UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Ingegneria del Software – Requirement Engineering

Prof. Sergio Di Martino

Obiettivi della lezione

- Comprendere il concetto di Ciclo di Vita del Software
- L'Ingegneria dei Requisiti
- Comprendere le tecniche per l’analisi / specifica dei requisiti
  - Use Cases
  - Diagrammi di Cockburn

Il Ciclo di Vita del Software

Processo

- “*Un processo è un particolare metodo per fare qualcosa costituito da una sequenza di passi che coinvolgono attività, vincoli e risorse*” (Pfleeger)
- “*Processo: una particolare metodologia operativa che nella tecnica definisce le singole operazioni fondamentali per ottenere un prodotto industriale*” (Zingarelli)
- “*Processo software: un metodo per sviluppare del software*” (Sommerville)

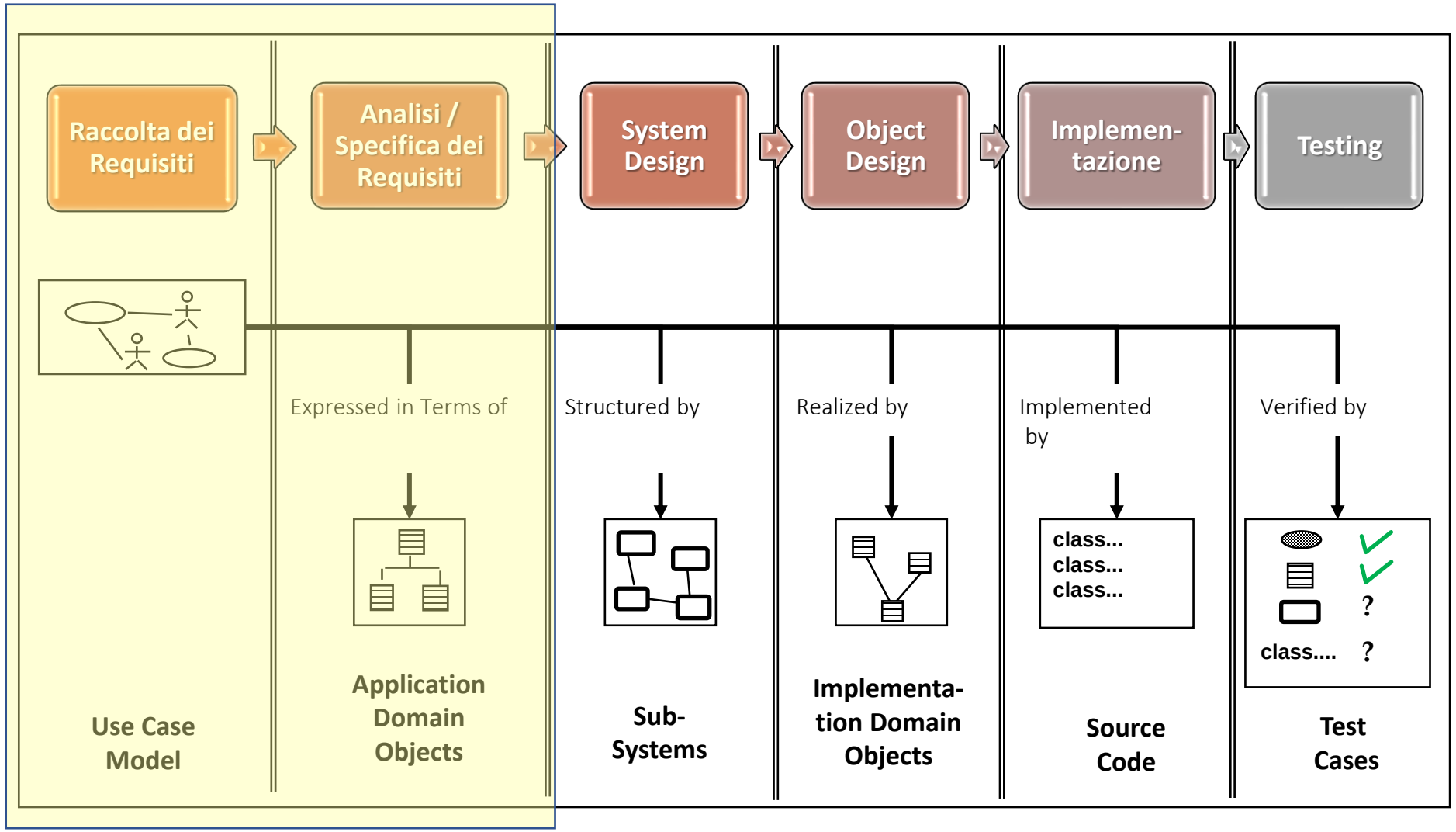
Processo/Ciclo di vita del software

- Insieme organizzato di attività che sovrintendono alla costruzione del software da parte del team di sviluppo utilizzando metodi, tecniche, metodologie e strumenti.
- È suddiviso in fasi, che vanno dalla nascita fino alla dismissione (o morte) del software.
  - Per un parallelo con la biologia, è noto come ***ciclo di vita del software***
- Molti autori usano i termini ***processo*** *[di sviluppo del] software* e ***ciclo di vita*** *del software* come sinonimi.

Processo software: Standard IEEE 610.12-1990

- *«Sofware development process: The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use».*
  - *Note: These activities may overlap or be performed iteratively.*

Ciclo di Vita del Software

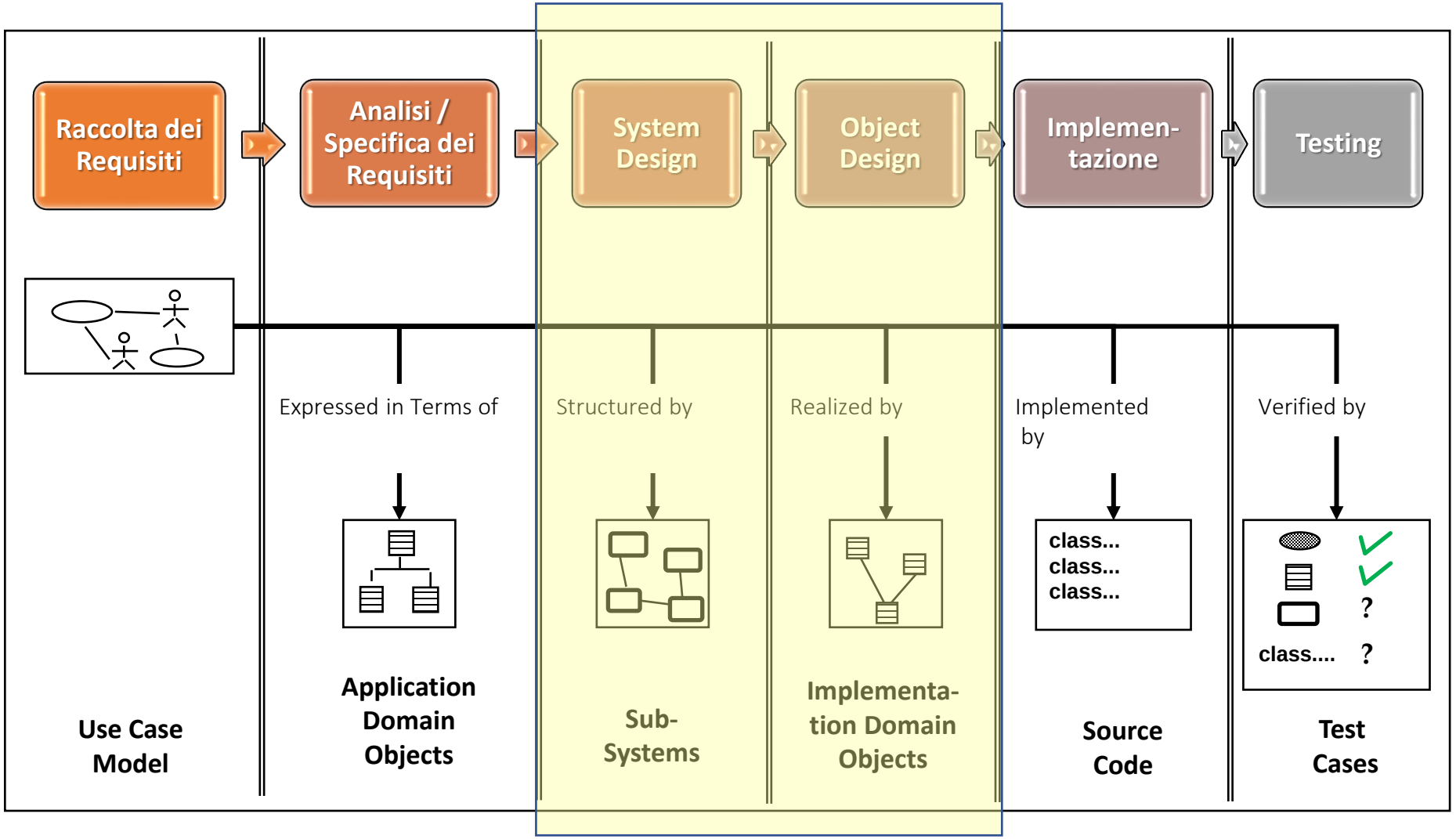


Raccolta/Analisi/Specifica dei requisiti

- Analisi completa dei bisogni dell'utente e dominio del problema
- Coinvolgimento di committente e ingegneri del SW
- Obiettivo
  - Descrivere le caratteristiche di qualità che l'applicazione deve soddisfare

- Output:
  - documento di specifica dei requisiti

Ciclo di Vita del Software

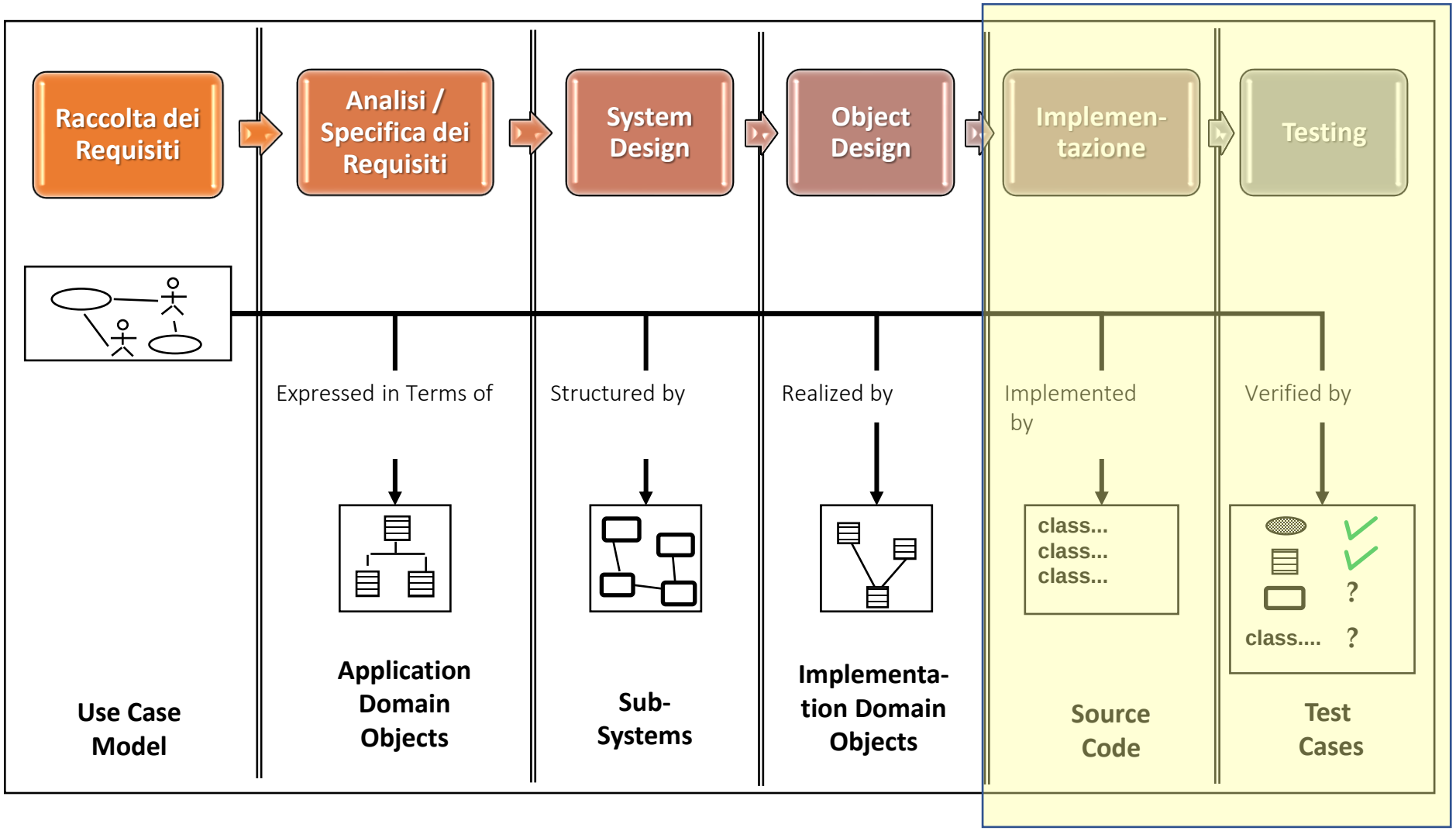


Progettazione (o System/Object Design)

- Definizione di una struttura opportuna per il SW
- Scomposizione del sistema in componenti e moduli
  - allocazione delle funzionalità ai vari moduli
  - definizione delle relazioni fra i moduli
- Distinzione fra:
  - System Design: struttura modulare complessiva (componenti)
  - Object Design: dettagli interni a ciascuna componente
- Obiettivo

- Output: documento di specifica di progetto
  - possibile l'uso di linguaggi per la progettazione (UML)

Ciclo di Vita del Software



Fasi basse del processo

- **Implementazione**: ogni modulo viene codificato nel linguaggio scelto e testato in isolamento
- **Testing**
  - Composizione dei moduli nel sistema globale
  - Verifica del corretto funzionamento del sistema
  - Validazione che il sistema faccia ciò che vuole il cliente
- **Installazione**: distribuzione e gestione del software presso l'utenza
- **Manutenzione**: evoluzione del SW. Segue le esigenze dell'utenza. Comporta ulteriore sviluppo per cui racchiude in sé nuove iterazioni di tutte le precedenti fasi

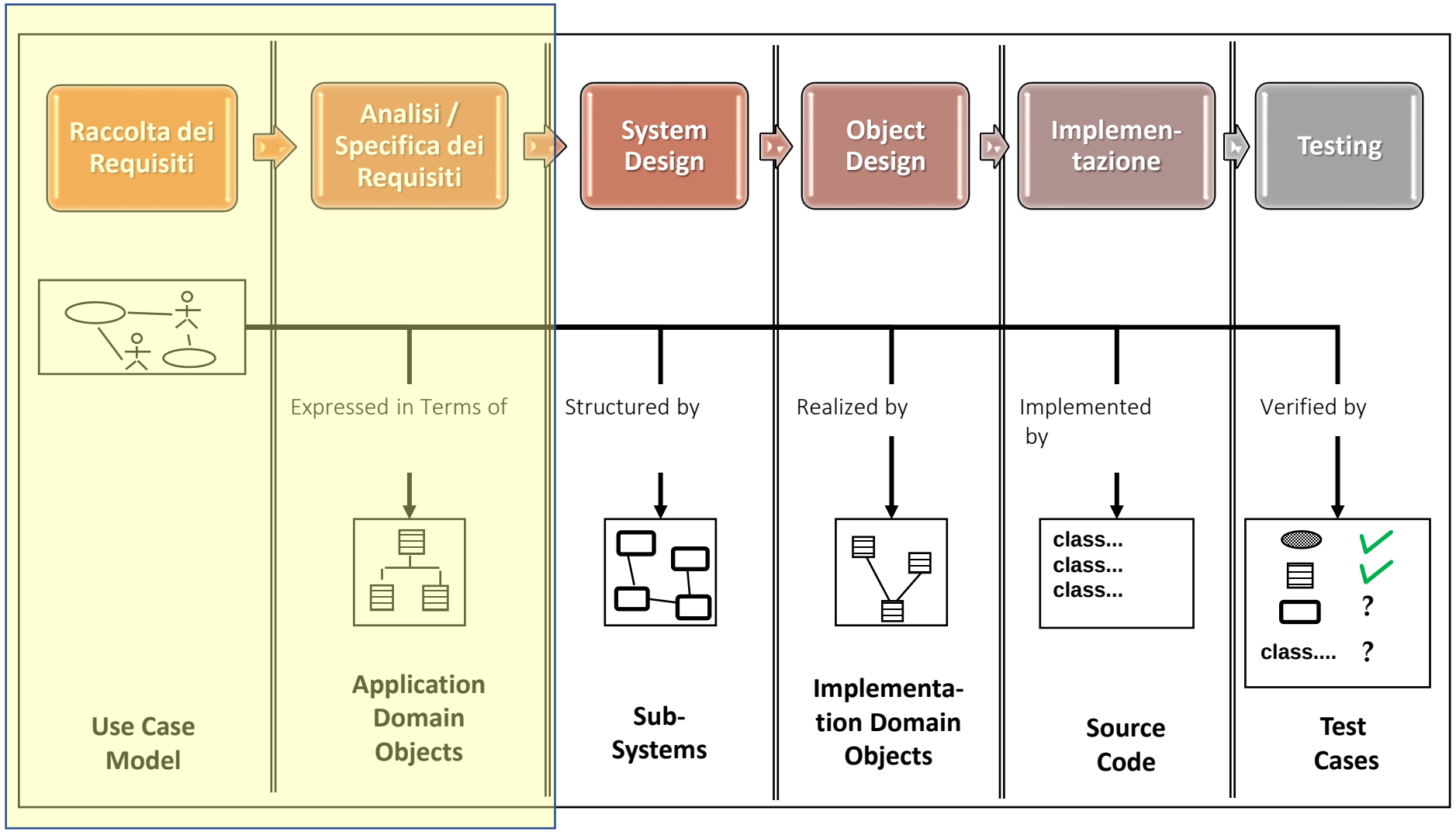
Problemi nel processo di sviluppo del software

- E’ qualcosa di altamente intellettuale e creativo, basato su giudizi delle persone
  - Non è possibile (almeno con i sistemi attuali) automatizzarlo
- I requisiti sono complessi e ambigui
  - Il cliente non sa bene, dall’inizio, cosa deve fare il software
- I requisiti sono variabili
  - Cambiamenti tecnologici, organizzativi, etc…
- Modifiche frequenti sono difficili da gestire
  - Difficile stimare i costi ed identificare cosa consegnare al cliente

La Requirement Engineering

Capitolo 4 - Sommerville

Ciclo di Vita del Software



Requisiti

- Def. (IEEE Glossary) requirement.
  1. *A condition or capability needed by a user to solve a problem or achieve an objective.*
  2. *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.*
  3. *A documented representation of a condition or capability as in (1)(2).*
- Ingegneria dei requisiti: Definire i requisiti del sistema di interesse.

The Demons of Ambiguity (Harry Robinson)

- Why does the sign use the plural “children” instead of the singular “child”?
- Who qualifies as a “child”?
- What does it mean to be “present”?
- Does this rule apply only when school is in session? What about weekends? Holidays? Nighttime?

Requisiti software

- Requisiti di un sistema software:

  Cosa deve fare il sistema + Vincoli operativi

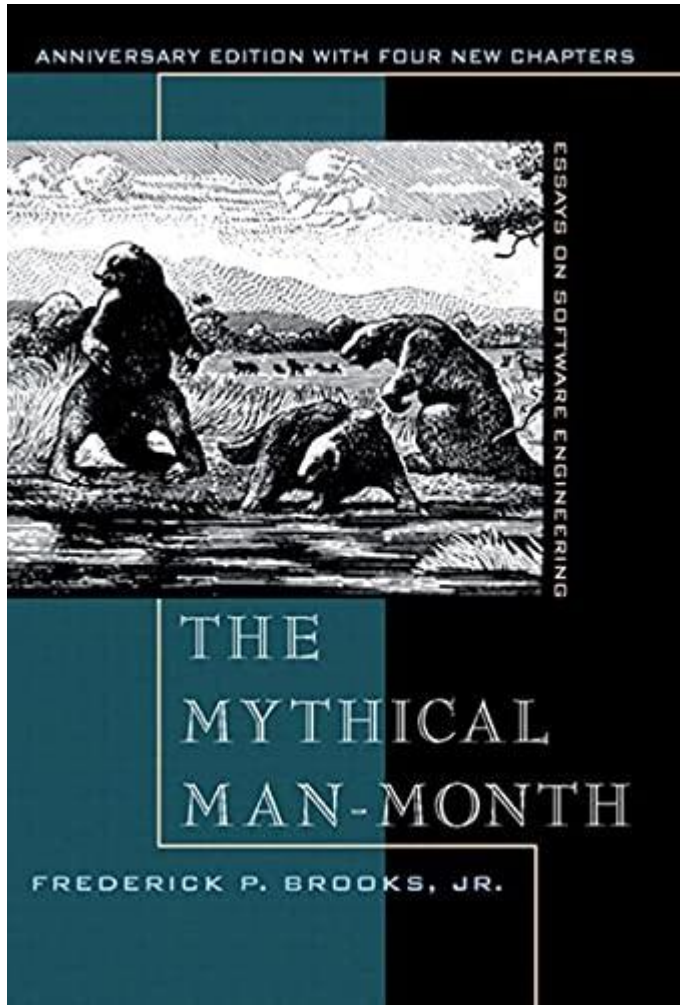
- Esempi:
  - Il sistema dovrà archiviare i dati di una biblioteca, in particolare dati relativi ai libri, ai giornali, le riviste, video, audio e DVD.
  - Il sistema dovrà permettere agli utenti di fare delle ricerche per titolo, autore, o ISBN.
  - L'interfaccia al sistema dovrà essere realizzata come un'app per iOS e Android.
  - Il sistema dovrà gestire almeno 20 transazioni per secondo
  - Il sistema dovrà riconoscere l'utente attraverso un sensore biometrico

L’ambiguità dei requisiti

- Il termine *requisito* non è usato nell'industria del software in modo coerente.
- In alcuni casi, un requisito è semplicemente una formulazione astratta di un servizio che il sistema dovrebbe fornire, oppure un vincolo del sistema.
- Altre volte, può essere una definizione formale e dettagliata di una finzione del sistema:
  - *Se una società vuole dare in appalto un grande progetto di sviluppo software, deve definire sue esigenze in modo abbastanza astratto da non predefinire alcuna soluzione.*
    I requisiti devono essere scritti in modo che diversi appaltatori possano fare le offerte proponendo vari metodi per soddisfare le necessità del cliente Dopo che l'appalto è stato assegnato, l'appaltatore deve scrivere per il cliente una definizione del sistema molto dettagliata, in modo che il cliente possa capire e verificare che cosa farà il software. Entrambi questi documenti possono essere chiamati documenti dei requisiti.

Fred Brook's

- *"The most difficult part of building a software system is to decide, precisely, what must be built.*
  No other part of the work can undermine so badly the resulting software if not done correctly. No other part is so difficult to fix later."

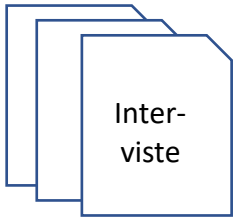


Requirement Engineering

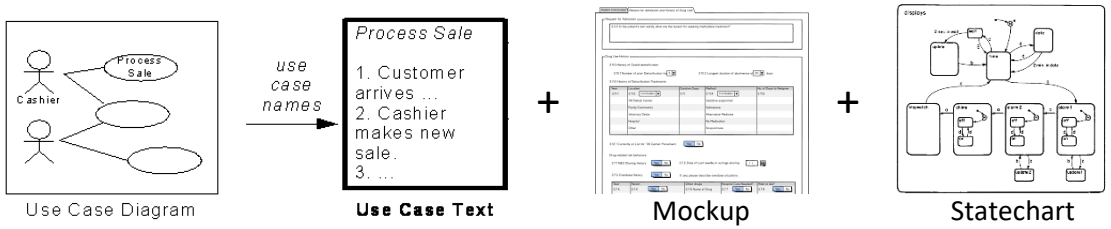
- L'*Ingegneria dei Requisiti* comprende tutte le attività che si occupano di requisiti:
  - Raccolta dei requisiti
  - Analisi dei requisiti
  - Specifica / Documentazione dei requisiti
  - Verifica e Validazione dei requisiti

Requirement Engineering

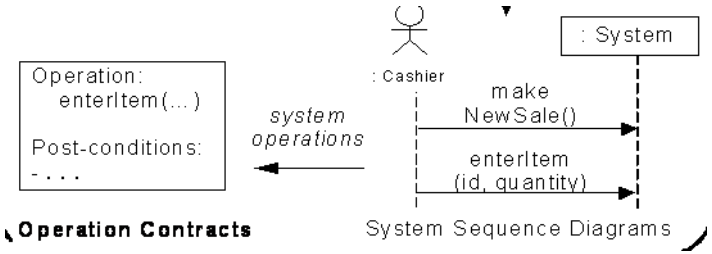
1. **Requirement Elicitation**



2. **Requirement Analysis**



3. **Requirement Specification**



4. **Requirement Validation**

Requirement Engineering

- **L'ingegneria dei requisiti** comprende (almeno) 4 attività:

1. Raccolta (scoperta) dei Requisiti (**Requirement Elicitation**)
   - Vengono intervistati gli stakeholders per avere un insieme di requisiti del nuovo sistema
2. Analisi dei Requisiti (**Requirement Analysis**)
   - Le informazioni raccolte vengono organizzate in un documento, con supporto di diagrammi di alto livello (UML Use Cases + Cockburn + Mock-up)
3. Specifica dei Requisiti (**Requirement Specification**)
   - I requisiti riorganizzati nella fase di analisi vengono formalizzati per mezzo di opportuni linguaggi di modellazione (UML Class – Sequence – Activity - Statechart Diagrams)
4. Convalida dei requisiti (**Requirement Validation**)
   - Si controlla che effettivamente i requisiti definiscano il sistema che il cliente voleva

Tipologie di Requisiti

Tipologie di Requisiti

- I Requisiti SW possono essere classificati secondo due diversi punti di vista:
  - Livello di Dettaglio
    - Requisiti Utente
    - Requisiti di Sistema
  - Tipo di Requisito rappresentato
    - Requisiti Funzionali
    - Requisiti Non Funzionali
    - Requisiti di Dominio

Livello di dettaglio

- Possono espressi a vari livelli di astrazione e formalismo, dando luogo a 2 tipologie di requisiti:
- **Requisiti Utente**
  - Descrivono l'insieme di funzionalità richieste al sistema (comportamento osservabile dall'esterno) e i vincoli operativi del sistema.
  - Scritti in linguaggio naturale.
  - Il punto di vista è quello dell'Utente, che li sottopone a un possibile Sviluppatore per ottenere una offerta (requisiti aperti a soluzioni alternative)
- **Requisiti di Sistema**
  - Formulazione dettagliata, strutturata, e testabile di servizi e vincoli.
  - Scritti in linguaggio naturale, notazioni semi-formali, linguaggi formali
  - Il punto di vista è quello dello Sviluppatore, che li puo' usare anche per il contratto con il Cliente (requisiti piu' restrittivi)

User and System Requirements

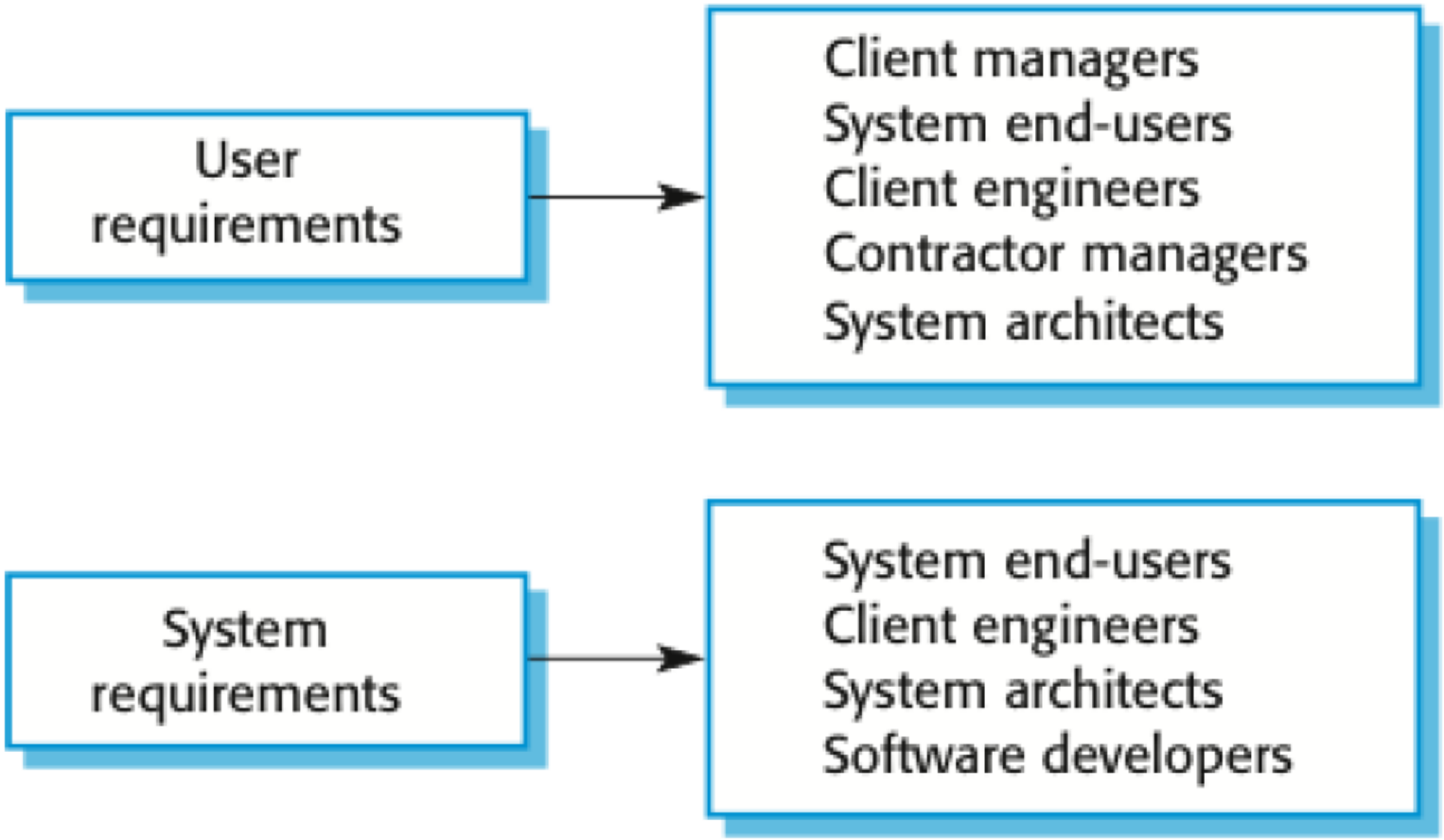
USER REQUIREMENT (UN Requisito Utente)

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

SYSTEM REQUIREMENT (diventa PIU' requisiti di Sistema)

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 automatically 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, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Destinatari dei diversi tipi di requisiti



Tipi di requisiti

- I **Requisiti Funzionali** descrivono le funzionalità offerte dal sistema (normalmente attivati da azioni degli utenti)
  - “Quando l’utente richiede la visualizzazione dell’estratto conto… allora il sistema deve …”
- I **Requisiti Non-Funzionali** descrivono vincoli sui servizi offerti dal sistema, e sullo stesso processo di sviluppo
  - “La visualizzazione dell’estratto conto deve avvenire entro 4 secondi dalla sua richiesta”
- I **Requisiti di Dominio** (funzionali e non-funzionali) riflettono vincoli generali del dominio applicativo, non sempre esplicitati
  - “L’accesso alla cassa continua da parte dell’addetto bancario al rifornimento deve avvenire secondo le consuete procedure di sicurezza a doppia-chiave”

Requisiti Funzionali

- Descrivono le interazioni tra il sistema e il suo ambiente indipendentemente dalla sua implementazione (l'ambiente include l'utente e ogni altro sistema esterno)
- ESEMPI
  - A user shall be able to search the appointments lists for all clinics.
  - The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
  - Each staff member using the system shall be uniquely identified by his or her 8-digit employee number…

Qualità dei Requisiti Funzionali

- I requisiti funzionali devono essere:
  - **Completi**: devono indicare tutti i servizi richiesti dagli utenti
  - **Coerenti**: i requisiti non devono avere definizioni contraddittorie

- Per sistemi grossi è difficile ottenere requisiti completi e coerenti
  - I vari stakeholders hanno esigenze diverse, spesso in contrasto

Requisiti non funzionali

- Descrivono gli aspetti del sistema che non sono direttamente legati al comportamento (funzionalità) del sistema.
- ESEMPI (MHC-PMS).
  - The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.
  - Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.
  - The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.
- Includono una grande varietà di richieste che si riferiscono a diversi aspetti del sistema, dall'usabilità alle performance

Goals and requirements

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

Usability requirements

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
- Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes 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

Requisiti di dominio

- Si riferiscono a caratteristiche del sistema che derivano da caratteristiche generali del dominio applicativo

- Problemi:
  - A volte sono Requisiti impliciti: il Cliente li dà per scontati e non li esprime esplicitamente: ma lo Sviluppatore non lo sa…
  - A volte sono Requisiti espliciti ma oscuri: Il Cliente descrive requisiti utilizzando termini e concetti che lo Sviluppatore ignora…(➔ Glossario)

Requisiti di dominio – Esempi

- Gestione biblioteca
  - ‘There shall be a standard user interface to all databases which shall be based on the Z39.50 standard’
    - Lo Standard è noto al personale della biblioteca, non agli sviluppatori
- POS NextGEN
  - Il codice identificativo dell’articolo, letto mediante codice a barre, può essere basato sui seguenti schemi di codifica:
    - UPC
    - EAN
    - JAN
    - SKU

Requirement Engineering

Fase 1: Requirement Elicitation

Requirement Elicitation

- La raccolta dei requisiti è considerata l'attività più difficile, perché richiede la collaborazione tra più gruppi di partecipanti con differenti background.
  - **Stakeholders**: "insieme dei soggetti che hanno un interesse nei confronti di un'organizzazione e che con il loro comportamento possono influenzarne l'attività".

    →
  - tutte le persone in qualche modo interessate alla messa in opera del sistema
  - Il cliente e gli utenti finali sono esperti nel loro dominio, con la propria terminologia, e hanno una idea generale (spesso vaga) di cosa il sistema debba fare, e poca (o nulla) esperienza nello sviluppo del software
  - Gli sviluppatori hanno esperienza nel produrre sistemi software, ma hanno una conoscenza limitata del dominio di applicazione (ambiente degli utenti finali)

Requirement Elicitation

- La raccolta dei requisiti si focalizza sul punto di vista dell'utente e definisce i "boundary" (confini) del Sistema da Sviluppare (System under Development, o SuD).
- Vengono specificati:
  - Funzionalità del sistema
  - Interazione tra utente e sistema
  - Errori che il sistema deve individuare e gestire
  - Vincoli e condizioni di utilizzo del sistema
- Non fanno parte dell'attività di raccolta dei requisiti:
  - la selezione delle tecnologie da usare per lo sviluppo
  - il progetto del sistema e le metodologie da usare
  - …. in generale tutto quello che non è direttamente visibile all'utente

Come raccogliere i requisiti

- L’elicitation dei requisiti avviene attraverso un processo di raccolta delle informazioni, coinvolgendo tutti gli stakeholders.
- Può essere svolto in tre modi, eventualmente complementari:
  1. Interviste
  2. Scenari
  3. Mock-Up

Interviste

- Possono essere:
  - Chiuse
    - Lo stakeholder risponde ad un insieme predefinito di domande
  - Aperte (Brainstorming)
    - Dialogo a ruota libera
- Le interviste non sempre sono una tecnica efficace per raccogliere requisiti, perché non sempre gli stakeholder sono in grado/vogliono descrivere il loro dominio

Scenari

- Uno scenario è "*una descrizione narrativa di cosa le persone fanno e sperimentano mentre provano ad usare i sistemi di elaborazione e le applicazioni*" [M. Carrol, Scenario-based Design, Wiley, 1995]

- Uno scenario è una descrizione concreta, focalizzata, e informale di una singola caratteristica di un sistema utilizzata da un singolo attore.

- Gli scenari possono essere utilizzati in diversi modi durante il ciclo di vita del software

Esempio di Scenario

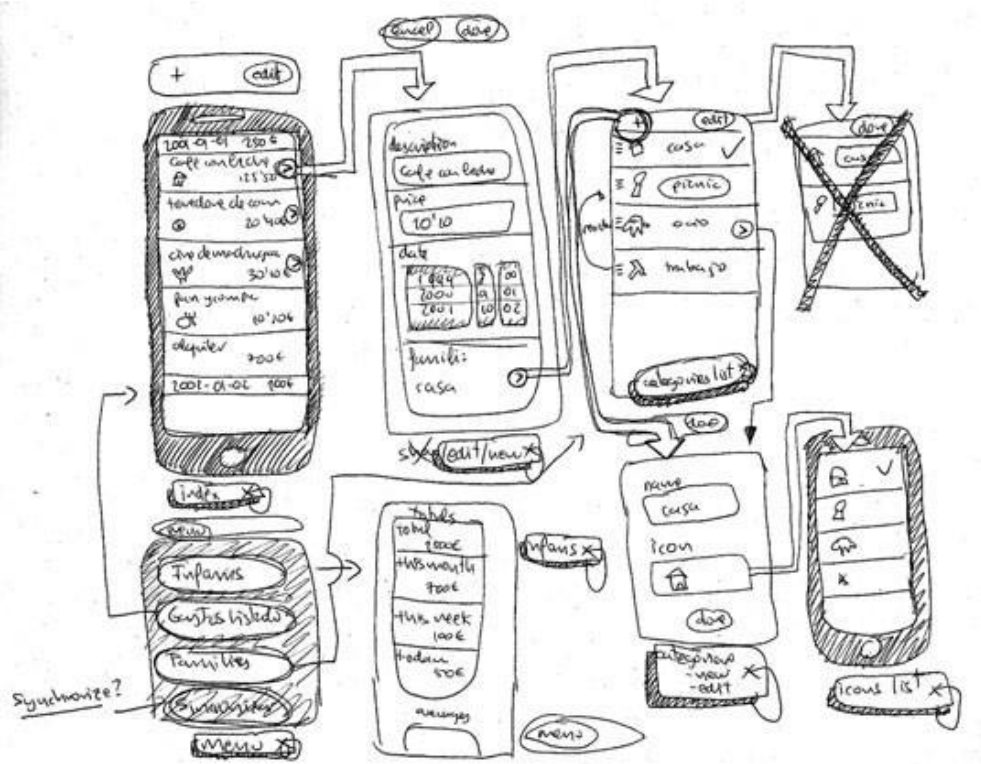
- **Gestisci Restituzione Articoli**
- *Scenario principale di successo:*
  - Un cliente arriva alla cassa con alcuni articoli da restituire. Il cassiere usa il sistema per registrare ciascun articolo restituito …
- *Scenari alternativi:*
  - Se il cliente aveva pagato con la carta di credito, e la transazione di rimborso sul relativo conto di credito è stata respinta, allora il cliente viene informato e viene rimborsato in contanti.
  - Se ...

Euristiche per trovare gli Scenari

- Chiedersi o chiedere al cliente le seguenti domande:
  - Quali sono i compiti primari che l'utente vuole che esegua il sistema?
  - Quali informazioni saranno create, memorizzate, modificate, cancellate, o aggiunte dall'utente nel sistema?
  - Di quali cambiamenti esterni l'utente deve informare il sistema?
  - Di quali cambiamenti o eventi del sistema deve essere informato l'utente?

Prototipi / Mock-up

- La prototipazione può aiutare molto nella fase di elicitation
- Utilizzare i Mock-up
  - Rappresentazione statica dell'interfaccia, intesa come prototipo per avere feedback dall'utente
  - Realizzata con «wire-frame»
  - Esempio con Balsamiq



L’Elicitation è la parte più difficile

- Problemi sugli obiettivi
  - I boundaries del sistema non sono chiari
  - Gli Stakeholders forniscono moltissime informazioni irrilevanti
- Problemi di comprensione
  - Gli StakeHolders non sanno mai esattamente cosa vogliono
  - Gli StakeHolders non hanno una visione chiara di possibilità e limiti dei sistemi informativi
  - Gli StakeHolders hanno difficoltà a comunicare tutte le loro esigenze
  - Il linguaggio naturale è intrinsecamente ambiguo
- Problemi di volatilità
  - I requisiti possono cambiare di continuo (stima reale: almeno il 25% dei requisiti cambierà durante il progetto)

Difficoltà nella Requirement Elicitation

Verificabilità di requisiti non funzionali

- I Requisiti non funzionali dovrebbero sempre essere Verificabili
- Goal (non verificabile)
  - ‘Il sistema deve essere facile da usare per controllori esperti, e deve essere tale da minimizzare gli errori degli utenti’
- Requisito non-funzionale (verificabile)
  - ‘controllori esperti devono poter imparare a usare tutte le funzioni del sistema in max. 2 ore di apprendimento.
  - Dopo l’apprendimento, il controllore deve essere in grado di operare senza commettere piu’ di 2 errori al giorno’.

Conflitti fra requisiti non-funzionali

- Esempio: Sistema di rilevamento pattume spaziale sullo Shuttle
  - Req.1 - System should fit into 4Mbytes of memory
  - Req.2 - System should be written in ADA

- Puo’ risultare impossibile compilare un programma ADA con le funzionalià richieste, e che occupi solo 4Mbytes: uno dei requisiti va escluso

Validazione dei requisiti

- E’ un passo critico nel processo di sviluppo
- I requisiti sono continuamente validati da cliente e utenti
- La validazione dei requisiti richiede di controllare:
  - **Correttezza**: una specifica è corretta se rappresenta accuratamente il sistema che il cliente richiede e che gli sviluppatori intendono sviluppare;
  - **Completezza**: una specifica è completa se tutti i possibili scenari per il sistema sono descritti, incluso i comportamenti eccezionali
  - **Coerenza**: se i requisiti non si contraddicono tra di loro
  - **Chiarezza**: una specifica è chiara se non è possibile interpretare la specifica in due modi diversi

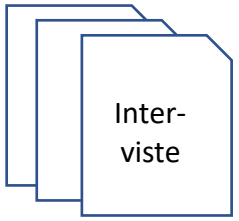
Validazione dei requisiti (2)

- Realismo: La specifica dei requisiti è realistica se può essere implementata tenendo conto dei vincoli
- Verificabilità: La specifica dei requisiti è verificabile se, una volta che il sistema è stato costruito, test ripetuti possono essere delineati per dimostrare che il sistema soddisfa i requisiti
  - Es: Il prodotto dovrebbe avere una buona interfaccia – Buono non definito
- Tracciabilità: Ogni funzione del sistema può essere individuata e ricondotta al corrispondente requisito funzionale.
  Include anche l'abilità di tracciare le dipendenze tra i requisiti, le funzioni del sistema, gli artefatti, incluso componenti, classi, metodi e attributi di oggetti
  - È cruciale per lo sviluppo di test e per valutare i cambiamenti

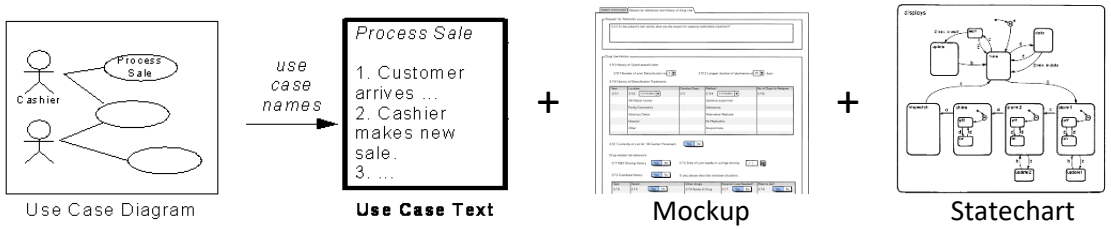
Requirement Analysis e UML

Requirement Engineering

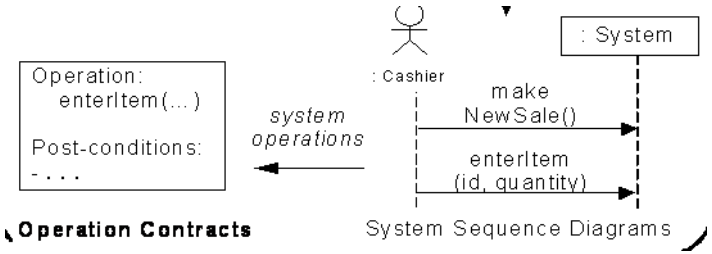
1. **Requirement Elicitation**



2. **Requirement Analysis**



3. **Requirement Specification**



4. **Requirement Validation**

Modelli del sistema

- Nella fase di Requirement Analysis, tutto il testo ed i Mock-Up raccolti nell'Elicitation vengono rielaborati e strutturati, portando alla generazione di un primo documento di **modellazione astratta** del sistema
- L'obiettivo di questi modelli è di descrivere in maniera non ambigua il problema da trattare, in termini di:
  - Categorie di utenti
  - Insieme di funzionalità
  - Insieme di informazioni da gestire

Fasi della Requirement Analysis

Analizzare i documenti ottenuti dalla Elicitation, al fine di:

1. Identificare gli Attori
   - Individuare le differenti classi di utenti che il sistema dovrà supportare
2. Identificare gli Use Case
   - Derivare dagli scenari un insieme di use case che rappresentano in modo completo il sistema software da sviluppare.
   - Uno use case è un’astrazione che descrive una classe di scenari.

Fasi della Requirement Analysis (2)

3. Dettagliare gli Use Case
   - Assicurarsi che le funzionalità del sistema siano completamente specificate, dettagliando ogni use case e descrivendo il comportamento del sistema in situazioni di errore, e condizione limite
4. Identificare i requisiti non funzionali
   - Tutti gli stakeholders si accordano sugli aspetti visibili all'utente finale ma che non sono direttamente relati alle funzionalità del sistema:
     - vincoli sulle prestazioni
     - utilizzo delle risorse
     - sicurezza
     - e qualità.

Introduzione a UML

Unified Modeling Language (UML)

- Un insieme di linguaggi (e notazioni) universali, per rappresentare qualunque tipo di sistema (software, hardware, organizzativo, …)
- Standard OMG (Object Management Group), dal nov.1997
- Creatori:
  - Grady Booch, Ivar Jacobson e Jim Rumbaugh
  - noti come “los gringos” dell’Ingegneria del SW

Definizione ufficiale

- *“Un linguaggio per specificare, visualizzare, e realizzare i prodotti di sistemi software, e anche per il business modeling. L’ UML rappresenta una collezione di "best engineering practices" che si sono dimostrate utili nella progettazione di sistemi complessi e di grandi dimensioni.”*

Cos’è UML

- E’ un linguaggio di modellazione e specifica di sistemi, basato sul paradigma object-oriented
- Serve a specificare le caratteristiche di un nuovo sistema, oppure a documentarne uno già esistente
- E’ uno strumento di comunicazione tra i diversi ruoli coinvolti nello sviluppo e nell’evoluzione dei sistemi
  - UML svolge un'importantissima funzione di lingua franca nella comunità della progettazione e programmazione a oggetti.
  - Gran parte della letteratura di settore usa UML per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico

Cosa NON è UML

- E’ un linguaggio, non un metodo
  - Notazione, sintassi e semantica sono standard
- Ma UML non è legato ad uno specifico processo, e non fornisce indicazioni sul proprio utilizzo
- Quindi può essere (ed è) utilizzato da persone e gruppi che seguono approcci diversi (è “indipendente dai metodi”)
- UML è indipendente dal linguaggio di programmazione utilizzato

I tre aspetti della modellazione

- UML consente di descrivere un sistema secondo tre aspetti principali, in relazione fra loro:
  - **Il modello funzionale** rappresenta il sistema dal punto di vista dell'utente. Ne descrive il suo comportamento così come esso è percepito all'esterno, prescindendo dal suo funzionamento interno.Sserve nell'analisi dei requisiti.
    - Use Case Diagram.
  - **Il modello a oggetti** rappresenta la struttura e sottostruttura del sistema utilizzando i concetti O-O. Questo tipo di modellazione può essere utilizzata sia nella fase di analisi che di design, a diversi livelli di dettaglio.
    - Class diagram, object diagram, e deployment diagram.
  - **Il modello dinamico** rappresenta il comportamento degli oggetti del sistema, cioè le dinamiche delle loro interazioni. È strettamente legato al modello a oggetti.
    - Sequence diagram, activity diagram e statechart diagram.

UML - considerazioni

- UML permette di descrivere un sistema software a diversi livelli di astrazione, dal piano più svincolato dalle caratteristiche tecnologiche fino all'allocazione dei componenti software nei diversi processori in un'architettura distribuita.
- UML è sufficientemente complesso per rispondere a tutte le necessità di modellazione, ma è opportuno "ritagliarlo" in base alle specifiche esigenze dei progettisti e dei progetti, utilizzando solo ciò che serve nello specifico contesto
- "For 80% of all software, only 20% of UML"
- "keep the process as simple as possible!"

UML Core Conventions

- I Rettangoli sono classi o istanze
- Gli ovali sono funzioni o Use Case
- Le istanze sono denotate dal nome sottolineato
  - myWatch: SimpleWatch
- I tipi sono denotati da nomi NON sottolineati
  - SimpleWatch
  - Firefighter
- I diagrammi sono dei grafi
  - I nodi sono le entità
  - Gli archi sono relazioni tra le entità

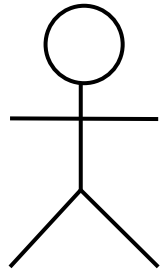
Gli Use Case Diagrams

Use Case Diagrams

- Modello UML usato per rappresentare graficamente l'insieme dei requisiti funzionali di un sistema
- Espresso in termini di:
  - Attori – Categorie di utenti (umani e non) del sistema
  - Casi d'Uso – Funzionalità offerte dal sistema
- Inteso come mezzo di comunicazione Fornitore – Cliente per definire le funzionalità del sistema → Va tenuto quanto più semplice possibile
- NON modella "come" funziona il sistema

Gli Attori

- Un attore modella un'entità esterna, dotata di comportamento, che interagisce con il sistema:
  - Classe di Utenti
  - Sistema esterno
  - Ambiente fisico
- Un attore ha un nome univoco ed una descrizione opzionale.
- Esempi:
  - Cassiere: Una persona
  - Servizio Autorizzazione al Pagamento: Sistema informativo pre-esistente che interagisce col Sistema da Sviluppare



Cassiere

Euristiche per Identificare gli Attori

- Per identificare gli attori, esistono alcune linee guida
  - Tracciare i Boundaries del SuD.
  - Quali gruppi di utenti sono supportati dal sistema per svolgere il loro lavoro?
  - Quali gruppi di utenti eseguono le principali funzioni del sistema?
  - Quali gruppi di utenti eseguono le funzioni secondarie, come il mantenimento e l'amministrazione?
  - Con quale sistema hardware o software esterni il sistema interagisce? Ogni sw già esistente con cui il SuD interagirà, è un attore
    - Es: In POS NextGEN, il sw di una banca, pre-esistente, che ci autorizza al prelievo dalla carta di credito è un attore.
  - Un attore non corrisponde ad una persona fisica, ma ad un ruolo.
    - Una persona può svolgere diversi ruoli nello stesso sistema. Es: sono l'*Administrator* di un sito, ma se non mi loggo sono un *Utente non Registrato*

Gli Use Case

- Un caso d'uso è una funzionalità offerta dal sistema, che porta ad un risultato misurabile per un attore.
- I casi d'uso modellano i Requisiti Funzionali del SuD.
- Un caso d'uso astrae tutti gli scenari per una data funzionalità → Uno scenario è un'istanza di un caso d'uso.
  - Uno Use Case rappresenta, attraverso un flusso di eventi, una classe di funzionalità offerte dal sistema
  - Uno Use Case specifica le interazioni Attori - Sistema per una data funzionalità (insieme di scenari)

Use Case Text

2 - Identificare gli Use Case

- Un caso d'uso è **SEMPRE** descritto dal punto di vista degli attori (sistema = black-box)
- I Casi d'uso catturano il comportamento **esterno** del sistema da sviluppare, senza dover specificare come tale comportamento viene realizzato
- Un caso d'uso è **SEMPRE** iniziato da un attore, quindi può interagire con altri attori
- Un caso d'uso deve portare ad un **vantaggio** tangibile per l'attore che lo compie (so that?)
- Opzionalmente si aggiungono anche Mock-up

Euristiche per l'individuazione dei Casi d'Uso

- Per ogni attore, definirne gli obiettivi
  - Creare una tabella Attore-Obiettivi

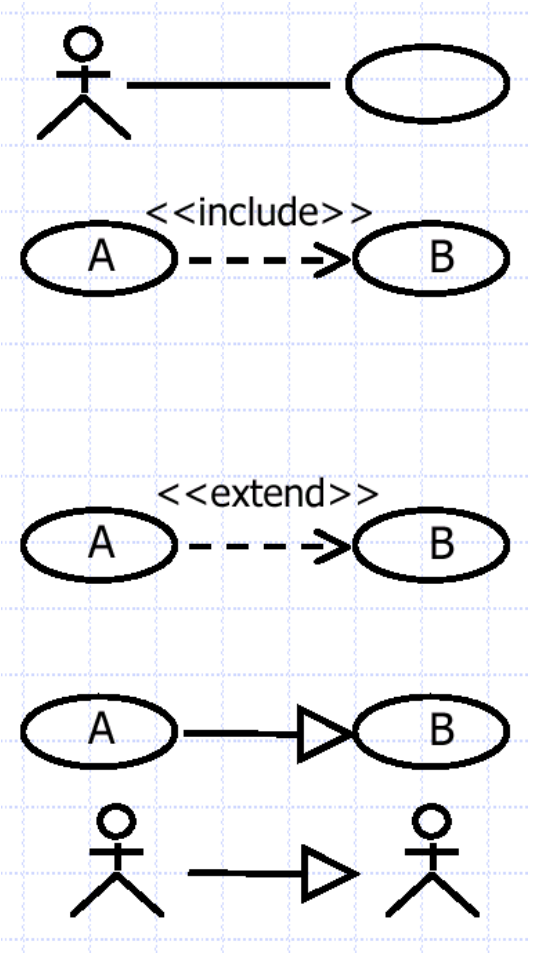
- "Il test della dimensione"
  - Errore comune: fare UC di un singolo passo, all'interno di una sequenza di passi correlati.
    - Es: "Preme bottone *Salva*"

- "Il test del Capo"
  - Immaginare che il vostro capo vi chieda "Cosa hai fatto tutto il giorno?" e voi rispondete col nome di un caso d'uso. Se il capo è contento, è un buon caso d'uso.
    - Es. di errore: "Seleziona Stampa dal menù"

Diagramma dei Casi d’Uso

- Contiene
  - Attori
  - Casi d’uso
  - Relazioni
- Usato per modellare il contesto di un sistema
  - Gli attori sono esterni al sistema
  - I casi d’uso sono all’interno del sistema
- Usato per modellare (visualmente) i requisiti funzionali
  - Ogni caso d’uso corrisponde a uno o più requisiti funzionali
  - Ogni caso d’uso è coinvolto in una qualche relazione
- Diversi livelli di dettaglio
  - Decomposizione gerarchica

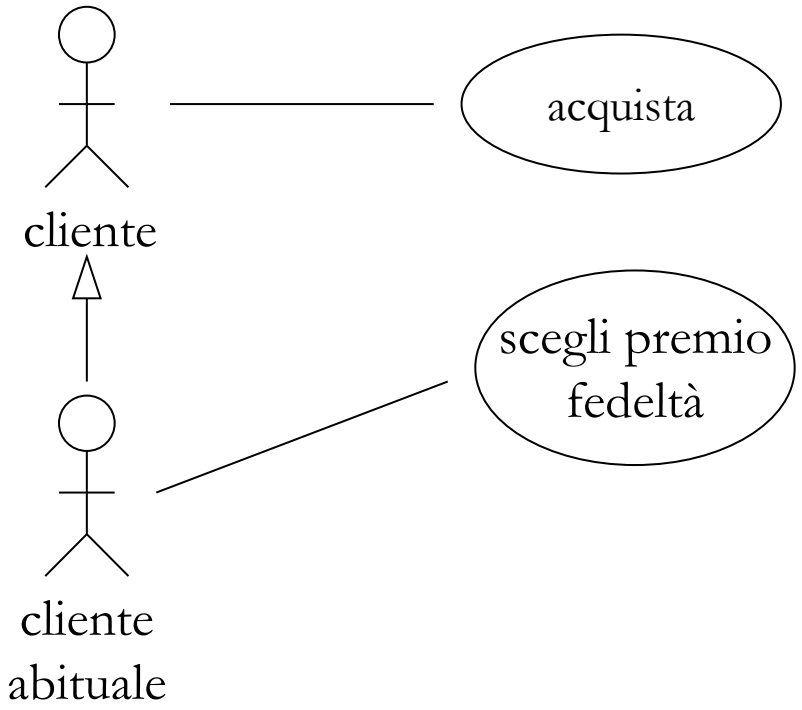
Possibili relazioni negli UCD



- Association: identifica relazioni semplici tra attori e casi d'uso
- Include: fattorizza comportamenti comuni. Utile per evitare di riscrivere testo
- Extend: identifica varianti. A può essere visto come una variante di B
- Generalization si applica sia ad attori che a use case. A eredita il comportamento di B. A può essere sostituito ad ogni occorrenza di B

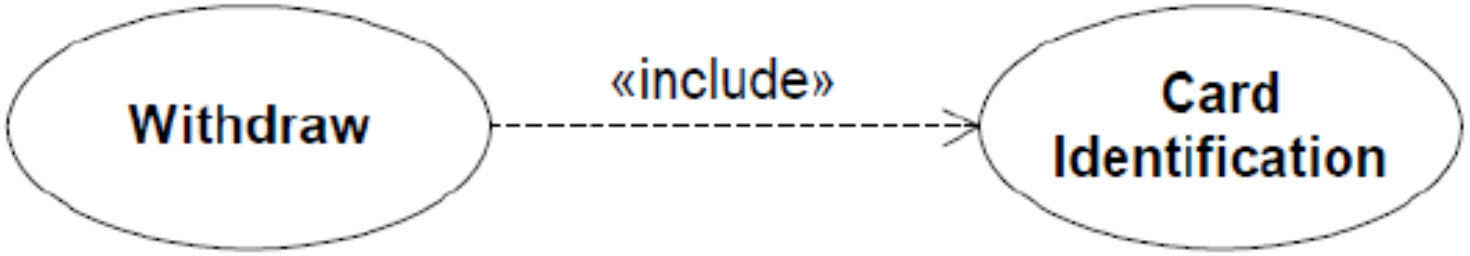
Generalizzazione tra attori

- La relazione di generalizzazione fra attori si applica quando un attore è un sottotipo di un altro e questo comporta delle differenze nel rapporto con il sistema.
- La relazione viene indicata da una freccia con la punta non riempita



Relazioni fra use cases: <<include>>

- The Include relationship is intended to be used **when there are common parts of the behavior of two or more UseCases**.
- This common part is then extracted to a separate UseCase, to be included by all the base UseCases having this part in common.

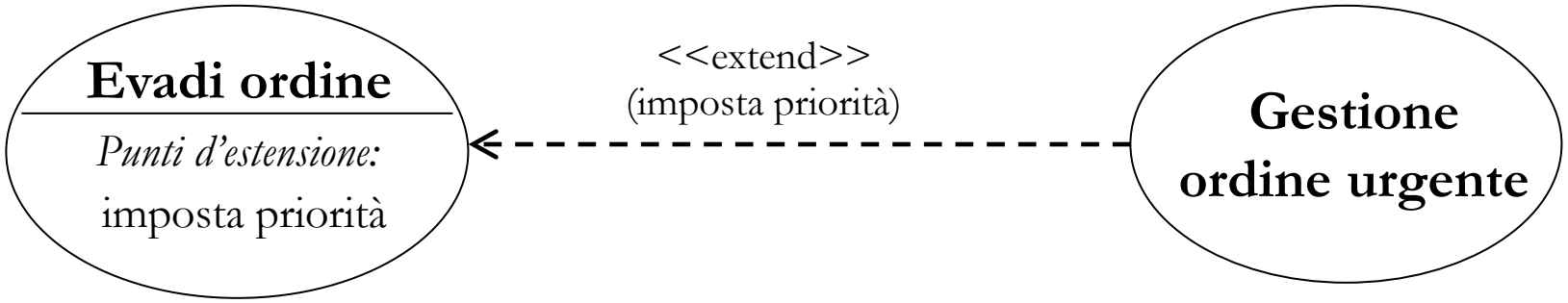


Relazioni fra use cases: <<include>> (2)

- The primary use of the Include relationship is for **reuse of common parts**
  - What is left in a base UseCase is usually not complete in itself but dependent on the included parts to be meaningful.
- This is reflected in the direction of the relationship, indicating that the base UseCase depends on the addition but not vice versa.
- Può essere usata per migliorare la decomposizione funzionale o per esplicitare il riuso di UC:
  - Problema Decomposizione Funzionale: una funzione nella definizione originale del problema è troppo complessa per essere descritta atomicamente
  - Riuso: molti UC eseguono dei sottopassi in comune.

Relazioni fra use cases: <<extend>>

- Extend is intended to be used when there is **some additional behavior that should be added**, possibly conditionally, to the behavior defined in one or more UseCases.
- The extended UseCase is defined independently of the extending UseCase and is meaningful independently of the extending UseCase.
- On the other hand, the extending UseCase typically defines behavior that may not necessarily be meaningful by itself.



Descrizione Testuale degli UC

Descrizione testuale degli UC

- Per **ogni** Use Case nel diagramma ci vuole **una** descrizione testuale dettagliata
- Obiettivo è specificare in ogni aspetto l'interazione attore(i)/sistema, dettagliando la funzionalità dal punto di vista dell'Attore.
  - Casual representation: Testo Libero
  - Fully Dressed Use Case: Template da riempire
- Esistono molte rappresentazioni Fully Dressed, concettualmente simili tra loro

Descrizione testuale degli UC (2)

- Nella sua forma più generale, una descrizione di uno Use Case 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 Use Case;
  3. Una descrizione di cosa può causare errori e come possono essere gestiti i problemi risultanti;
  4. Una descrizione dello stato del sistema al termine dello Use Case.

Il Template di A. Cockburn

USE CASE *#x*	*NOME UC*			
Goal in Context	*Descrizione dell'obiettivo di questo UC*			
Preconditions	*Tutte le condizioni che devono valere per far partire lo UC*			
Success End Condition	*Stato in cui si deve trovare il contesto se lo UC è andato a buon fine*			
Failed End Condition	*Stato in cui si deve trovare il contesto se lo UC non è andato a buon fine, con motivo del problema*			
Primary Actor	*Attore principale dello UC*			
Trigger	*Azione dell'attore principale che avvia lo UC*			
DESCRIPTION	**Step n°**	**Attore 1**	**Attore *n***	**Sistema**
	1	*Azione trigger*		
	2			*Risposta*
	..	*Azione 2*		
	..		*…*	*…*
	n			*Azione finale*

Il Template di Cockburn (cont.)

EXTENSIONS	**Step**	**Attore 1**	**Attore n**	**Sistema**
	m <condition>	*…*	*…*	*…*
	…	*..*	*…*	*…*
	…			*Azione finale*
SUBVARIATIONS	**Step**	**Attore 1**	**Attore n**	**Sistema**
	n	*Branching Action*	*…*	*…*
	…	*..*	*…*	*…*
	Step a cui ci si ricollega			
OPEN ISSUES -	*Elencare tutti gli aspetti ancora da chiarire. Alla consegna del doc non ci devono essere open issues*			
Due Date	*Data entro cui queste funzionalità devono essere implementate*			

Guida alla scrittura di Casi d’Uso

- I nomi dei casi d’uso dovrebbero sempre includere verbi
- I nomi di Attori dovrebbero essere sostantivi
- I casi d’uso devono iniziare con un’azione di un attore (trigger)
- Le relazioni causali tra passi successivi dovrebbero essere chiare
- Un caso d’uso dovrebbe descrivere una transazione utente completa
- Le eccezioni dovrebbero essere descritte nelle sezioni apposite
- Un caso d’uso non dovrebbe descrivere un’interfaccia del sistema (utilizzare mock-up!)
- Un caso d’uso non dovrebbe superare due o tre pagine

Use case diagram: riassunto

- I casi d'uso rappresentano le funzionalità di alto livello offerte dal sistema ai vari attori.
- La maggior parte dei casi d'uso vengono generati in fase di analisi, ma altri ne possono emergere in fase di sviluppo
- I casi d'uso rappresentano un punto di vista esterno al sistema → NO dettagli implementativi
- I casi d'uso sono espressi in forma grafica e testuale, comprensibile ai "non addetti ai lavori" e rappresentano, quindi, il sistema più semplice per discutere con gli stakeholders al fine di scoprire tutti i requisiti

Gli errori più comuni – I

- Modellare operazioni che non coinvolgono il sistema
  - Ad esempio, se il sistema è il software per gestire prestiti libri e il cliente deve consegnare un modulo cartaceo all'impiegato, questa operazione non coinvolge il sistema, e pertanto non va modellata.

Gli errori più comuni – II

- Introdurre generalizzazioni improprie fra attori
  - Ogni attore deve avere use cases associati, e l'attore sottotipo può dare inizio a tutte gli use case del padre.
  - Se un attore figlio non ha use case propri la generalizzazione è inutile, oppure nel diagramma mancano degli use cases

  - Tutti gli attori figli hanno use case propri?

Gli errori più comuni – III

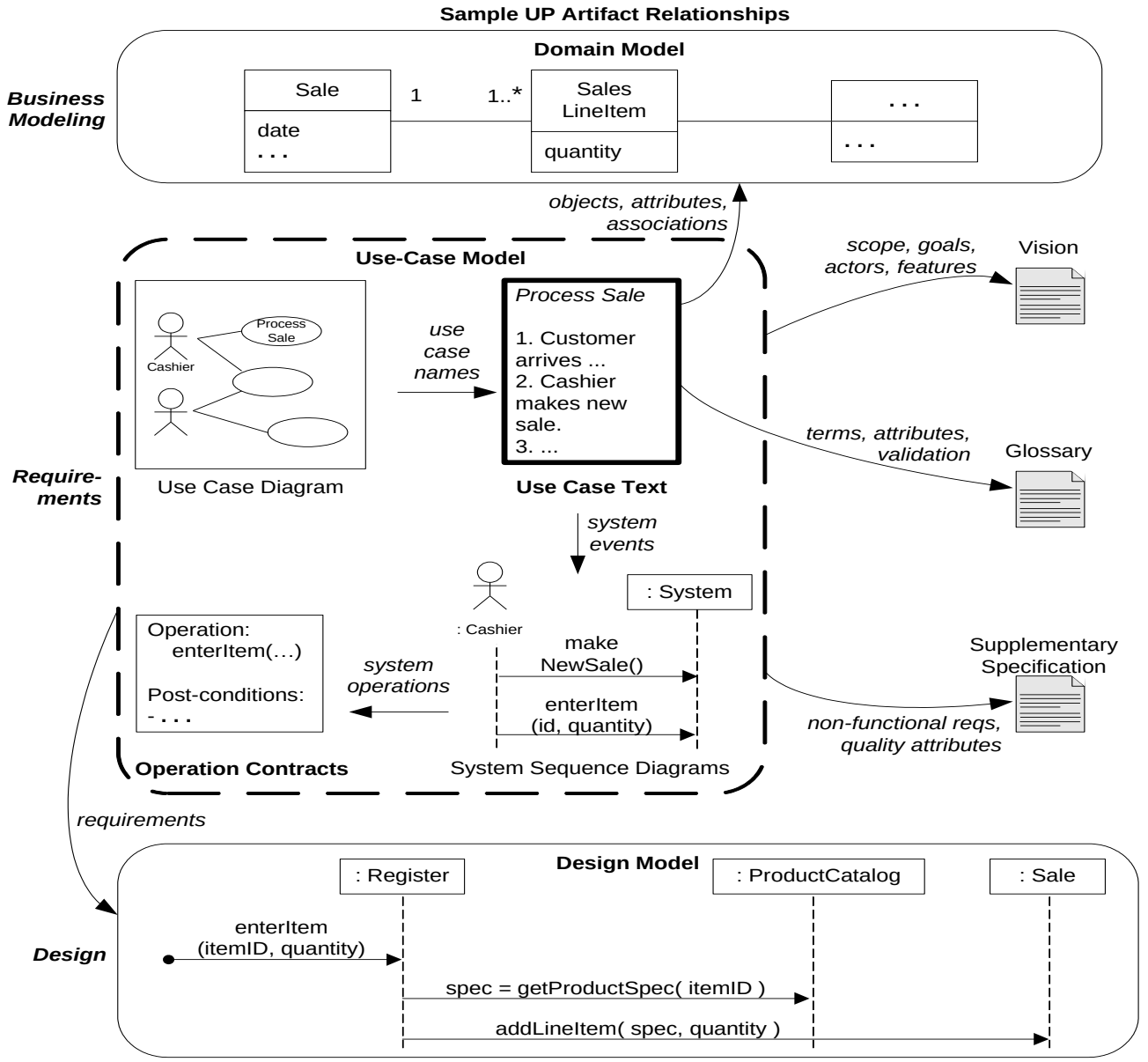
- Il diagramma è troppo complesso
  - Bisogna fare un uso moderato delle relazioni fra use case e delle generalizzazioni fra attori
  - La modellazione delle deve avvenire ad un grado di astrazione sufficientemente elevato; non è questo il diagramma da usare per esprimere tutti i dettagli contenuti nelle specifiche in linguaggio naturale
  - Occorre essere ordinati (evitare linee che si intersecano, ecc...)
- **Un diagramma complesso è indice di un cattivo analista.**

Gli errori più comuni – IV

- Relazioni errate
  - L'*include* non è la via corretta per rappresentare relazioni temporali tra Use Case
    - Es. di errore: In un sito di commercio elettronico, se il caso d'uso “Effettua acquisto” includesse "Login", significa che l'utente deve fare un login per ogni acquisto, anche all'interno della stessa sessione!
    - Non devo rifare il login se effettuo due acquisti di seguito!
  - In questo caso si usano le **precondizioni** nella descrizione testuale

Il Documento dei Requisiti Software

Artefatti coinvolti nel RAD



Documento di Analisi dei Requisiti (RAD)

- I risultati della raccolta dei requisiti e dell'analisi sono documentati nel RAD
- Il RAD descrive completamente il sistema in termini di requisiti funzionali e non funzionali e serve come base del contratto tra cliente e sviluppatori.
- I partecipanti coinvolti sono: cliente, utenti, project manager, analisti del sistema, progettisti
- La prima parte del documento, che include Use Case e requisiti non funzionali, è scritto durante la raccolta dei requisiti
- La formalizzazione della specifica in termini di modelli degli oggetti è scritto durante l'analisi