Indice

4 – Ingegneria dei requisiti

Il processo di ricerca, analisi, documentazione e verifica di questi servizi e vincoli è chiamato **ingegneria dei requisiti** (RE, *Requirements Engineering*).

Il termine **requisito** viene abusato con conseguenti incoerenze da parte dell'industria. I vari significati che acquisisce sono:

- Un requisito è una formulazione astratta e di alto livello di un servizio che il sistema dovrebbe fornire o di un vincolo del sistema;
- Un requisito è una definizione formale e dettagliata di una funzione del sistema.

I **problemi**, nell'ingegneria dei requisiti, che nascono da queste incoerenze derivano dalla mancanza di una separazione netta tra questi diversi livelli di descrizione:

1. **Requisiti dell'utente**: scritti nel linguaggio naturale e rappresentati da diagrammi, sono tutti i servizi che il sistema dovrebbe fornire e i vincoli sotto cui deve operare.
2. **Requisiti del sistema**: scritto in linguaggio tecnico, rappresenta una descrizione dettagliata delle funzioni, dei servizi e dei vincoli operativi del sistema software. Inoltre, dovrebbe esserci un documento che definisce esattamente che cosa deve essere implementato.

La figura seguente illustra un **esempio** sulla distinzione tra i requisiti dell'utente e quelli del sistema. È tratto da un sistema informativo medico per la cura di pazienti con problemi di salute mentale, chiamato Mentcare. Esso è stato approfondito e studiato, come caso di studio, nel primo capitolo (Capitolo 1, paragrafo "Sistema informativo di una clinica psichiatrica").

User requirements definition

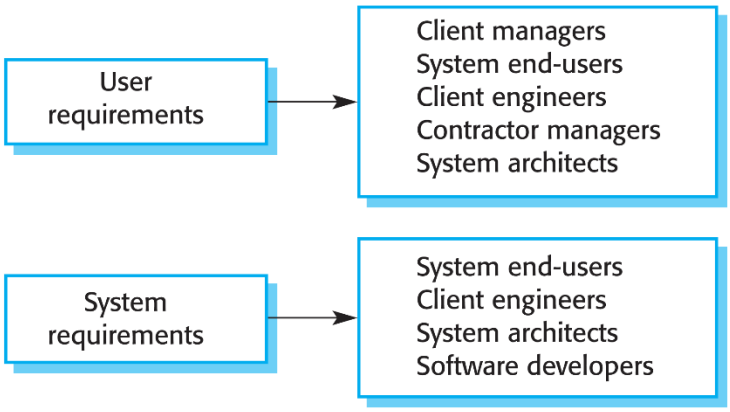
1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Dalla figura è possibile notare come il requisito dell'utente è generico. Al contrario, i requisiti del sistema sono informazioni più specifiche sui servizi e sulle funzioni del sistema.

Si propone uno studio più dettagliato sui diversi tipi di lettori dei due tipi di requisiti:



I vari tipi di lettori illustrati in figura, sono esempi di stakeholder del sistema. Gli **stakeholder** del sistema includono chiunque sia influenzato dal sistema in qualche maniera e chiunque abbia un legittimo interesse verso di esso. Gli stakeholder possono essere utenti finali del sistema, manager o persone esterne.

Per **esempio**, gli stakeholder per il sistema Mentcare sono:

1. I **pazienti** che devono essere curati;
2. I **dottori** che curano i pazienti;
3. Le **infermiere** che assistono i dottori;
4. Il **personale medico** che gestisce gli appuntamenti;
5. Il **personale informatico** che gestisce le manutenzioni del sistema;
6. Un **direttore etico** che assicura che il sistema soddisfi la normativa etica;
7. I **manager** dell'assistenza sanitaria;
8. Il **personale addetto alle cartelle cliniche**.

4.1 – Requisiti funzionali e non funzionali

I requisiti dei sistemi software sono spesso divisi in requisiti funzionali e non funzionali:

1. **Requisiti funzionali**: sono definizioni di servizi che il sistema deve fornire; indicano come il sistema dovrebbe reagire a particolari input e come dovrebbe comportarsi in particolari situazioni. Talvolta è necessario scrivere anche cosa il sistema non dovrebbe fare.
2. **Requisiti non funzionali**: sono vincoli sulle funzioni o servizi offerti dal sistema. Includono vincoli temporali e sul processo di sviluppo e vincoli imposti dagli standard.

I requisiti non sono indipendenti e spesso un requisito genera o limita altri requisiti. I requisiti del sistema pertanto non specificano semplicemente le funzioni o i servizi che servono al sistema, ma anche le funzionalità necessarie per garantire che questi servizi/funzioni siano realizzati in modo efficace.

4.1.1 – Requisiti funzionali

I **requisiti funzionali** descrivono ciò che il sistema dovrebbe fare. Se espressi come requisiti **utente**, i requisiti funzionali devono essere scritti nel linguaggio naturale, così che gli utenti del sistema ed eventuali manager, possano capirli. Al contrario, i requisiti funzionali **del sistema**, hanno l'obbiettivo di estendere i requisiti dell'utente e sono destinati agli sviluppatori del sistema poiché le funzioni (input, output ed eccezioni) sono descritte dettagliatamente.

L'**imprecisione** nella specifica dei requisiti può provocare discussioni tra clienti e sviluppatori del software.

Infatti, in linea teorica, le specifiche dei requisiti funzionali di un sistema dovrebbero essere complete e coerenti:

- **Complete**, ovvero tutti i servizi richiesti dagli utenti devono essere definiti;
- **Coerenti**, i requisiti non devono avere definizioni contraddittorie.

Nonostante siano principi basilari, nei software di grandi dimensioni è difficile rispettarli. I motivi sono molteplici: con sistemi grandi e complessi, gli errori e omissioni aumentano; gli stakeholder sono numerosi, ognuno con richieste, aspettative ed esigenze differenti.

4.1.2 – Requisiti non funzionali

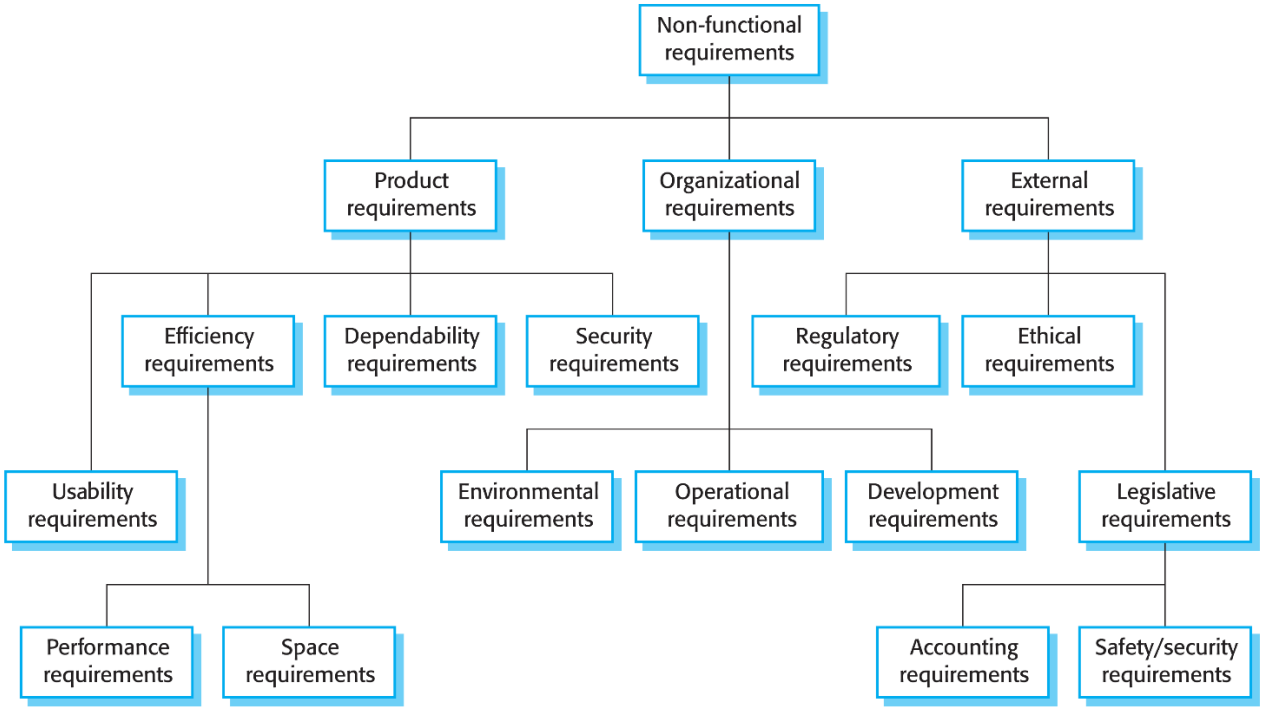
I **requisiti non funzionali** sono requisiti che **non** riguardano direttamente specifici servizi forniti dal sistema ai suoi utenti. Solitamente specificano o limitano le caratteristiche del sistema nel suo complesso. Si possono riferire a proprietà del sistema (affidabilità, tempi di risposta e l'uso della memoria) oppure definiscono vincoli sull'implementazione, come la capacità dei dispositivi di I/O.

Nel caso in cui, i requisiti non funzionali **non vengono soddisfatti**, l'intero sistema potrebbe risultare inutilizzabile. Per **esempio**, se un sistema di controllo degli aerei non soddisfa i requisiti di affidabilità, non potrà ottenere la certificazione di sicurezza necessaria per operare; se un sistema integrato per il controllo in tempo reale non raggiunge le prestazioni richieste, le sue funzioni di controllo non potranno operare correttamente.

L'**implementazione** di questi requisiti può interessare l'intero sistema:

1. I requisiti non funzionali possono **influenzare sull'intera architettura** di un sistema.
2. Un **singolo requisito non funzionale**, come quello della protezione, per esempio, potrebbe generare vari requisiti funzionali tra loro correlati che definiscono nuovi servizi del sistema, che si rendono necessari se il requisito non funzionale deve essere implementato. In aggiunta, **potrebbe generare requisiti che limitano quelli esistenti**; per esempio, potrebbe limitare l'accesso alle informazioni nel sistema.

La seguente figura mostra una classificazione dei requisiti non funzionali che possono derivare da caratteristiche richieste dal software, dall'organizzazione che sta sviluppando il software o da sorgenti esterne.



1. **Requisiti del prodotto**: specificano o limitano il comportamento del software. Per esempio, i requisiti di prestazioni che definiscono la velocità di esecuzione del programma e la quantità di memoria richiesta.
2. **Requisiti organizzativi**: sono requisiti generali del sistema che derivano dalle politiche e dalle procedure delle organizzazioni del cliente e dello sviluppatore. Per esempio, gli standard di lavorazione da seguire, i requisiti del processo di sviluppo che specificano il linguaggio di programmazione.
3. **Requisiti esterni**: tutti i requisiti che derivano da fattori esterni al sistema e al suo processo di sviluppo. Per esempio, i requisiti normativi.

Un **problema** comune dei requisiti funzionali è che gli stakeholder propongono i requisiti come obbiettivi generici, per esempio facilità d'uso, capacità di ripristino del sistema dopo un malfunzionamento o rapida risposta agli utenti. Questi **obbiettivi vaghi causano problemi agli sviluppatori** perché danno adito alla libera interpretazione e alla successiva discussione quando il sistema viene consegnato.

Quando è possibile, bisognerebbe **descrivere i requisiti non funzionali quantitativamente**, in modo che possano essere verificati in maniera oggettiva. La figura seguente mostra una serie di possibili misurazioni da usare per specificare le proprietà non funzionali di un sistema.

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Megabytes/Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

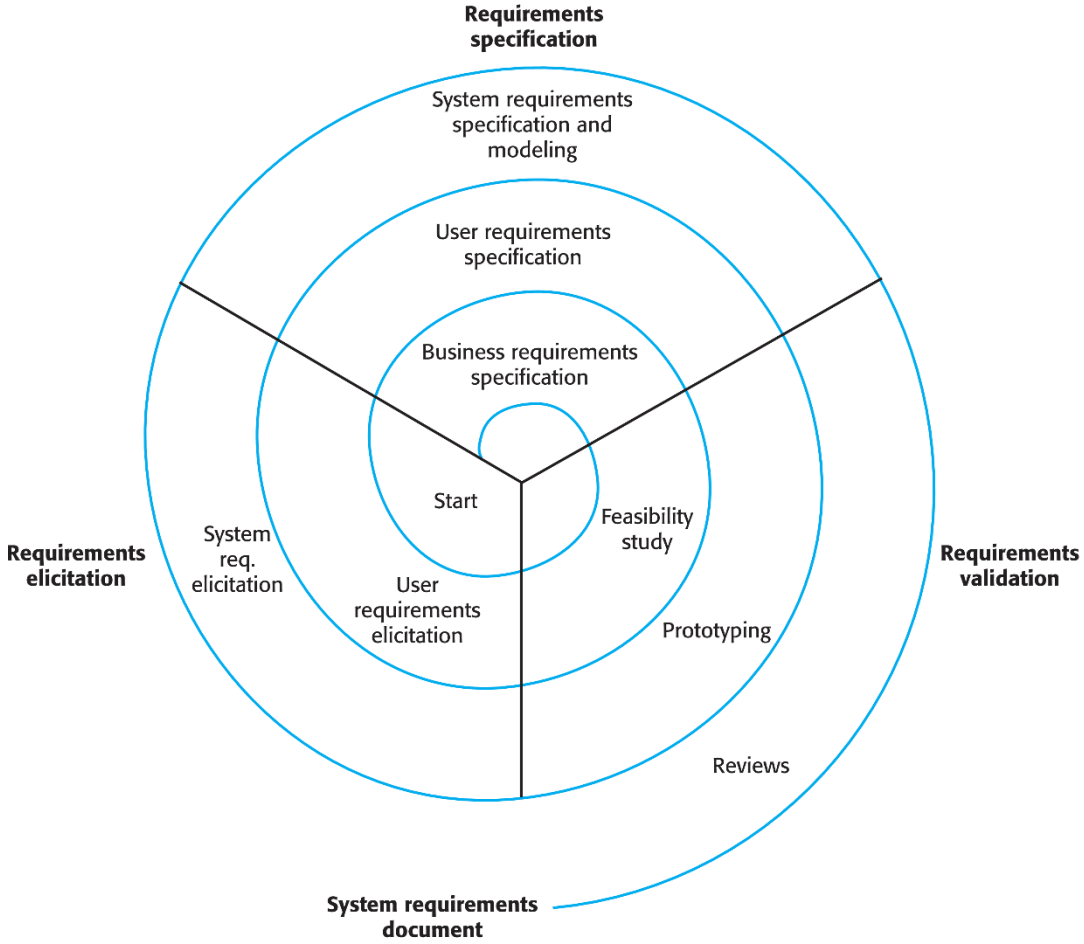
Purtroppo, **nella pratica** i clienti trovano difficile tradurre i propri obbiettivi in requisiti misurabili. Per alcuni obbiettivi, come la manutenibilità, non esistono semplici misurazioni che possano essere effettuate. In altri casi, i clienti non riescono a scegliere un numero che possa definire l'affidabilità del sistema.

Infine, dato che i **costi** della verifica oggettivi di requisiti non funzionali misurabili possono essere molto alti, il **cliente potrebbe considerarli ingiustificati**.

4.2 – Processi di ingegneria dei requisiti

L'ingegneria dei requisiti è formata da tre attività chiave: scoperta dei requisiti attraverso l'interazione con gli stakeholder (deduzione e analisi), conversione dei requisiti in una forma standard (specifica) e controllo che i requisiti definiscano realmente il sistema voluto dal cliente (convalida).

La seguente figura mostra come queste attività sono interlacciate.



Le attività sono organizzate come un processo iterativo attorno a una spirale. L'output del processo di ingegneria dei requisiti è un documento dei requisiti del sistema. La quantità di tempo e gli sforzi dedicati a ogni attività in una iterazione dipende dallo stadio del processo generale, dal tipo di sistema che si sta sviluppando e dal budget disponibile.

Ovviamente, nelle prime fasi del processo saranno spesi più sforzi per comprendere i requisiti aziendali di alto livello, i requisiti non funzionali e i requisiti dell'utente del sistema. Nelle fasi finali, i giri più esterni della spirale, gli sforzi saranno dedicati alla deduzione e alla comprensione dei requisiti non funzionali e alla definizione più dettagliata dei requisiti del sistema.

È possibile uscire dalla aspirale dopo aver raccolto tutti o alcuni dei requisiti dell'utente.

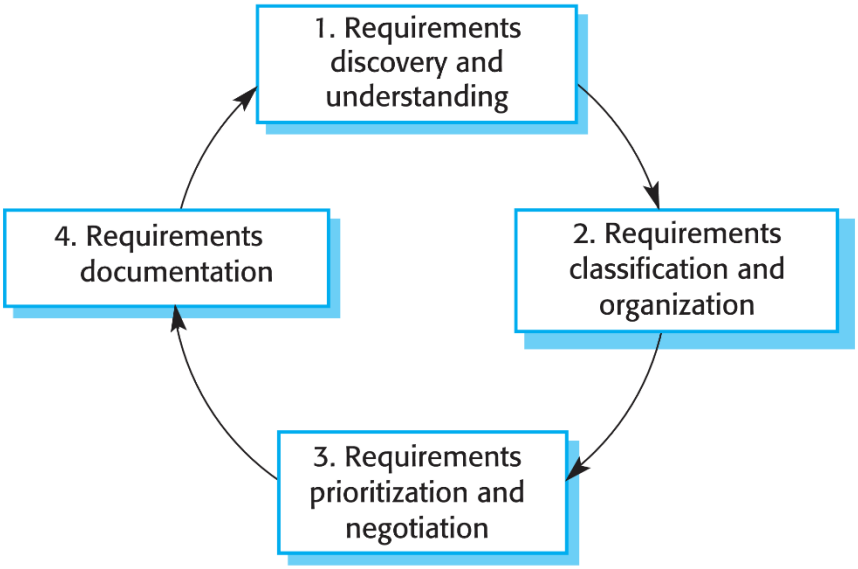
4.3 – Deduzione dei requisiti

L'**obbiettivo** del processo di deduzione dei requisiti consiste nel capire il lavoro svolto dagli stakeholder e come essi potrebbero utilizzare un nuovo sistema e supporto del loro lavoro. Durante la fase di deduzione dei requisiti, gli ingegneri del software lavorano con gli stakeholder.

La **deduzione** e l'**analisi dei requisiti** degli stakeholder è un **processo difficile** per alcune ragioni:

1. Gli **stakeholder non sanno cosa vogliono da un sistema informatico**, se non in termini molto generici. Essi trovano difficoltà, talvolta, ad esprimere cosa intendono far fare al sistema o potrebbero fare richieste irrealizzabili perché non sanno che cosa può essere realmente fatto con un sistema software.
2. Gli **stakeholder esprimono i requisiti nei propri termini** e gli ingegneri dei requisiti potrebbero non capire questi requisiti.
3. Diversi **stakeholder** hanno **requisiti diversi** che possono essere **espressi in modi differenti**.
4. **Fattori politici** possono **influenzare** i **requisiti del sistema**; per esempio, i manager potrebbero inserire requisiti che aumentano la loro influenza nell'organizzazione.
5. L'**ambiente economico e aziendale** in cui si analizzano i requisiti **è dinamico**; esso cambia durante il processo di analisi. Quindi, l'importanza di un requisito può cambiare oppure possono emergere nuovi requisiti da nuovi stakeholder.

Un modello del processo di deduzione e analisi è mostrato nella seguente figura e di seguito vengono elencate le attività:



1. **Scoperta e comprensione dei requisiti** (*Requirements discovery and understanding*): il processo di interazione con gli stakeholder del sistema con l'obbiettivo di scoprire i loro requisiti;
2. **Classificazione e organizzazione dei requisiti** (*Requirements classification and organization*): l'attività che raggruppa i requisiti correlati e li organizza in gruppi coerenti;

3. **Negoziazione e priorità dei requisiti** (*Requirements prioritization and negotiation*): quando sono coinvolti più stakeholder, i requisiti possono essere in conflitto. Questa attività ha l'obbiettivo di dare una priorità ai requisiti;
4. **Documentazione dei requisiti** (*Requirements documentation*): i requisiti vengono documentati e diventano l'input del successivo giro della spirale (presentata nel paragrafo 4.2 di questo capitolo).

La figura nella pagina precedente, mostra che la deduzione e l'analisi dei requisiti sono un processo iterativo con un continuo feedback da ciascuna attività alle altre.

La comprensione dei requisiti da parte dell'analista migliora a ogni giro del ciclo. Il ciclo termina quando il documento dei requisiti è completato.

4.3.1 – Tecniche di deduzione dei requisiti

Interviste

Le **interviste formali o informali** con gli stakeholder del sistema sono parte della gran parte dei processi di ingegneria dei requisiti. In queste interviste i team di ingegneria fanno domande agli stakeholder sul sistema che utilizzano e sul sistema che deve essere sviluppato; dalle loro risposte ottengono i requisiti.

Esistono **due tipi** di interviste:

1. **Interviste chiuse**, in cui lo stakeholder risponde a un insieme predefinito di domande;
2. **Interviste aperte**, in cui non c'è niente di predefinito. Quindi, il team esamina vari problemi degli stakeholder.

Chiaramente, con le interviste aperte è difficile ottenere grandi risultati. Molte interviste iniziano da alcune domande preliminari e poi si focalizzano sul sistema che deve essere sviluppato.

Dedurre le conoscenze attraverso le interviste può essere difficile per due ragioni:

1. Tutti gli **specialisti** dell'applicazione usano una **terminologia** e un **gergo specifici**. Utilizzano termini così precisi e minuziosi che gli **ingegneri** dei requisiti spesso **non** riescono a **capire**.
2. Gli **stakeholder** hanno **difficoltà a spiegare** alcuni concetti familiari, **oppure** pensano che siano **così importanti** che è **inutile citarle**. Per esempio, per un bibliotecario è ovvio che tutti i nuovi libri debbano essere catalogati prima di aggiungerli alla biblioteca, ma potrebbe non esserlo per l'intervistatore tanto da non includerlo tra i requisiti.

Le interviste non sono una tecnica efficace per dedurre conoscenze sui requisiti e sui vincoli organizzativi, perché ci sono delle sottili relazioni di potere tra i vari stakeholder dell'organizzazione. Le strutture organizzative ufficiali raramente rispecchiano la realtà di chi veramente prende le decisioni, ma gli intervistati potrebbero non voler rivelare le vere strutture a un estraneo.

Gli **intervistatori efficaci hanno due caratteristiche**:

1. Sono di **larghe vedute**, **evitano** le **idee preconcette** sui requisiti e sono **disposti ad ascoltare** gli stakeholder. Se lo stakeholder salta fuori con un requisito sorprendente, sono pronti a cambiare idea sul sistema.
2. **Inducono l'intervistato** a iniziare la discussione con una domanda, una proposta di requisiti o suggeriscono di lavorare insieme a un prototipo del sistema.

Ovviamente, la **semplice intervista** può portare alla **mancanza di informazioni essenziali**, per questo motivo questa tecnica di deduzione dovrebbe essere sempre **utilizzata insieme ad altre**.

Etnografia

Un motivo per cui molti sistemi software sono consegnati, ma mai utilizzati, è che non viene valutato nel modo giusto l'importanza di come i fattori sociali e organizzativi influenzano il funzionamento pratico di un sistema.

Quindi, è **molto importante**, durane il processo di ingegneria del software, **cercare di capire i problemi** sociali e organizzativi **che influiscono sull'uso del sistema**.

L'**etnografia** è una **tecnica di osservazione** utilizzata per capire i processi operativi e per derivare i requisiti del software a supporto di tali processi. Un analista si immerge nell'ambiente di lavoro in cui il sistema sarà usato, osserva il lavoro quotidiano prendendo nota dei compiti in cui i partecipanti sono coinvolti.

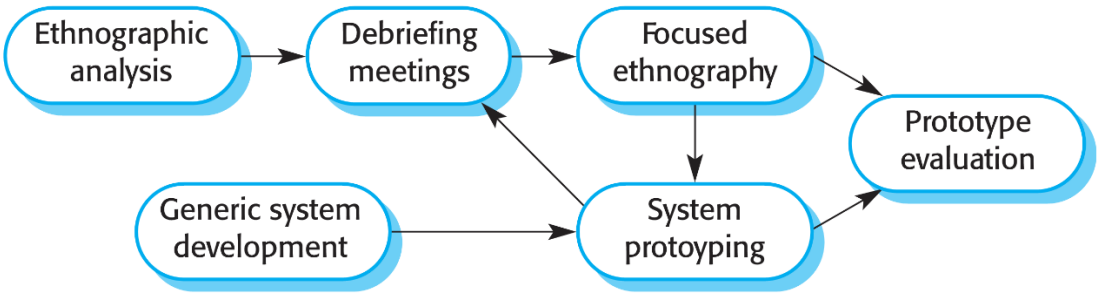
Il valore dell'etnografia è l'aiuto che fornisce all'analista nella scoperta dei requisiti impliciti del sistema.

Lucy Suchman (Link biografia) fu **tra i primi a usare l'etnografia** per studiare il lavoro d'ufficio. Scoprì che le attività lavorative reali erano molto più ricche, complesse e dinamiche dei semplici modelli ipotizzati dai sistemi di automazione dell'ufficio.

L'**etnografia** è **efficace** **per scoprire due tipi di requisiti**:

1. **Requisiti** che **derivano** dal **modo in cui le persone lavorano realmente**, anziché dal modo in cui le definizioni di processo dicono che dovrebbero lavorare;
2. I **requisiti** che **derivano dalla cooperazione e dalla consapevolezza** delle attività di altre persone.

L'etnografia può essere combinata con la prototipazione, come mostrato in figura. La prima fornisce informazioni per lo sviluppo del prototipo, in modo da ridurne i cicli di perfezionamento. Mentre la seconda, concentra l'etnografia sull'identificazione di problemi e domande da discutere poi con l'etnografo, il quale dovrebbe quindi cercare le relative risposte nella fase successiva dello studio.



Anche qui, dato che questo approccio si focalizza sull'utente finale, non è adatto a scoprire requisiti organizzativi o di dominio più ampi. Pertanto, **l'etnografia dovrebbe essere utilizzata come una di tante tecniche di deduzione dei requisiti**.

4.3.2 – Storie e scenari

Storie e scenari vengono utilizzati quando si intervistano gruppi di stakeholder per discutere il sistema con altri stakeholder e sviluppare requisiti di sistema più specifici.

Le **storie** sono testi narrativi e presentano una descrizione di alto livello del modo in cui il sistema è utilizzato; gli **scenari** di solito sono strutturati con informazioni specifiche raccolte come input e output. Le storie sono più efficaci per preparare una "visione d'insieme". Parti delle storie possono poi essere sviluppate più dettagliatamente e rappresentate come scenari.

Il **vantaggio** delle **storie** è che chiunque può facilmente mettersi in relazione con esse. Questo è utile per ottenere informazioni da un pubblico più vasto di quello che si potrebbe realisticamente intervistare.

Uno **scenario** inizia con la descrizione dell'interazione. Durante il processo di deduzione, si aggiungono dettagli per creare una descrizione più completa dell'interazione. Nella sua forma più generale, uno scenario include:

1. Una descrizione di ciò che il sistema e gli utenti si aspettano quando inizia lo scenario;
2. Una descrizione del flusso normale di eventi nello scenario;
3. Una descrizione di cosa può causare errori e come possono essere gestiti i problemi risultanti;
4. Informazioni su altre attività che potrebbero svolgersi contemporaneamente;
5. Una descrizione dello stato del sistema al termine dello scenario.

Come le storie, anche gli scenari possono essere utilizzati per agevolare la discussione con gli stakeholder che potrebbero avere modi differenti di raggiungere lo stesso risultato.

4.4 – Specifica dei requisiti

La **specifica dei requisiti** è il processo di scrivere i requisiti dell'utente e del sistema in un documento. Gli stakeholder interpretano i requisiti in modi differenti, e spesso includono incoerenze e conflitti intrinseci.

Il **documento dei requisiti** include tabelle e diagrammi appropriati. La seguente tabella riassume le possibili notazioni per scrivere i requisiti del sistema:

Frasi del linguaggio naturale	I requisiti sono scritti utilizzando frasi numerate nel linguaggio naturale. Ogni frase deve esprimere un requisito.
Linguaggio naturale strutturato	I requisiti sono scritti nel linguaggio naturale secondo una forma o modello standard. Ogni campo in un modello fornisce le informazioni su un aspetto del requisito.
Notazioni grafiche	Modelli grafici, corredati da annotazioni di testo, sono usati per definire i requisiti funzionali del sistema. Sono comunemente utilizzati i casi d'uso e i diagrammi di sequenza del linguaggio UML (Unified Modelling Language).
Specifiche matematiche	Sono notazioni basate su concetti matematici, come gli insiemi o le macchine a stati finiti. Sebbene queste specifiche non ambigue possano ridurre l'ambiguità del documento dei requisiti, molti clienti non capiscono una specifica formale; non possono verificare che il documento dei requisiti rappresenta ciò che essi vogliono e, quindi, sono riluttanti ad accettarlo come parte del contratto.

Il **documento dei requisiti** non dovrebbe includere dettagli sull'architettura o sulla progettazione del sistema. Quindi non si dovrebbe utilizzare un gergo tecnico, notazioni strutturate o formali. Si dovrebbe **scrivere i requisiti dell'utente nel linguaggio naturale**, con **semplici tabelle**, **moduli** e **diagrammi intuitivi**.

I **requisiti del sistema** sono utilizzati dagli ingegneri del software come **base di partenza per la progettazione del sistema**; aggiungono dettagli e spiegano come il sistema dovrebbe realizzare i requisiti dell'utente.

I requisiti del sistema descrivono soltanto il comportamento esterno del sistema e i suoi vincoli operativi. Non è possibile né auspicabile escludere tutte le informazioni sulla progettazione per diverse ragioni:

1. Potrebbe essere necessario progettare un'architettura iniziale del sistema per agevolare la definizione della struttura delle specifiche dei requisiti.
2. In molti casi, i sistemi devono interagire con altri sistemi esistenti. Questo vincola la progettazione e impone requisiti al nuovo sistema.
3. Potrebbe essere necessario l'uso di una specifica architettura per soddisfare requisiti non funzionali, come la programmazione a N-versioni per raggiungere l'affidabilità.

4.4.1 – Specifica nel linguaggio naturale

Il linguaggio naturale è espressivo, intuitivo e universale. È anche potenzialmente vago e ambiguo, e la sua interpretazione dipende dal background culturale del lettore.

Il linguaggio naturale continuerà ad essere il modo più utilizzato di specificare i requisiti del sistema e del software.

Per ridurre al minimo le incomprensioni si possono seguire delle semplici linee guida:

1. Si inventa un formato standard e ci si assicura che tutte le definizioni dei requisiti siano conformi a questo formato. In questo modo si riduce il rischio di omissioni e semplifica il controllo dei requisiti. Se possibile, si dovrebbe scrivere i requisiti con una o due frasi nel linguaggio naturale.
2. Si utilizza con coerenza il linguaggio per distinguere un requisito obbligatorio da un requisito desiderabile. I requisiti obbligatori sono quelli che il sistema deve supportare e si scrivono utilizzando l'indicativo "deve". I requisiti desiderabili non sono essenziali e si scrivono utilizzando il condizionale "dovrebbe".
3. Si utilizza la formattazione del testo per mettere in evidenza parti di un requisito.
4. Non si dà per scontato che i lettori capiscano il linguaggio tecnico dell'ingegneria del software. Si dovrebbe dunque evitare termini del gergo, abbreviazioni e acronimi.
5. Si tenta di associare una logica al requisito di ciascun utente. La logica dovrebbe spiegare il motivo per cui il requisito è stato incluso e indicare chi lo ha proposto (la fonte del requisito), in modo che si sappia chi consultare se il requisito dovesse essere modificato.

4.4.2 – Specifiche strutturate

Il linguaggio naturale strutturato è un modo di scrivere i requisiti secondo una forma standard, anziché come testo libero. Questo approccio mantiene gran parte dell'espressività e della comprensibilità del linguaggio naturale, ma garantisce una certa uniformità delle specifiche.

Le notazioni del linguaggio strutturato usano schemi per specificare i requisiti dei sistemi. La specifica può usare costrutti del linguaggio di programmazione per mostrare alternative e iterazioni e può mettere in evidenza elementi chiave utilizzando colori o caratteri differenti.

Se si utilizza un modulo standard per specificare i requisiti funzionali, dovrebbero essere incluse le seguenti informazioni:

1. Una descrizione della funzione o dell'entità che si sta specificando;
2. Una descrizione dei suoi input e delle origini di questi input;
3. Una descrizione dei suoi output e delle destinazioni di questi output;
4. Informazioni sui dati necessari per i calcoli o su altre entità che sono richieste dal sistema;
5. La descrizione dell'azione da seguire;
6. Se si utilizza un metodo funzionale, una precondizione che spieghi cosa deve verificarsi prima che la funzione sia chiamata, e una post condizione che specifichi cosa avviene sicuramente dopo che la funzione è stata chiamata;
7. Una descrizione degli effetti collaterali dell'operazione, se ce ne sono.

L'uso di **specifiche strutturate risolve alcuni problemi delle specifiche scritte nel linguaggio naturale**. La variabilità è ridotta e i requisiti sono organizzati in modo più efficace. Tuttavia, è **difficile scrivere i requisiti in modo non ambiguo**, specialmente quando bisogna specificare calcoli complessi.

Per **risolvere** questo **problema** è possibile aggiungere informazioni supplementare al linguaggio naturale, per esempio tabelle o modelli grafici del sistema, che possono spiegare come devono essere svolti i calcoli, come cambia lo stato del sistema, come gli utenti interagiscono con il sistema e come vengono effettuate le sequenze di operazioni.

Le **tabelle sono utili** quando c'è una serie di possibili situazioni alternative ed è necessario descrivere le azioni da intraprendere.

4.5 – Il diagramma dei casi d'uso

Il **diagramma dei casi d'uso** è un diagramma che esprime un comportamento desiderato o offerto. L'oggetto che viene esaminato è solitamente un sistema o una sua parte.

Si individuano:

- **Chi** o che cosa ha a che fare con il sistema (**attore**);
- **Che cosa** l'attore può fare (**caso d'uso**).

Solitamente, il diagramma dei casi d'uso è il primo tipo di diagramma ad essere creato in un processo o ciclo di sviluppo, nell'ambito dell'analisi dei requisiti.

I **casi d'uso** modellano graficamente i requisiti funzionali e specifica cosa ci si aspetta da un sistema. Tuttavia, "nasconde" come il sistema lo implementa dato che vengono rappresentate una sequenza di azioni che producono un risultato osservabile da un attore.

Viene **utilizzato** per: **descrivere i requisiti** (analisi iniziale), **convalidare l'architettura** e **verificare il sistema**.

I **desiderata** sono ciò che il cliente desidera. Per formalizzarli in requisiti, ci sono molte difficoltà che le rendono una grande sfida per gli ingegneri del software. Inoltre, una specifica errata o incompleta delle richieste del cliente è una delle cause principali del fallimento dei progetti software. Il diagramma e la modellazione dei casi d'uso aiutano l'interazione con il cliente e migliorano l'estrazione dei requisiti (funzionali).

I desiderata vengono raccolti da coloro che interagiscono con il cliente, ovvero gli attori. Successivamente, vengono stilati i casi d'uso, ovvero i compiti che eseguiscono.

Solitamente, un **attore** specifica un ruolo assunto da un utente o altra entità che interagisce col sistema nell'ambito di un'unità di funzionamento (caso d'uso). Esso è esterno al sistema e non necessariamente è umano, può essere anche un oggetto fisico, agente software, condizioni ambientali, ecc. Gli attori eseguono i casi d'uso ovviamente.

L'**individuazione degli attori** richiede di capire chi e che cosa interagisce col sistema, per capire quale ruolo ricopre. Alcune domande utili potrebbero essere:

- Chi/cosa usa il sistema?
- Che ruolo ha chi/cosa interagisce col sistema?
- In quale parte dell'organizzazione è utilizzato il sistema?
- Chi/cosa avvia il sistema?
- Chi supporterà e manterrà il sistema?
- Altri sistemi interagiscono col sistema?
- Ci sono funzioni attivate periodicamente? Esempio il backup.
- Chi/cosa ottiene o fornisce informazioni dal sistema?
- Un attore ha diversi ruoli? Lo stesso ruolo è assegnato a più attori?

Un **caso d'uso** è la specifica di una sequenza di azioni, incluse eventuali sequenze alternative e/o di errore che un sistema (o sottosistema) può eseguire interagendo con attori esterni. È qualcosa che un attore vuole il sistema faccia: è descritto dal punto di vista dell'attore ed è

causato da un'azione di un attore. Il nome, chiamato **etichetta**, dovrebbe essere basato su un verbo o su un sostantivo che esprime un avvenimento.

Un caso d'uso è sempre iniziato da un attore e un evento è sempre legato all'entità che lo ha generato. Esistono **due tipi di attore**:

- **Attore primario**, ovvero l'attore che inizia un caso d'uso;
- **Attore secondario**, ovvero l'attore che interagisce nell'ambito di quel caso d'uso.

Per **individuare un caso d'uso** è necessario un'operazione **iterativa**. Per individuare i casi d'uso possono essere utili le seguenti domande:

- Ciascun attore che funzioni si aspetta?
- Il sistema gestisce informazioni? Se sì, quali sono gli attori che provocano questo comportamento?
- Alcuni attori vengono informati quando il sistema cambia stato?
- Gli attori devono informare il sistema di cambiamenti improvvisi?
- Alcuni eventi esterni producono effetti sul sistema?
- Quali casi d'uso mantengono il sistema?
- I requisiti funzionali sono tutti coperti dai casi d'uso?

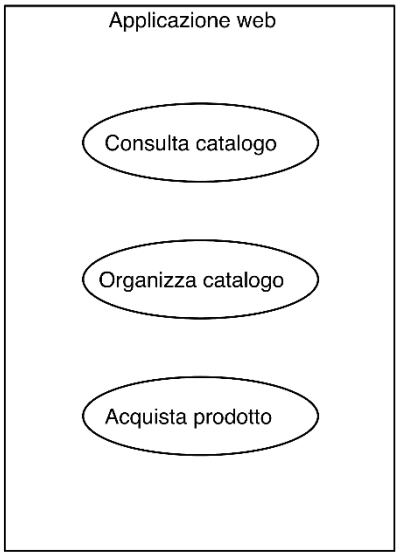
Per **recuperare** informazioni su un caso d'uso, si possono usare le seguenti fonti:

- Interviste con gli esperti del dominio;
- Bibliografia del dominio del sistema;
- Sistema già esistenti;
- Conoscenza personale del dominio.

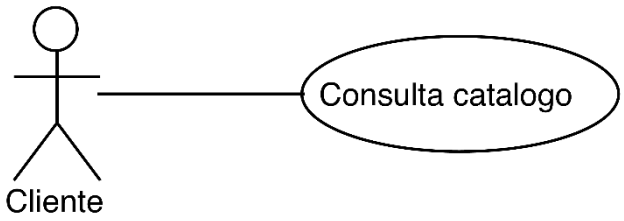
Per **descrivere un caso d'uso**, si utilizzare una descrizione in modo generico e sequenziale il flusso di eventi, ovvero descrivere prima la precondizione (stato iniziale del sistema) e poi elencare la sequenza di passi. Si descrivono le interazioni con gli attori e i messaggi scambiati, si aggiungono eventuali punti di estensione e si descrive in modo chiaro, preciso e breve.

4.5.1 – Elementi del diagramma dei casi d'uso

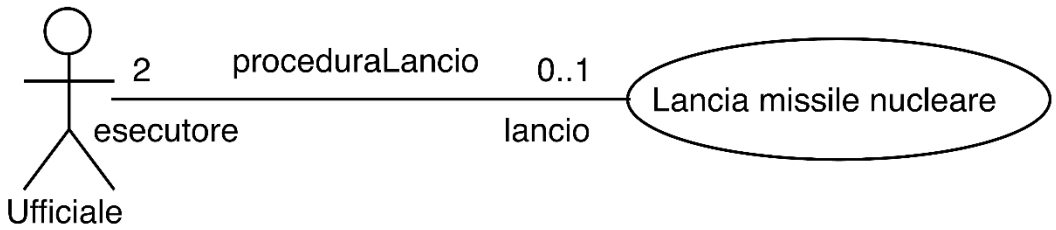
Il primo elemento del diagramma è il **sistema** che delimita l'argomento del diagramma, specificando i confini del sistema. Per esempio:



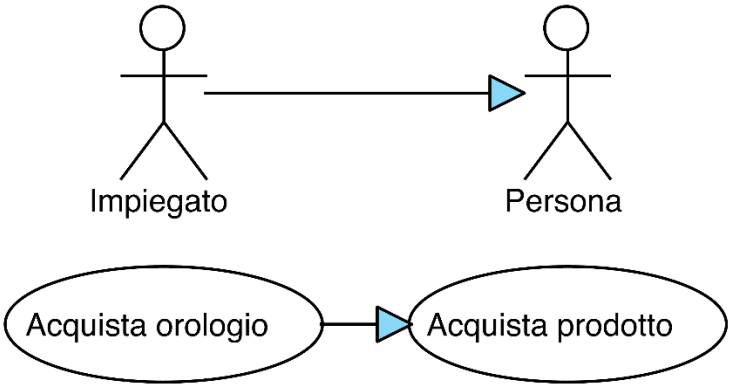
Il secondo elemento è l'**associazione** che collega gli attori ai casi d'uso. Un attore si può associare solo a casi d'uso, classi e componenti (che verranno eventualmente associati ad altri casi d'uso, classi e componenti). Un caso d'uso non dovrebbe essere associato ad altri casi d'uso riguardanti lo stesso argomento. Per esempio:



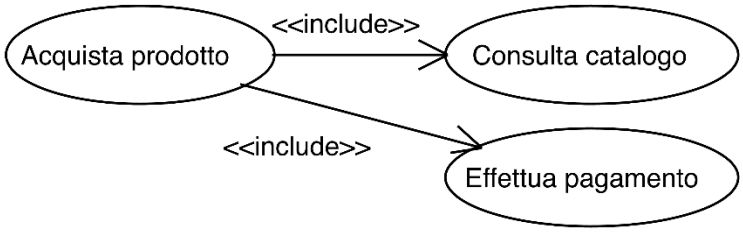
Un altro esempio in cui ci sono alcune caratteristiche opzionali comuni, come il nome, la molteplicità e i ruoli:



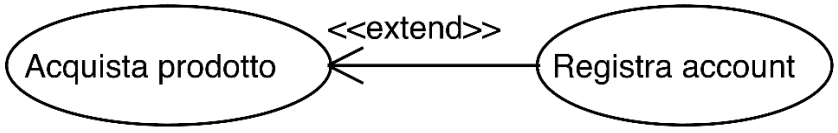
Il terzo elemento è la **generalizzazione** che collega un attore o caso d'uso ad un altro più generale. Il figlio può sostituire il genitore dovunque questi appaia. Per esempio:



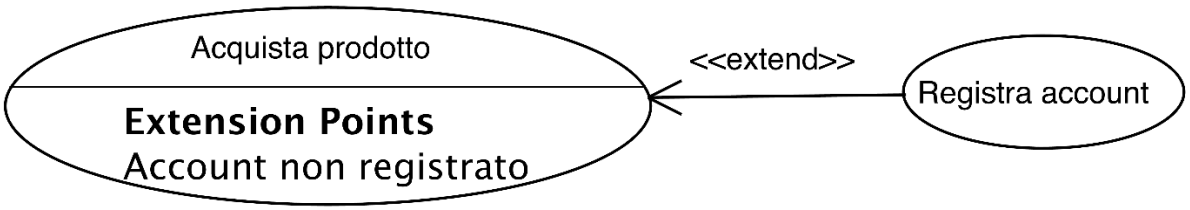
Il quarto elemento è **include**, ovvero una dipendenza tra casi d'uso. Il caso *incluso* fa parte del comportamento di quello che lo include. L'inclusione non è opzionale ed avviene in ogni istanza del caso d'uso. La corretta esecuzione del caso d'uso che include dipende da quella del caso d'uso incluso. Non si possono fermare cicli di include ed è usato per riutilizzare parti comuni a più casi d'uso. Per esempio:



Il quinto elemento è **extend**, ovvero una dipendenza tra casi d'uso. Il caso d'uso che estende (client) specifica un incremento di comportamento a quello esteso (supplier). Si tratta di comportamento supplementare ed opzionale che gestisce casi particolari o non standard. Diverso dall'elemento generalizzazione tra casi d'uso, in una generalizzazione entrambi i casi d'uso sono ugualmente significativi; in un extend, il client non ha necessariamente senso se preso da solo. Per esempio:



Il sesto elemento è **extension points**, ovvero un caso d'uso raggiunto da almeno un extend può opzionalmente visualizzare i propri extension points. Specifica i punti e/o le condizioni dell'esecuzione in cui il comportamento viene esteso. Se gli extension points sono molti, alcuni tool possono supportare la rappresentazione a rettangolo (i casi d'uso sono classificatori). Per esempio:



Ci sono alcune **differenze** tra **include** e **extend**:

1. Include specifica il comportamento obbligatorio, mentre extend specifica il comportamento supplementare, o varianti.
2. Nell'include la freccia va dal caso d'uso che include verso quello incluso, mentre nell'extend la freccia va dal caso d'uso che estendo verso quello esteso.
3. Sono entrambi costrutti utili, ma non se ne deve abusare, o la leggibilità ne risente.

Per concludere, si fa una riflessione per capire **se sono sufficienti o meno questi diagrammi**. Spesso nasce l'esigenza di abbinare i diagrammi dei casi d'uso a **specifiche testuali** più formali. I diagrammi dei casi d'uso **non sono adatti per questi scopi**:

- Mostrare la sequenza temporale delle operazioni;
- Mostrare lo stato del sistema e degli attori prima e dopo l'esecuzione del caso d'uso.

4.5.2 – Specifiche dei casi d'uso

Ogni caso d'uso ha un **nome** e una **specifica**.

La **specifica** è composta da:

- **Precondizioni**. Condizioni che devono essere vere prima che il caso d'uso si possa eseguire;
- **Sequenza degli eventi**. I passi che compongono il caso d'uso;
- **Post condizioni**. Condizioni che devono essere vere quando il caso d'uso termina l'esecuzione.

Un **esempio** è il seguente:

Nome del caso d'uso	**Caso d'uso: PagamentoIVA**
Identificatore univoco	**ID:** UC1
Gli attori interessati dal caso d'uso	**Attori:** Tempo, Fisco
Lo stato del sistema prima che il caso d'uso possa iniziare	**Precondizioni:** 1. Si è concluso un trimestre fiscale
I passi del caso d'uso	**Sequenza degli eventi:** 1. Il caso d'uso inizia quando si conclude un trimestre fiscale. 2. Il sistema calcola l'ammontare dell'IVA dovuta al Fisco. 3. Il sitema trasmette un pagamento elettronico al Fisco.
Lo stato del sistema quando l'esecuzione del caso d'uso è terminata	**Postcondizioni:** 1. Il Fisco riceve l'importo IVA dovuto.

4.5.3 – Sequenza degli eventi

Una **sequenza degli eventi** è un elenco di azioni che definisce il caso d'uso nella sua completezza. Il caso d'uso si considera eseguito solo se l'esecuzione arriva fino alla fine.

Un'**azione** è sempre iniziata da un attore oppure dal sistema (in UML, gli eventi sono sempre legati a chi li crea), quindi:

- **Passo iniziale**: 1. Il caso d′uso inizia quando < attore > < azione > ⋯
- **Passi successivi**: < numero >. Il < attore/sistema > < azione >

Inoltre, ogni caso d'uso:

- Ha una sequenza di transizioni (eventi) normale o di base;
- Può avere varie sequenze alternative;
- Ha varie sequenze di transazioni eccezionali per la gestione di errori o casi particolari.

Quindi, per descrivere correttamente un flusso di eventi si deve:

- Descrivere solo gli eventi relativi al caso d'uso, e non quel che avviene in altri casi d'uso;
- Descrivere come e quando il caso d'uso inizia e finisce;
- Descrivere il flusso base degli eventi;
- Descrivere ogni flusso alternativo.

4.5.4 – Ramificazioni e sequenze alternative

Per capire le **ramificazioni**, si prende in considerazione un **esempio** dell'evento "aggiorna carrello" di un negozio on-line.

Le possibili ramificazioni, dopo aver raggiunto un articolo al carrello sono due:

1. Se il cliente richiede una nuova quantità il sistema aggiorna la quantità di quell'articolo;
2. Se il client seleziona rimuovi articolo il sistema elimina quell'articolo dal carrello.

Le **sequenze alternative** sono invece ramificazioni che possono essere espresse utilizzando il "**Se**". Per **esempio**, il cliente abbandona la pagina del carrello, ovvero una condizione che si può verificare in qualunque momento.

Un **esempio** di ramificazione:

Caso d'uso: AggiornaCarrello
ID: UC2
Attori: Cliente
Precondizioni: 1. Il contenuto del carrello è visibile
Sequenza degli eventi: 1. Il caso d'uso inizia quando il Cliente seleziona un articolo nel carrello. 2. Se il Cliente seleziona "rimuovi articolo" 2.1 Il Sistema elimina l'articolo dal carrello. 3. Se il Cliente digita una nuova quantità 3.1 Il Sistema aggiorna la quantità dell'articolo presente nel carrello
Postcondizioni: 1. Il contenuto del carrello è stato aggiornato
Sequenza alternativa 1: 1. In qualunque momento il Cliente può abbandonare la pagina del carrello
Postcondizioni:

Per **individuare le sequenze alternative**, ad ogni passo della sequenza degli eventi principale si deve cercare:

- Alternative all'azione eseguita in quel passo;
- Errori possibili nella sequenza principale;
- Interruzioni che possono avvenire in qualunque momento della sequenza principale.

Tuttavia, data la complessità delle sequenze alternative, solitamente vengono descritte separatamente. La sintassi è la stessa, ma con la differenza che si sostituisce "casi d'uso" con "sequenza degli eventi alternativa". Inoltre, il primo passo può indicare il punto della sequenza principale da cui provengono le sequenze.

4.5.5 – Scenari

Uno **scenario** rappresenta una particolare interazione tra uno o più attore e il sistema. **Non** contiene ramificazioni o sequenze alternative, è semplicemente la **cronaca di un'iterazione** vera o verosimile.

Per **esempio**, si pensi agli attori come persone fisiche, un utente che acquista un prodotto ed esegue una serie di passi.

Una definizione *alternativa* è: un caso d'uso è un insieme di scenari possibili se un utente prova a raggiungere un dato risultato.

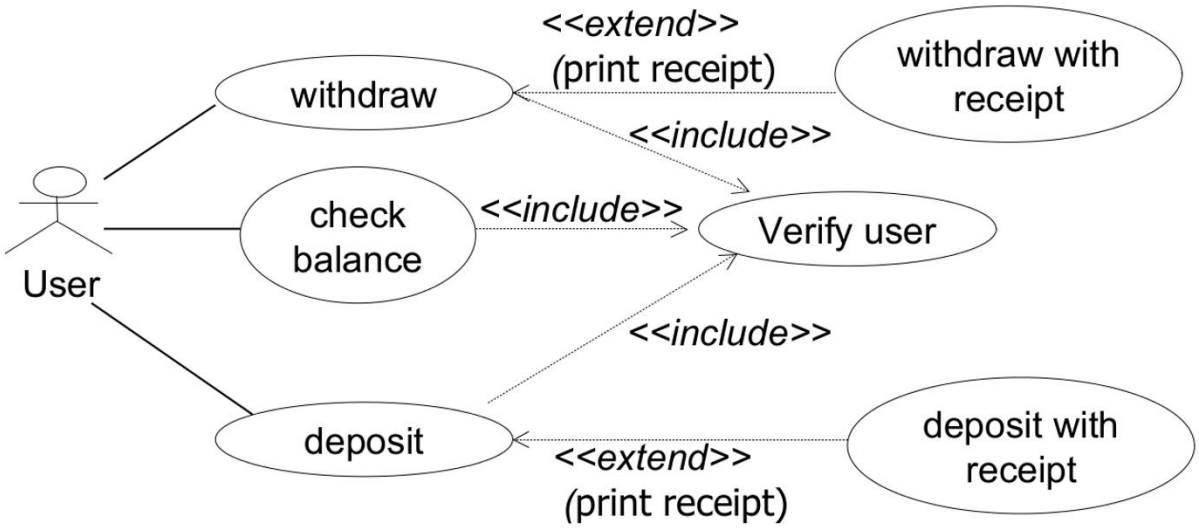
Lo **scenario principale** non è altro che la sequenza principale, mentre lo **scenario secondario** è una sequenza alternativa. Esistono diverse strategie per limitare il numero degli scenari secondi:

- Si documentano solo quelli considerati più importanti;
- Se esistono scenari secondari simili, se ne documenta uno solo, se necessario aggiungendo annotazioni per spiegare come gli altri scenari differiscano dall'esempio.

Per esempio, si consideri il seguente sistema:

- Il sistema usa uno sportello Bancomat;
- L'utente deve poter depositare assegni;
- L'utente deve poter ritirare contante;
- L'utente deve poter chiedere il saldo;
- L'utente deve poter ottenere una ricevuta se lo richiede. La ricevuta riporta il tipo di transazione, la data, il numero di conto, la somma ed il nuovo saldo;
- Dopo ciascuna transazione viene visualizzato il nuovo saldo.

Il **diagramma dei casi d'uso**:



Di seguito viene scritta la **specifica** dei **principali casi d'uso**:

- [**Caso d'uso withdraw**]:
  - **Precondizione**. L'utente ha selezionato l'opzione di ritirare
  - **Flusso principale**.
    - Include utenti verificati
    - Richiedere all'utente l'importo da prelevare
    - Controllo se la somma inserita è disponibile nel conto dell'utente
    - Controllo se la quantità di denaro è disponibile all'interno del bancomat
    - Rimozione della somma dall'account
    - Dare la somma di denaro all'utente
    - (Opzionale) Stampa della ricevuta
    - Stampa del saldo corrente nel conto corrente dell'utente
  - **Flusso eccezionale**.
    - Se la somma di denaro non è disponibile, richiedere all'utente una somma di denaro minore

- [**Caso d'uso deposit**]:
  - **Precondizione**. L'utente ha selezionato l'opzione di deposito
  - **Flusso principale**.
    - Include utenti verificati
    - Richiedere all'utente l'importo da depositare
    - Aprire lo slot per inserire il denaro
    - Effettuare un controllo sulla quantità di denaro inserita
    - (Opzionale) Stampa della ricevuta
    - Stampa del saldo aggiornato nel conto corrente dell'utente

- [**Caso d'uso check balance**]:
  - **Precondizione**. L'utente ha selezionato l'opzione di controllare il saldo
  - **Flusso principale**.
    - Include utenti verificati
    - Stampa del saldo corrente

- [**Caso d'uso verify user**]:
  - **Precondizione**. Nessuna
  - **Flusso principale**.
    - L'utente inserisce l'ID della carta
    - L'utente inserisce il PIN della carta
    - Il sistema controlla la validità dell'ID e del PIN della carta
  - **Flusso eccezionale**.
    - Se la combinazione non è valida, l'utente viene rifiutato

Esistono una serie di **errori tipici sui diagrammi**:

- Diagrammi di flusso invece di casi d'uso. Un caso d'uso è una sequenza di azioni, non una singola azione
- Nome del caso d'uso che appare più volte nel diagramma
- Le frecce tra i casi d'uso sono tratteggiate o etichettate *extend* o *include*
- Errori con *extend*: la freccia va dal caso che descrive l'evento alternativo al caso standard
- Errori con *include*: la freccia va dal caso chiamante al caso che descrive le azioni da includere

Non solo, esistono una serie di **errori tipici** anche **sugli scenari**:

- Assenza di precondizioni
- Mancata connessione alla rappresentazione grafica
- Nomi diversi per le stesse entità nelle rappresentazioni grafica e testuale
- Flusso eccezionale: mancanza di indicazioni nel flusso principale del punto in cui va controllata la condizione eccezionale