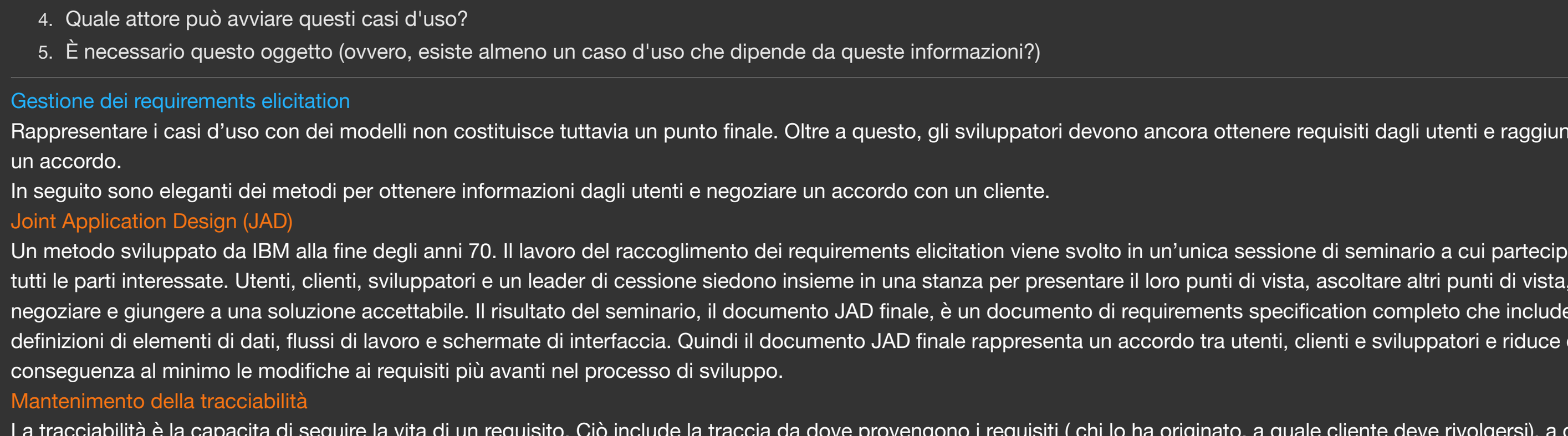
Sia il caso d'uso esteso che le estensioni sono casi d'uso completi. Ognuno di essi deve avere condizioni di ingresso e di fine ed essere comprensibile dall'utente come un tutto indipendente. Nel momento in cui viene generato l'eccezione che passa al caso d'uso che lo estende, non si torna più al caso base
Includere relazioni tra i casi d'uso
Ci permette di ridurre la ridondanza tra i casi d'uso.
Supponiamo, ad esempio, che un Dispatcher debba consultare la mappa della città quando apre un incidente (per valutare quali aree sono a rischio durante un incendio) e quando alloca risorse(per trovare quali sono le risorse più vicine all'incidente). In questo caso, il caso d'uso ViewMap descrive il flusso di eventi richiesti durante la visualizzazione della mappa della città e viene utilizzato sia dai casi d'uso OpenIncident che AllocateResuorces.



Esso permette quindi di avere delle descrizioni più brevi e pochi dati duplicati.

Estendi VS Include Relazioni
Includere ed estendere per certi aspetti sono simili ma hanno una logica diversa.
Un comportamento che è legato a un evento e si verifica in numero limitato di casi d'uso dovrebbero essere rappresentato con una relazione inclusa.
Invece un comportamento che può verificarsi in qualsiasi momento o la cui occorrenza può essere più facilmente specificata come condizione di ingresso deve essere rappresentato con una relazione estesa. Esso comprende situazioni eccezionali.
Quindi nel caso di estendere, la condizione che lo attiva viene inserito nelle "entry condition" dello use case che estende.
Mentre per Include, ogni use case che include deve specificare dove viene invocato lo use case incluso.

Relazioni di generalizzazione in casi d'uso
La realizzazione di una generalizzazione determina un comportamento comune tra i casi d'uso. Il figlio (caso d'uso) eredita la struttura , il comportamento e le relazioni di un altro attore(genitore).
In UML, i casi d'uso sono classi e la generalizzazione è la relazione di ereditarietà tra due casi d'uso con cui un caso d'uso eredita tutte le proprietà e le relazioni di un altro caso d'uso.



Euristica per il controllo incrociato di casi d'uso e oggetti partecipanti

1. Quali casi d'uso creano questo oggetto (ad esempio, durante quali casi d'uso vengo inseriti i valori degli attributi dell'oggetto nel sistema)?
2. Quali attori possono accedere a queste informazioni?
3. Quali casi d'uso modificano e distruggono questo oggetto (ovvero, quali casi d'uso modificano o rimuovono queste informazioni dal sistema)?
4. Quale attore può avviare questi casi d'uso?
5. È necessario questo oggetto (ovvero, esiste almeno un caso d'uso che dipende da queste informazioni?)

Gestione dei requirements elicitation
Rappresentare i casi d'uso con dei modelli non costituisce tuttavia un punto finale. Oltre a questo, gli sviluppatori devono ancora ottenere requisiti dagli utenti e raggiungere un accordo.
In seguito sono eleganti dei metodi per ottenere informazioni dagli utenti e negoziare un accordo con un cliente.
Joint Application Design (JAD)
Un metodo sviluppato da IBM alla fine degli anni 70. Il lavoro del raccoglimento dei requirements elicitation viene svolto in un'unica sessione di seminario a cui partecipano tutti le parti interessate. Utenti, clienti, sviluppatori e un leader di cessione siedono insieme in una stanza per presentare il loro punti di vista, ascoltare altri punti di vista, negoziare e giungere a una soluzione accettabile. Il risultato del seminario, il documento JAD finale, è un documento di requirements specification completo che include le definizioni di elementi di dati, flussi di lavoro e schermate di interfaccia. Quindi il documento JAD finale rappresenta un accordo tra utenti, clienti e sviluppatori e riduce di conseguenza al minimo le modifiche ai requisiti più avanti nel processo di sviluppo.
Mantenimento della tracciabilità
La tracciabilità è la capacità di seguire la vita di un requisito. Ciò include la traccia da dove provengono i requisiti (chi lo ha originato, a quale cliente deve rivolgersi), a quali aspetti del sistema e del progetto influenza (ad esempio, quali componenti realizzano il requisito, quale test ne verifica la realizzazione). Questo consente agli sviluppatori di dimostrare che il sistema è completo, ai tester di dimostrare che il sistema è conforme a suoi requisiti, ai progettisti di registrare la logica alla base del sistema e ai manutentori per valutare l'impatto del cambiamento.
L'approccio più semplice per mantenere la tracciabilità consiste nell'utilizzare riferimenti incrociati tra documenti, modelli e artefatti da codice. Ogni singolo elemento (ad es. Requisito, componente, classe, operazione, test case) è identificato da un numero univoco. Le dipendenze vengono quindi documentate manualmente come riferimento incrociato testuale contenente il numero dell'elemento sorgente e il numero dell'elemento target.
Documentazione Requisiti Elicitation
I risultati dell'elaborazione dei requisiti e delle attività di analisi sono documentati nel documento di analisi dei requisiti (RAD). Questo documento descrive completamente il sistema in termini di requisiti funzionali e non funzionali. Le persone che utilizzeranno questo RAD per informarsi sono il cliente, gli utenti, la gestione del progetto, gli analisti di sistema. La prima parte del documento, compresi gli use case e i requisiti non funzionali, è scritta durante Requirements Elicitation. Mentre la formalizzazione della specifica in termini di modelli di oggetti viene scritta durante l'analisi.
Il Template adottato per la rad è:

1. Introduzione
   - 1.1 Scopo del sistema ;
   - 1.2 Portata del sistema;
   - 1.3 Obiettivi e i criteri del successo del progetto;
   - 1.4 Definizioni, acronimi, e abbreviazioni;
   - 1.5 Riferimenti;
   - 1.6 Panoramica;
2. Sistema Corrente
3. Sistema Proposto
   - 3.3.1 Panoramica;
   - 3.3.2 Requisiti funzionali;
   - 3.3.3 Requisiti non funzionali;
   - 3.3.4 Sopportabilità;
   - 3.3.5 Implementazione;
   - 3.3.6 Interfacce;
   - 3.3.7 Pacchetti;
   - 3.3.8 Legale;
   - 3.4 Modelli di sistema
     - 3.4.1 Scenari;
     - 3.4.2 Modelli di casi d'uso;
     - 3.4.3 Object Model;
     - 3.4.4 Modelli Dinamici
     - 3.4.5 Interfaccia Utente—>percorso di navigazione e mockup
4. Glossario

La prima sezione della RAD è un'introduzione. Il suo scopo è fornire una breve panoramica della funzione del sistema e delle ragioni del suo sviluppo, della sua portata e riferimenti al contesto di sviluppo. Esso compone anche gli obiettivi e i criteri di successo del progetto.
La seconda sezione, Sistema Corrente, descrive lo stato attuale delle cose. Se il nuovo sistema sostituirà un sistema esistente, questa sezione descrive le funzionalità e i problemi del sistema corrente. Altrimenti, questa sezione descrive come vengono ora eseguite le attività supportate dal nuovo sistema. Nel caso di FRIEND, l'attuale procedimento è basato su cartaceo: gli spedizionieri tengono traccia delle assegnazioni delle risorse comprando i moduli. La comunicazione tra spedizionieri e ufficiali di campo avviene via radio. L'attuale sistema richiede un'elevata documentazione e costi di gestione che FRIEND intende ridurre.
La terza sezione, Sistema Proposto, documenta la richiesta di requisiti e il modello di analisi del nuovo sistema. E' diviso in quattro sottosezioni:

1. Panoramica presenta una panoramica funzionale del sistema;
2. I requisiti funzionali descrivono le funzionalità del sistema;
3. I requisiti non funzionali descrivono i requisiti atti a ottimizzare il sistema, a livello di utente. Come usabilità, prestazioni, ecc..
4. I modelli di sistema descrivono scenari, casi d'uso, modello a oggetti e modelli dinamici per il sistema. Le sottosezioni Object Model e Modelli Dinamici sono descritti nella fase di analisi.