UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

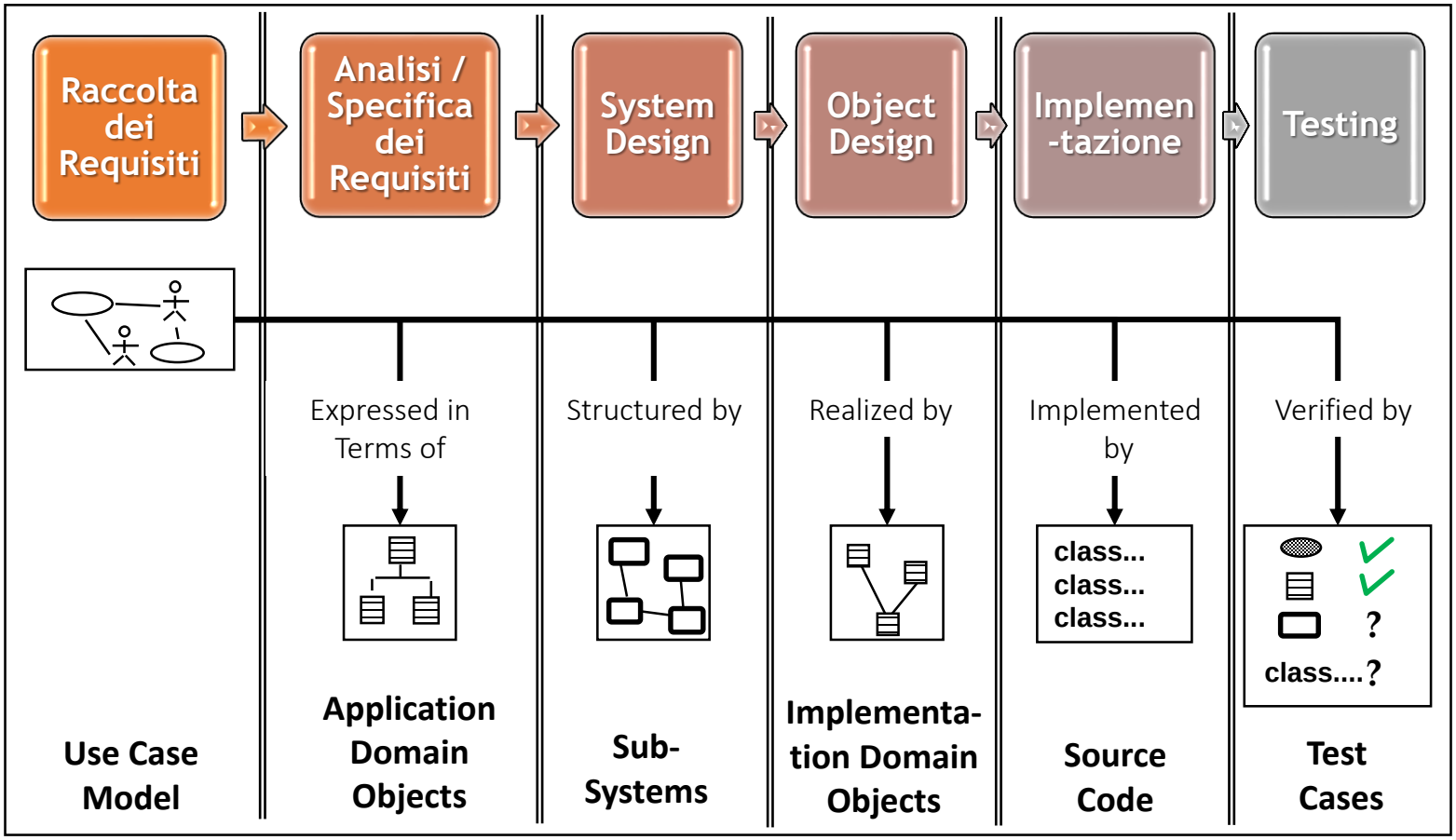
Ingegneria del Software – Il Documento dei Requisiti SW

Prof. Sergio Di Martino

Obiettivi della lezione

- Capire cosa deve contenere un Documento dei Requisiti
- Comprendere le tecniche per la raccolta dei requisiti
  - Scenari → Use Cases
- Introduzione a UML
- Comprendere le tecniche per l'analisi / specifica dei requisiti
  - Use Cases
  - Diagrammi di Cockburn

Ciclo di Vita del Software



Il Documento dei Requisiti Software

Documento dei requisiti SW

- Formalizzare i fabbisogni del cliente relativamente al sistema da sviluppare, in modo **non ambiguo**
- **Cliente**, **utenti** e **sviluppatori** contribuiscono alla stesura del documento di specifica dei requisiti
  - Può essere usato come contratto tra cliente e sviluppatori
- Il documento prevede vari livelli di raffinamento, dal linguaggio naturale, al linguaggio strutturato, ai modelli UML.
  - Tutti i documenti rappresentano la stessa informazione ma sono scritti usando linguaggi diversi, per utenti diversi

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?
  - And what if these situations are combined?

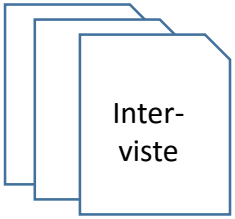
Requirement Engineering

- **L'ingegneria dei requisiti** comprende tutte le attività che si occupano di requisiti:

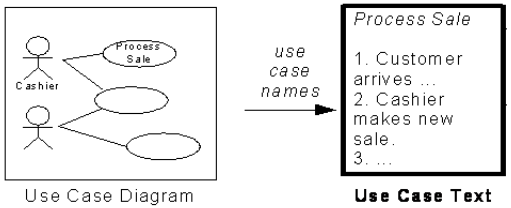
1. Raccolta (scoperta) dei Requisiti (**Requirement Elicitation**)
   - Vengono intervistati gli stakeholders per avere un insieme di requisiti del nuovo sistema
2. Analisi dei Requisiti (**Requirement Analysis**)
   - Le informazioni raccolte vengono organizzate in un documento, con supporto di diagrammi di alto livello (UML Use Cases + Cockburn)
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

Requirement Engineering

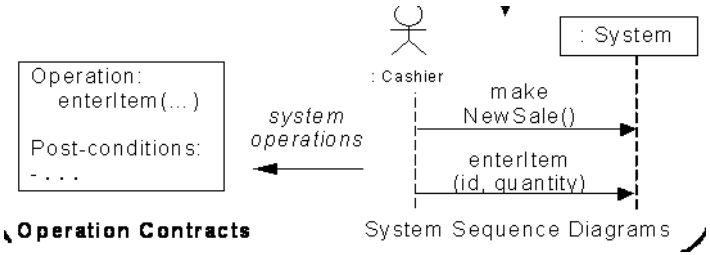
1. **Requirement Elicitation**



2. **Requirement Analysis**



3. **Requirement Specification**



4. **Requirement Validation**

Requirement Elicitation

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
- Interviste
  - 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

Come raccogliere i requisiti (2)

- Utilizzare gli “scenari”
  - Descrizioni di esempi tipici di interazione col sistema che si intende sviluppare
  - Descrizioni di esempi reali invece che di descrizioni astratte
  - L’astrazione va fatta dall’analista dei requisiti
- Utilizzare i Mock-up
  - Rappresentazione statica dell’interfaccia, intesa come prototipo per avere feedback dall’utente
  - Realizzata con «wire-frame»
  - Esempio…

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)

Requirement Analysis

Modelli del sistema

- Nella Requirement Analysis si trovano i primi **modelli** del sistema
- Un modello è una semplificazione della realtà
- I modelli:
  - aiutano a rappresentare un sistema come è o come vorremmo che fosse
  - Permettono di comunicare in maniera non ambigua
  - forniscono una guida formale nella costruzione di un sistema
  - documentano le decisioni prese

Vantaggi della modellazione

- Divide et impera: tramite i modelli ci si focalizza su un solo aspetto alla volta
  - ogni modello può essere espresso a differenti livelli di precisione
- Ogni sistema non banale richiede non un solo modello ma un piccolo insieme di modelli, che possono essere costruiti e studiati separatamente, ma che sono strettamente interrelati

Unified Modeling Language (UML)

- linguaggio (e notazione) universale, 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

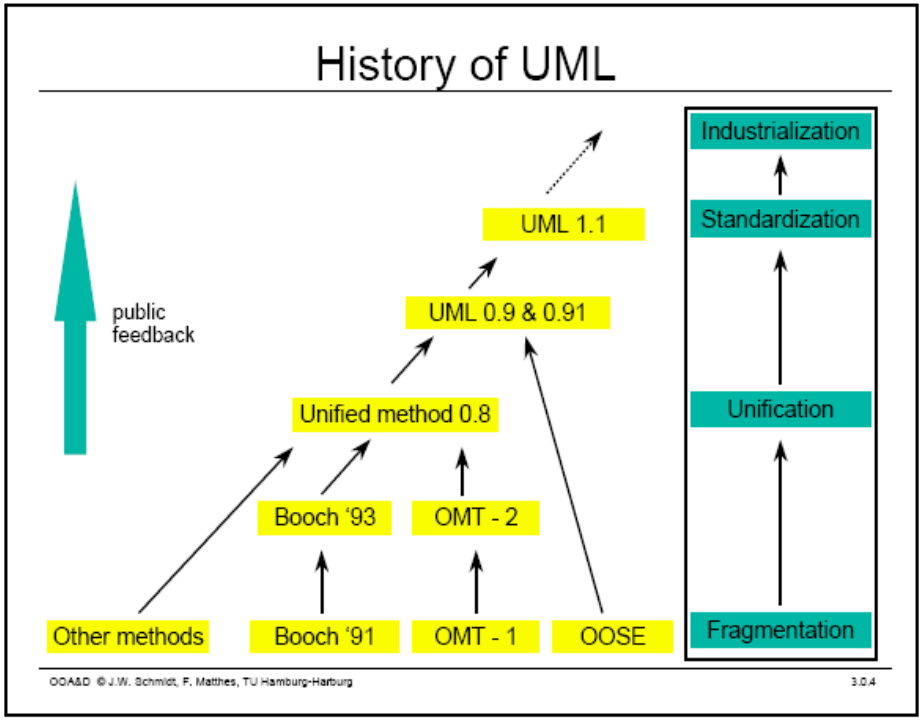
- E’ un linguaggio di modellazione e specifica di sistemi, basato sul paradigma object-oriented
- Serve a specificare le caratteristiche di un nuovo sistema, oppure a documentarne uno già esistente
- E’ uno strumento di comunicazione tra i diversi ruoli coinvolti nello sviluppo e nell’evoluzione dei 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

- Non è una “metodologia”
- E’ un linguaggio, non un metodo completo
  - 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

Breve Storia

- I primi linguaggi di modellazione O-O risalgono agli anni '80.
  - Notazioni per descrivere la struttura di un software O-O ed il suo comportamento dinamico.
- La proliferazione di notazioni portò alla “guerre dei metodi”.
  - OMT (Object Modeling Technique) di Rumbaugh,
  - Il metodo Booch di Grady Booch.
  - OOSE (Object Oriented Software Engineering).
- A metà degli anni '90 diversi metodi iniziarono a fondersi.
  - Possibilità di integrazione dei principali formalismi in una notazione universale.



Breve Storia (2)

- Nel 1995 il progetto fu accolto sotto l'egida dell'OMG (Object Management Group).
  - L'OMG raccolse tutti i principali metodologisti del settore in un incontro internazionale per discutere della notazione unificata.
  - Booch, Rumbaugh e Jacobson misero a punto le release 0.9 e 0.91.
- Questo gruppo esteso realizzò, nel 1997, UML 1.0, che fu sottoposto alla OMG
- La release 1.1 di UML contribuì a consolidare la semantica del linguaggio.
- La versione 2.0 di UML, ufficializzata nel 2005, presenta numerose novità:
  - introduzione del linguaggio formale OCL come parte integrante di UML
  - numerosi nuovi elementi per la costruzione dei diagrammi tradizionali
  - alcuni nuovi tipi di diagrammi

Panoramica
su UML

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.

Altra classificazione dei Diagrammi UML (v. 2.x)

- diagrammi “strutturali”:
  - diagramma delle classi (class)
  - diagramma degli oggetti (object)
  - diagramma dei componenti (component)
  - diagramma delle strutture composite (composite structure)
  - diagramma di deployment(deployment)
  - diagramma dei package (package)
- diagrammi “comportamentali di interazione”:
  - diagramma di sequenza (sequence)
  - diagramma di comunicazione (communication)
  - diagramma dei tempi (timing)
  - diagramma di sintesi dell’interazione (interaction overview)
- diagrammi “comportamentali”:
  - diagramma dei casi d’uso (usecase)
  - diagramma di stato (statechart)
  - diagramma delle attività (activity)

UML - considerazioni

- UML può 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

Requirement Analysis e Use Case Diagrams

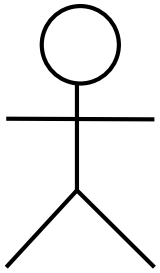
Fasi della Requirement Analysis

1. Identificare gli Attori
   - Individuare le differenti classi di utenti che il sistema dovrà supportare
2. Identificare gli Scenari
   - Realizzare scenari dettagliati che descrivono le funzionalità fornite dal sistema.
   - Gli scenari sono esempi concreti di utilizzo del sistema, descritti in termini di interazione tra utente e sistema, e servono per comprendere il dominio di applicazione.
3. 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)

4. 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
5. 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.

1- Identificare gli Attori



Cassiere

- 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

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*

2- Identificare gli 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**
- *Scenario principale di successo:*
  - Un cliente arriva alla cassa con alcuni articoli da restituire. Il cassiere usa il sistema POS 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'attore vuole che esegua il sistema?
  - Quali informazioni saranno create, memorizzate, modificate, cancellate, o aggiunte dall'utente nel sistema?
  - Di quali cambiamenti esterni l'attore deve informare il sistema?
  - Di quali cambiamenti o eventi del sistema deve essere informato l'attore?

3 - Identificare 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 i possibili 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)
- Si modella sempre con UML ***e*** una descrizione testuale

3 - 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

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

UML e gli Use Case Diagrams

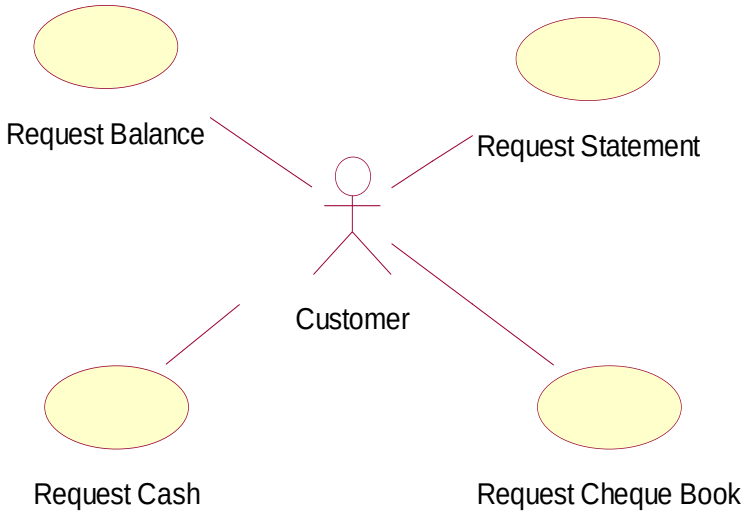
Definition of Use Case

- The oval represents a unique use-case which attempts to capture high level requirements or functionality that our system must provide to its users. As such, all use-cases must be initiated by Actors.
- Each use-case is documented elsewhere with a detailed description of the interaction between user and system and the benefits to the user.

Definition of Actor

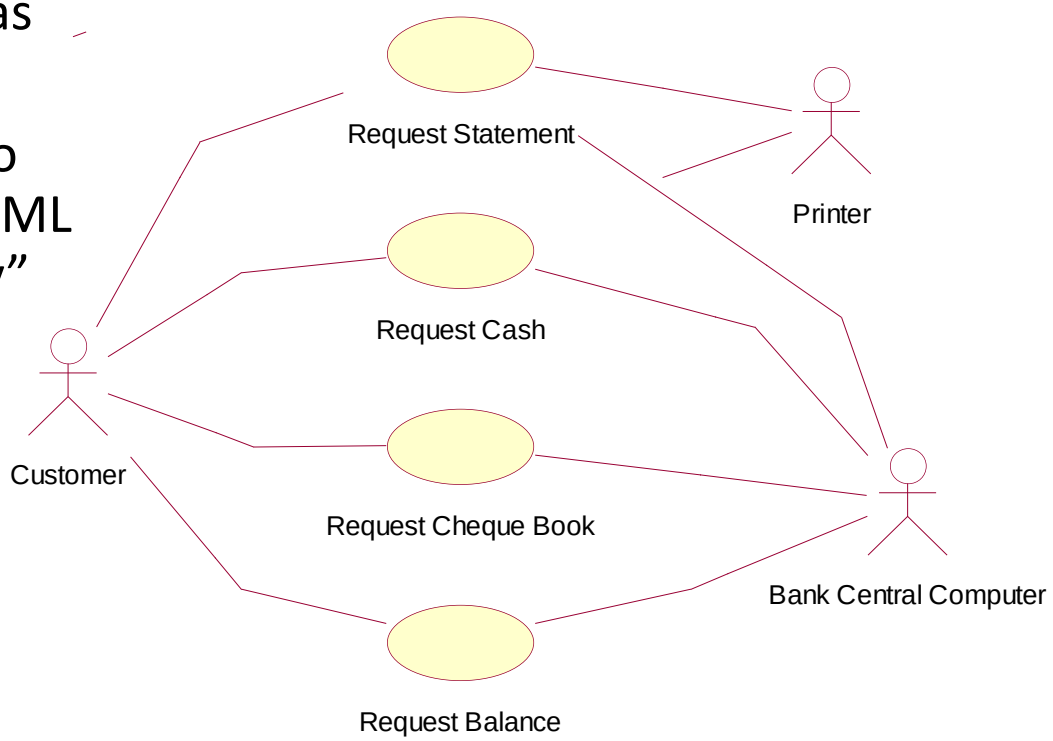
- An Actor represents an external entity outside the domain of the system we are modelling. Most commonly these are the users of the system who initiate one or more use-cases and are typically people, but could be other hw/sw.

- An Actor must be somebody/something that initiates an interaction with the system and gets some measurable benefit from that interaction.



Secondary Actors

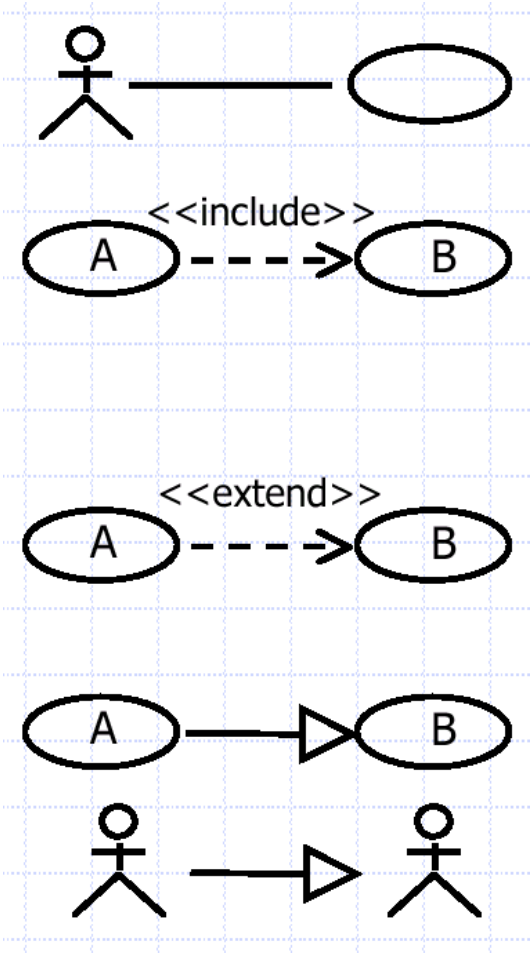
- In an Atm, would we show the Bank Computer and Printer as Actors?
- Possibly, but because they do not initiate a use-case, the UML refers to them as “secondary” actors..



Possibili relazioni negli UCD

- Tra attori e Use Case
  - Associazione
- Tra attori
  - Generalizzazione
- Tra Use Case
  - Generalizzazione
  - Inclusione (*include*)
  - Estensione (*extend*)

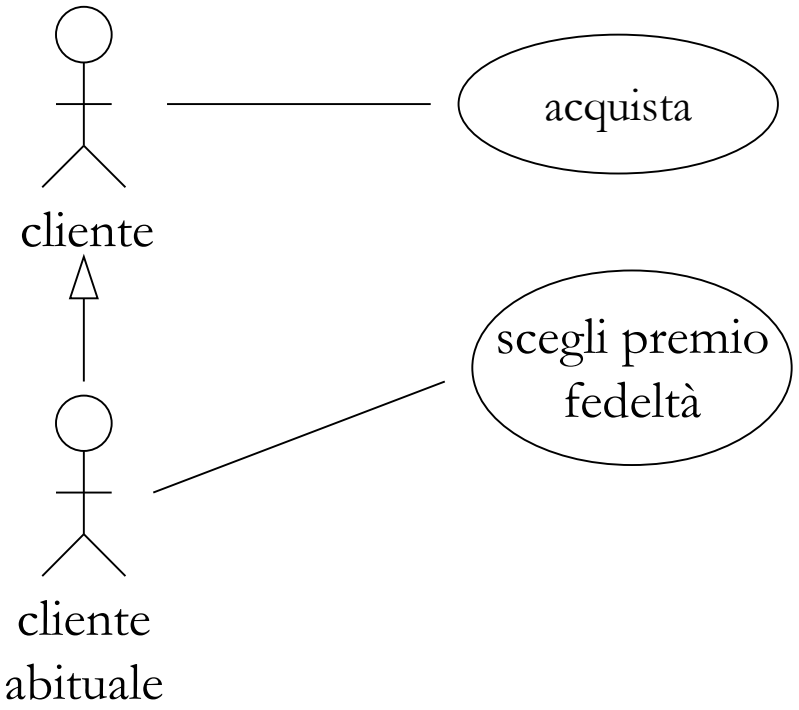
Possibili relazioni negli UCD



- Association: identifica relazioni semplici tra attori e casi d'uso
- Include: fattorizza proprietà comuni. Utile per riusare comportamenti
- Extend: identifica comportamenti simili (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

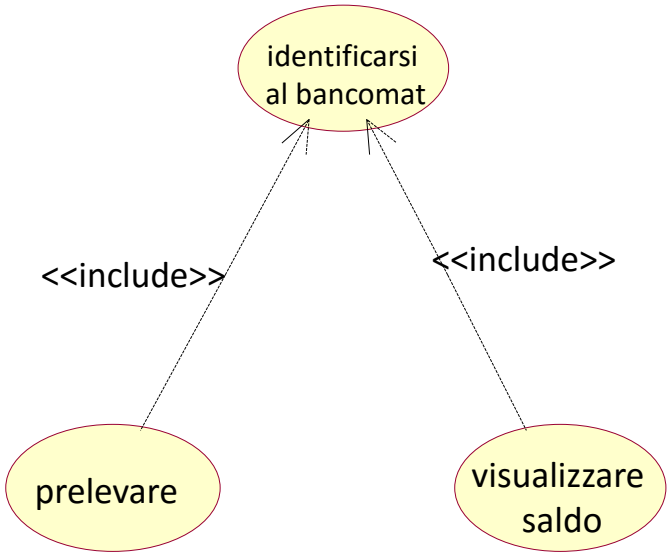
Generalizzazione tra attori

- 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>>

- La relazione di **inclusione** indica che la funzionalità rappresentata in uno UC include completamente la funzionalità rappresentata dall'altro
  - Si indica con una freccia aperta con linea tratteggiata sormontata dallo **stereotipo** *<<include>>*
  - Gli UC che includono devo necessariamente eseguire tutti i passi dello UC incluso

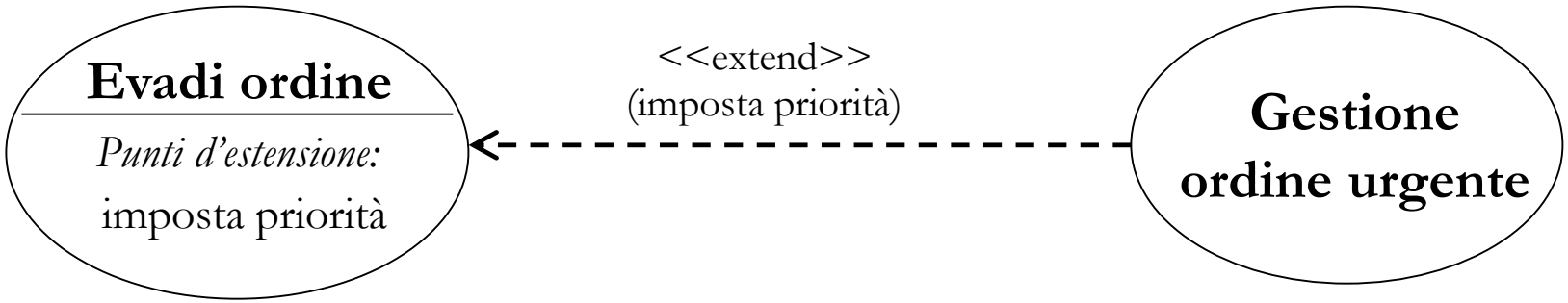


Relazioni fra use cases: <<include>> (2)

- 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.

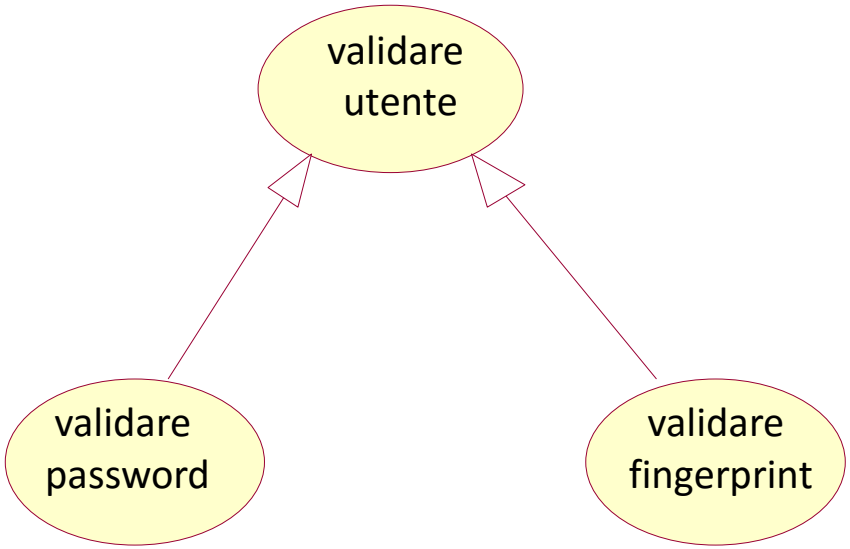
Relazioni fra use cases: <<extend>>

- La relazione di estensione accade quando uno UC estende il comportamento di un altro.
  - Lo use case base deve specificare graficamente i punti di estensione (extension points) cioè i punti in cui il comportamento viene esteso.
  - Si indica con una freccia aperta con linea tratteggiata sormontata dallo stereotipo <<extend>>

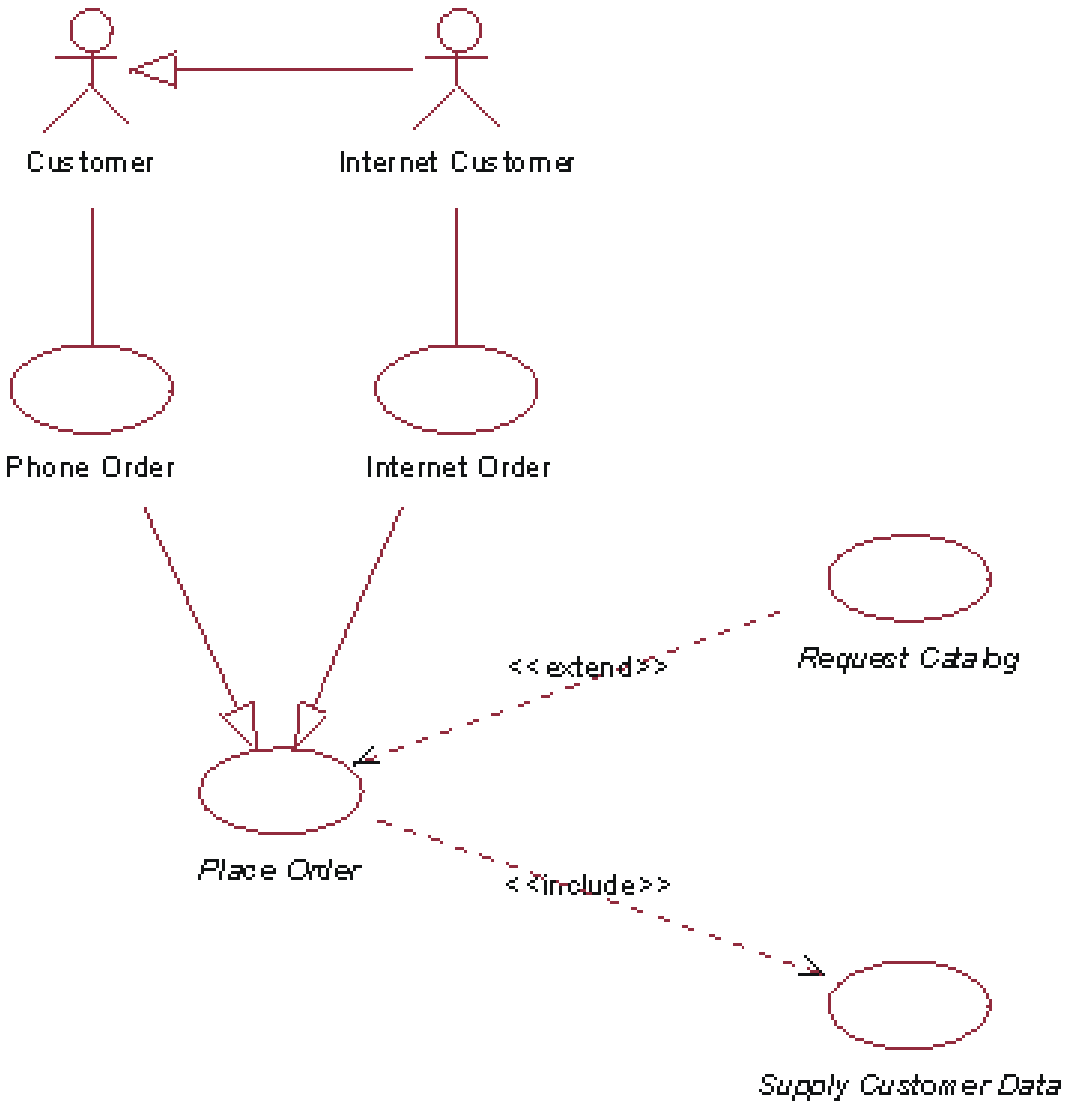


Relazioni fra use cases - II

- La relazione di generalizzazione fra use cases si ha quando un caso d'uso è un caso particolare di un altro, per questo è utile per rappresentare i percorsi alternativi di un'interazione complessa con il sistema.
  - Si indica con una linea continua e freccia con la punta non riempita



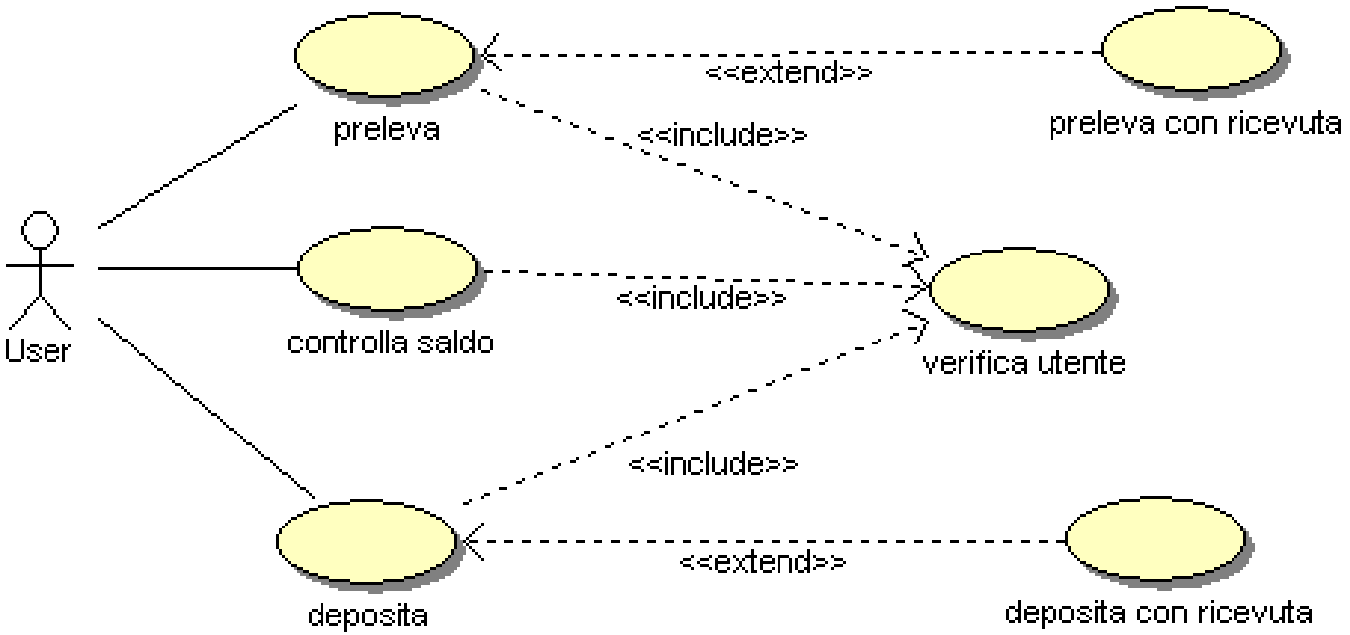
Leggere uno UCD



Esercizio: Sportello Bancomat

- Vi si chiede di realizzare il sw per uno sportello bancomat automatico
  - L'utente deve essere in grado di depositare assegni sul suo conto
  - L'utente deve essere in grado di prelevare contanti dal suo conto
  - L'utente deve poter interrogare il Sistema sul saldo del suo conto
  - Se lo richiede, l'utente deve poter ottenere la ricevuta per la transazione.
  - I tipi di transazione sono prelievo o deposito.
  - La ricevuta deve indicare la data della transazione, il numero del conto, il saldo precedente e successivo la transazione
  - Dopo ogni transazione, il nuovo saldo deve essere visualizzato all'utente

Esercizio: Una Possibile Soluzione



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 prototipi 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
- Ogni caso d'uso è dettagliato per mezzo di opportuni formalismi testuali
  - Es: Tabelle di Cockburn
- 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

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

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, and worth telling the reader?
Success Guarantee	What must be true on successful completion, and worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Miscellaneous	Such as open issues.

Il Template di A. Cockburn

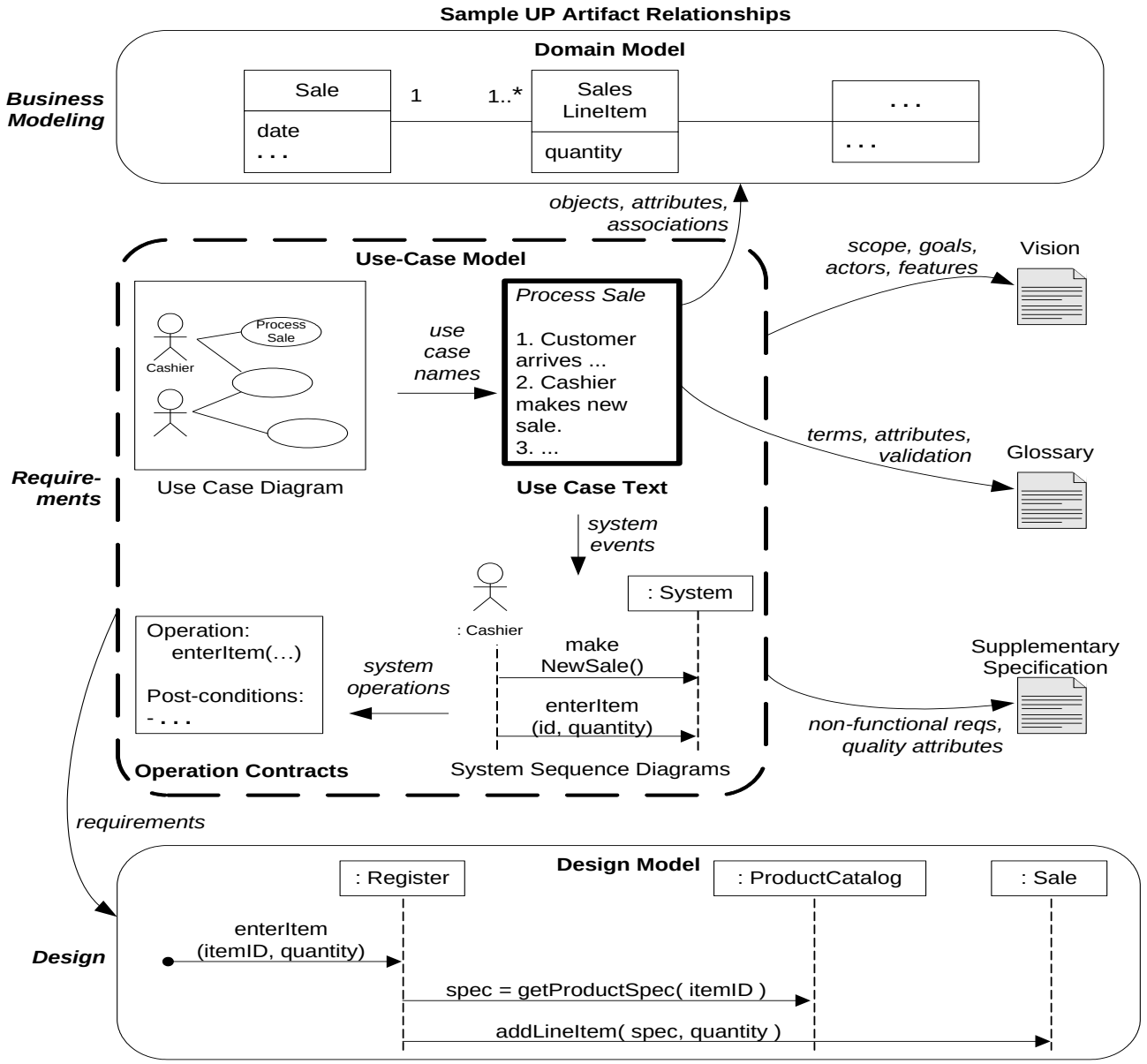
USE CASE *#x*	*NOME UC*			
Goal in Context	*Descrizione dell'obiettivo di questo UC*			
Scope & Level	*System Under Design; Level* = "user-goal" or "subfunction"			
Preconditions	*Tutte le condizioni che devono valere per far partire lo UC*			
Success End Condition	*Stato in cui si deve trovare il contesto 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*			
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 completate*			

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