

APS - Analisi e Progettazione del Software

Elia Ronchetti

@ulerich

2022/2023

Indice

1	Introduzione	4
1.1	Analisi e Progettazione orientata agli oggetti	4
1.2	Che cosa è UML	6
1.2.1	Tre modi per applicare UML	6
1.2.2	Vantaggi della mdoellazione visuale	7
2	Processi per lo sviluppo del software	8
2.1	Che cos'è UP	8
2.2	Il processo a cascata	9
2.3	Sviluppo iterativo ed evolutivo	9
2.3.1	Vantaggi dello sviluppo iterativo	10
2.3.2	Feedback e adattamento	10
2.3.3	Durata delle iterazioni e timeboxing	10
2.3.4	Sviluppo iterativo e flessibilità del codice e del progetto	11
2.4	Come eseguire l'analisi e progettazione in modo iterativo ed evoluto	11
2.4.1	Non cambiare gli obiettivi dell'iterazione	11
2.4.2	Iterazioni - Perché sono la chiave per UP	11
2.5	Le fasi di UP	12
2.5.1	Fasi e dicipline (o flussi di lavoro)	12
2.6	SCRUM	14
3	Analisi dei requisiti	15
3.1	Ideazione	15
3.2	Che sono sono i requisiti	16
3.2.1	Requisiti funzionali	16
3.2.2	Requisiti non funzionali	16
3.2.3	Quali sono i modi validi per troavare requisiti	17
3.3	Requisiti e principali elaborati di UP	17

4	Casi d'uso	19
4.1	Attori, scenari e casi d'uso	19
4.1.1	Perchè i casi d'uso	19
4.2	Tipi di Attore	20
4.3	Tre formati comuni per i casi d'uso	20
4.4	Come trovare i casi d'uso	21
4.4.1	Verificare l'utilità dei casi d'uso	21
4.4.2	Livello dei casi d'uso	21
4.5	Diagramma dei Casi d'uso	22
5	Modellazione di dominio	23
5.1	L'elaborazione	23
5.2	L'analisi a oggetti - Modellazione di Dominio	24
5.3	Come creare un modello di dominio	26
5.4	Attributi	28
5.5	Considerazioni finali	28
6	Diagrammi di Sequenza di Sistema (SSD)	30
6.1	Che cosa sono gli eventi e le operazioni di sistema	30
6.2	SSD e Glossario	32
6.3	Lo scopo di un SSD	32
7	Contratti delle operazioni di sistema	33
7.1	Le sezioni di un contratto	33
7.2	Che cos'è un'operazione di sistema	33
7.3	Post-condizioni	34
7.3.1	Possibili cambiamenti di stato	34
7.3.2	Aggiornamento del modello di dominio	34
7.4	Utilità e scrittura contratti	35

Capitolo 1

Introduzione

Che cosa sono l'analisi e la Progettazione

- Analisi - Enfatizza un'investigazione di un problema e dei suoi requisiti, anzichè di una soluzione
- La progettazione enfatizza una soluzione concettuale che soddisfa i requisiti del problema

Fare la cosa giusta (analisi) e fare la cosa bene (progettazione)

1.1 Analisi e Progettazione orientata agli oggetti

L'analisi orientata agli oggetti enfatizza sull'identificazione dei concetti o degli oggetti, nel dominio del problema.

La progettazione orientata agli oggetti enfatizza sulla definizione di oggetti software che collaborano per soddisfare i requisiti.

Esempio Oggetti - Aereo, Volo, Pilota ognuno con i propri attributi (tipo basi di dati). Analisi e progettazione hanno obiettivi diversi che vengono perseguiti in modi diversi, sono comunque attività sinergiche.

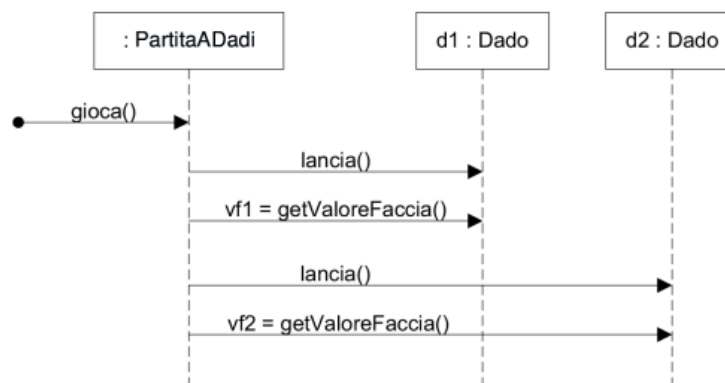
L'OO (Object Oriented) enfatizza la rappresentazione di oggetti

Definizione dei casi d'uso I casi d'uso sono delle storie scritte relative al mondo in cui il sistema viene utilizzato.

Definizione di un modello di dominio Un modello di dominio mostra i concetti o gli oggetti significativi del dominio, con i relativi attributi e associazioni.



Definizione dei diagrammi di interazione Un diagramma di interazione mostra le collaborazioni tra oggetti software.



Definizioni dei diagramma delle classi di progetto Un diagramma delle classi di progetto mostra una vista statica delle definizioni delle classi software, con i loro attributi e metodi.



Lo scopo di tutti questi diagrammi è facilitare la progettazione e il passaggio da idea a codice.

1.2 Che cosa è UML

Unified Modelling Language (UML) è un linguaggio visuale di modellazione dei sistemi e non solo. Rappresenta una collezione di best practice di ingegneria, dimostrate vincenti nella modellazione di sistemi vasti e complessi. Dato che è lo standard de facto favorisce la divulgazione di informazioni nella comunità di ingegneri del software.

UML NON è una metodologia

- UML è un linguaggio visuale
- UP (Unified Process) è una metodologia

UML modella i sistemi come insiemi di oggetti che collaborano tra loro

- Struttura Statica
 - Quali tipo di oggetti sono necessari
 - Come sono tra loro correlati
- Struttura Dinamica
 - Ciclo di vita di questi oggetti
 - Come collaborano per fornire la funzionalità richieste

1.2.1 Tre modi per applicare UML

- UML come abbozzo
- UML come progetto
- UML come linguaggio di programmazione

UML come Abbozzo Diagrammi informali e incompleti (spesso abbozzati a mano) che vengono creati per esplorare parti difficili dello spazio del problema o della soluzione, sfruttando l'espressività dei linguaggi visuali.

UML come Progetto Diagrammi di progetto relativamente dettagliati, utilizzati per il reverse engineering, per la documentazione e per la comunicazione, utilizzati quindi per visualizzare e comprendere meglio il codice esistente mediante diagrammi UML.

UML come linguaggio di programmazione In questo caso il codice viene generato direttamente e automaticamente da UML (approccio ancora in fase di sviluppo). **La modellazione agile enfatizza l'uso di UML come abbozzo**

Due punti di vista per applicare UML

- Punto di vista concettuale - I diagrammi descrivono oggetti del mondo reale o in un dominio di interesse
- Punto di vista software - I diagrammi descrivono astrazioni o componenti software

Entrambi usano la stessa notazione UML.

Il significato di classe nei diversi punti di vista

Nell'UML grezzo, i rettangoli illustrati sono chiamati classi, questo termine racchiude una varietà di casi: oggetti fisici, concetti astratti, elementi software, ecc.

Il significato varia in base al diagramma di utilizzo, per fare chiarezza:

- Classe concettuale - Oggetto o concetto del mondo reale - Significato attribuito nel **Modello di Dominio di UP**
- Classe software - Classe intesa come componente software (es. classe Java) - Significato attribuito nel **Modello di Progetto di UP**

1.2.2 Vantaggi della modellazione visuale

Disegnare o leggere UML implica che si sta lavorando in modo visuale e il nostro cervello è più rapido a comprendere simboli, unità e relazioni rappresentati con una notazione grafica.

Capitolo 2

Processi per lo sviluppo del software

Un processo per lo sviluppo del software (o processo software) definisce un approccio disciplinato per la costruzione, il rilascio e la manutenzione del software.

Definisce chi fa che cosa, quando e come per raggiungere un certo obiettivo.

- Cosa - Sono le attività
- Chi - Sono i ruoli
- Come - Sono le metodologie
- Quando - Riguarda l'organizzazione temporale delle attività

2.1 Che cos'è UP

Un processo per lo sviluppo software descrive un approccio alla costruzione, al rilascio ed eventualmente alla manutenzione del software. **Unified Process (UP)** è un processo iterativo diffuso per lo sviluppo del software per la costruzione di sistemi orientati ad oggetti.

UP è molto flessibile e aperto e incoraggia l'utilizzo di pratiche tratte da altri metodi iterativi come Extreme Programming (XP), Scrum, ecc.

Riassumendo

- UP è un processo iterativo
- Le pratiche UP forniscono una struttura di esempio

- UP è flessibile cioè iterativo, incrementale ed evolutivo
- Pilotato dai casi d'uso (requisiti) e dai fattori di rischio
- Incentrato sull'architettura

2.2 Il processo a cascata

Il processo a cascata è il più vecchio tra quelli utilizzati oggi (definito tra gli anni 60-70). Definito così per il suo ciclo di vita a cascata (o sequenziale), basato sullo svolgimento sequenziale delle diverse attività dello sviluppo del software.

Richiede che i requisiti siano chiari sin dall'inizio e spesso non è così dato che durante la progettazione emergono altri requisiti da parte del cliente.

Perchè il processo a cascata è soggetto a frequenti fallimenti É stata rilevata un'alta percentuale di fallimenti di progetti che utilizzano questo processo e ci sono diversi motivi legati a questo alto tasso di fallimenti.

Il motivo principale è che il processo a cascata presuppone che i requisiti siano prevedibili e stabili e che possano essere definiti all'inizio, ma ciò è falso. É stato dimostrato che ogni progetto software subisce mediamente il 25% di cambiamenti nei requisiti. Addirittura altri studi hanno evidenziato tassi dal 35 fino al 50%, per quanto riguarda i progetti più grandi. Nello sviluppo software il presupposto che i requisiti siano stabili e prevedibili nel tempo è fondamentalmente sbagliato. Nei progetti software il cambiamento è piuttosto una costante.

2.3 Sviluppo iterativo ed evolutivo

Lo sviluppo iterativo ed evolutivo è una pratica fondamentale in molti processi software moderni come UP e Scrum. In questo approccio al ciclo di vita, lo sviluppo è organizzato in una serie di mini progetti brevi di lunghezza fissa chiamati **iterazioni**. Ad ogni fase c'è un affinamento del progetto a seguito di feedback, per questo viene chiamato **sviluppo iterativo e incrementale** o anche **sviluppo iterativo ed evolutivo**.

L'instabilità dei requisiti e del progetto tende a diminuire nel tempo, nelle iterazioni finali è difficile (ma non impossibile), che si verifichi un cambiamento significativo dei requisiti.

2.3.1 Vantaggi dello sviluppo iterativo

- Minor probabilità di fallimento del progetto
- Miglior produttività
- Percentuali più basse di difetti
- Riduzione precoce anziché tardiva dei rischi maggiori
- Progresso visibile fin dall'inizio
- Feedback precoce e conseguente coinvolgimento dell'utente e adattamento
- Gestione della complessità

2.3.2 Feedback e adattamento

Attività fondamentale per il successo, come si struttura? E da dove proviene?

- Feedback proveniente dalle attività di sviluppo
- Feedback proveniente dai test e dagli sviluppatori che raffinano il progetto e i modelli
- Feedback circa l'avanzamento del tema nell'affrontare i requisiti, per raffinare le stime dei tempi e dei costi
- Feedback proveniente dal cliente e dal mercato

2.3.3 Durata delle iterazioni e timeboxing

Una buona pratica dello sviluppo iterativo è il timeboxing: le iterazioni hanno una lunghezza fissata (Molti processi iterativi raccomandano una lunghezza da 2 a 6 settimane).

Passi piccoli, feedback rapido e adattamento sono le idee centrali dello sviluppo iterativo. La durata di un'iterazione, una volta fissata non può cambiare, quando viene fissata viene detta **timeboxed**.

Non permettere al pensiero a cascata di invadere un progetto iterativo

2.4. COME ESEGUIRE L'ANALISI E PROGETTAZIONE IN MODO ITERATIVO ED EVOLUTO

2.3.4 Sviluppo iterativo e flessibilità del codice e del progetto

L'adozione dello sviluppo iterativo richiede che il software venga realizzato in modo flessibile affinché l'impatto dei cambiamenti sia basso. Il codice sorgente deve quindi essere facilmente modificabile e comprensibile (per facilitarne la modifica). **flessibile**

2.4 Come eseguire l'analisi e progettazione in modo iterativo ed evoluto

Un'attività critica nello sviluppo iterativo è la pianificazione delle iterazioni, non bisogna tentare di pianificare tutto il progetto in modo dettagliato fin dall'inizio. I processi iterativi promuovono la pianificazione guidata dal rischio e guidata dal cliente.

2.4.1 Non cambiare gli obiettivi dell'iterazione

Durante un'iterazione non è possibile cambiare i requisiti, dato che sono stati precedentemente fissati durante la pianificazione iterativa e poi bloccati. Questo perché così facendo il Team di sviluppo può lavorare al suo meglio.

2.4.2 Iterazioni - Perché sono la chiave per UP

Le iterazioni sono la chiave per UP, dato che ogni iterazione è come un mini-progetto che include:

- Pianificazione
- Analisi e progettazione
- Costruzione
- Integrazione e test
- Un rilascio

Si arriva al rilascio finale attraverso una sequenza di iterazioni. Le iterazioni possono sovrapporsi e questo è importante, dato che consente lo sviluppo parallelo e il lavoro flessibile in grandi squadre. Ad ogni iterazione si svolge una parte del lavoro totale di ogni disciplina.

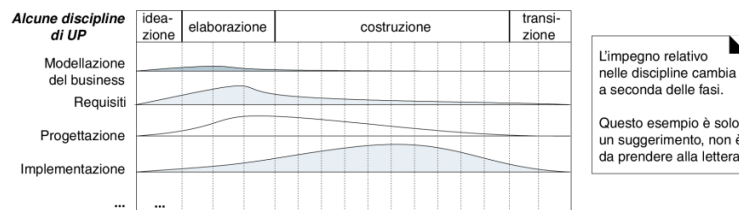
Esempio Si svolgono in parallelo sia modellazione, Progettazione e implementazione, chiaramente a intensità diverse in base all'iterazione (e quindi alla fase del progetto), per esempio il testing sarà maggiore dopo qualche iterazione, dato che ci saranno più cose da testare.

Ogni iterazione genera una **release** che costituisce un insieme di manufatti previsti e approvati. Un incremento è la differenza tra una release e quella successiva. Costituisce un passo in avanti verso il rilascio finale del sistema, per questo UP si chiama iterativo e incrementale.

2.5 Le fasi di UP

- identificazione - Avvio del progetto
- Elaborazione - Realizzazione del nucleo dell'architettura
- Costruzione - Realizzazione delle capacità operative iniziali
- Transizione - Completamento del prodotto

2.5.1 Fasi e discipline (o flussi di lavoro)



Per ogni fase si considera:

- L'attenzione in termini di flussi di lavoro
- L'obiettivo per la fase
- Il milestone alla fine della fase

Il processo con lo scenario di sviluppo di UP è nato per essere personalizzato, per questo tra gli elaborati e le pratiche di UP, quasi tutto è opzionale.

Alcune pratiche e principi sono fissi, come lo sviluppo iterativo e guidato dal rischio e la verifica continua della qualità.

Tutti gli elaborati (modelli, diagrammi, documenti) sono opzionali, con l'ovvia esclusione del codice.

La scelta delle pratiche degli elaborati UP per un progetto può essere scritta in un breve documento chiamato scenario di sviluppo.

Scenario di sviluppo

Disciplina	Pratica	Elaborato Iterazione	Ideazione I1	Elaboraz E1..En	Costr C1..Cn	Transiz T1..T2
Modellazione del business	modellazione agile	Modello di Dominio		i		
	workshop requisiti					
Requisiti	workshop requisiti	Modello dei Casi d'Uso	i	r		
	esercizio sulla visione	Visione	i	r		
	votazione a punti	Specifica Supplementare	i	r		
		Glossario	i	r		
Progettazione	modellazione agile	Modello di Progetto		i	r	
	sviluppo guidato dai test	Documento dell'Architettura Software		i		
		Modello dei Dati		i	r	

i=Inizio, r=raffinamento

Non si è capito lo sviluppo iterativo o UP se

- Si cerca di definire la maggior parte dei requisiti prima di iniziare la progettazione o l'implementazione
- Si pensa che i diagrammi UML e le attività di progettazione devono definire il progetto in dettaglio e che la programmazione è una mera traduzione di questi diagrammi
- Si ritiene che una durata adeguata per un'interazione sia tre mesi anziché tre settimane
- Si prova a pianificare il progetto in dettaglio dall'inizio alla fine, in modo speculativo

Risulta quindi fondamentale ricordarsi che è un metodo per visualizzare il progetto, per poter poi flessibilmente apportare modifiche, deve rimanere quindi agile. Per dettagli vedere il Manifesto Agile.

2.6 SCRUM

Scrum è un metodo agile che consente di sviluppare e rilasciare prodotti software con il più alto valore per i clienti ma nel più breve tempo possibile. Scrum si occupa principalmente dell'organizzazione del lavoro e della gestione di progetti.

Terminologia

- Scrum - Incontro giornaliero del Team SCRUM che esamina i progressi e definisce le priorità del lavoro da svolgere in quel giorno. Incontro breve faccia a faccia con tutto il team.
- ScrumMaster - Responsabile di assicurare che il processo di Scrum sia seguito e guida il team nell'uso efficace di Scrum. Responsabile dell'interfacciamento con il resto dell'azienda e di assicurare che il team non venga deviato da interferenze esterne
- Sprint - Un'iterazione di sviluppo. Le sprint sono solitamente di 2-4 settimane
- Velocity - Una stima di quanto lavoro rimanente un team può fare in un singolo sprint. Fondamentale per capire le prestazioni di una squadra e così migliorarle.
- Team di sviluppo - Un gruppo auto-organizzatore di sviluppatori di software, che non dovrebbe superare le 7 persone. Sono responsabili dello sviluppo del software e di altri documenti essenziali del progetto.
- Incremento potenzialmente rilasciabile - L'incremento software fornito da uno sprint
- Product Backlog - Questo è un elenco di elementi da fare (to do) che il team Scrum deve affrontare
- Product Owner - Può essere un cliente, ma potrebbe essere anche un product manager in una società di software o un rappresentante di altri stakeholder. Identifica le caratteristiche o i requisiti del prodotto.

Capitolo 3

Analisi dei requisiti

3.1 Ideazione

La maggior parte dei progetto richiede un breve passo iniziale in cui si esaminano i seguenti tipi di domande:

- Il progetto è fattibile?
- Comprare e/o costruire?
- Stima approssimativa e non affidabile dei costi
- Dovremmo procedere o fermarci?

L'ideazione non è la fase dei requisiti.

Il problema principale che risolve l'ideazione è il seguente:

Le parti interessate hanno un accordo di base, sulla visione del progetto, e vale la pena di investire un'indagine seria?

Lo scopo è quindi quello di stabilire una visione comune per gli obiettivi del progetto e capire se questo è fattibile.

In questa fase non si utilizza molto UML (verrà utilizzato soprattutto durante l'elaborazione).

Non hai capito l'ideazione se

- Dura più di qualche settimana
- Provi a definire molti requisiti
- Ci si aspetta che i piani e le stime siano affidabili

- I nomi di molti attori e casi d'uso non sono stati identificati
- Troppi casi d'uso sono stati scritti nel dettaglio
- Nessun caso d'uso è stato scritto in dettaglio

3.2 Che sono i requisiti

Un requisito è una capacità o una condizione a cui il sistema e più in generale il progetto, deve essere conforme.

I requisiti sono un aspetto veramente molto importante, si evidenzia addirittura che 34 % delle cause dei fallimenti dei progetti software riguardano l'attività dei requisiti.

Ci sono due tipi principali di requisiti

- Requisiti funzionali (comportamentali) - Descrivono il comportamento del sistema, in termini di funzionalità fornite ai suoi utenti e informazioni che il sistema deve gestire
- Requisiti non funzionali (tutti gli altri requisiti) - Sono relativi a proprietà del sistema nel suo complesso come per esempio sicurezza, pre-setazioni, scalabilità, usabilità...

3.2.1 Requisiti funzionali

Sono funzionalità o servizi che il sistema deve fornire, risposte che l'utente aspetta dal software in determinate condizioni, risultati che il software deve produrre in risposta a specifici input.

Il problema dell'imprecisione nella specifica dei requisiti Requisiti ambigui possono portare a diverse interpretazioni da sviluppatori e utenti. In linea di principio i requisiti dovrebbero essere completi e coerenti, quindi includere la definizione di tutti i servizi richiesti e non essere ambigui.

3.2.2 Requisiti non funzionali

questi definiscono le proprietà e i vincoli del sistema, ad esempio affidabilità, tempi di risposta, oppure vincoli come la capacità dei dispositivi I/O, le rappresentazioni dei dati nelle interfacce di sistema, ecc.

I requisiti funzionali possono essere più critici dei requisiti non funzionali, in caso contrario il sistema potrebbe risultare inutilizzabile.

Obiettivi e requisiti I requisiti non funzionali possono essere molto difficili da stabilire con precisioni e requisiti imprecisi possono essere difficili da verificare.

Un obiettivo può essere un'intenzione generale dell'utente come la facilità d'uso, mentre un requisito non funzionale verificabile è una dichiarazione che utilizza alcune misure oggettivamente verificabili. Gli obiettivi sono utili agli sviluppatori in quanto trasmettono le intenzioni degli utenti del sistema.

Metriche per specificare i requisiti non funzionali

Proprietà	Misura
Velocità	Transazioni elaborate al secondo Tempi di risposta a utenti/eventi Tempo di refresh dello schermo
Dimensione	Mbytes Numero di chip ROM
Facilità d'uso	Tempo di addestramento Numero di maschere di aiuto
Affidabilità	Tempo medio di malfunzionamento Probabilità di indisponibilità Tasso di malfunzionamento Disponibilità
Robustezza	Tempo per il riavvio dopo malfunzionamento Percentuali di eventi causanti malfunzionamento Probabilità di corruzione dei dati dopo malfunzionamento
Portabilità	Percentuali di dichiarazioni dipendenti dall'architettura di destinazione Numero di architetture di destinazione

3.2.3 Quali sono i modi validi per trovare requisiti

- Interviste con i clienti
- Scrivere casi d'uso con i clienti
- Workshop dei requisiti a cui partecipano sia sviluppatori che clienti
- Gruppi di lavoro con rappresentanti dei clienti
- Sollecitare feedback clienti alla di ogni iterazione

3.3 Requisiti e principali elaborati di UP

- Modello dei casi d'uso
- Specifiche supplementari
- Glossario
- Visione
- Regole di Business

Linee guida per la scrittura dei requisiti

- Ideare un formato standard e utilizzare per tutti i requisiti
- Utilizzo del linguaggio in modo consistente. Utilizzare DEVE per requisiti obbligatori, DOVREBBE per requisiti desiderabili
- Evidenziare porzioni di testo per identificare le parti più importanti dei requisiti
- Evitare gergo informatico

Capitolo 4

Casi d'uso

I casi d'uso sono storie scritte, ampiamente utilizzati per scoprire e registrare i requisiti. Un caso d'uso è un dialogo tra un attore e un sistema che svolge un compito.

I casi d'uso non sono elaborati orientati agli oggetti, influenzano però molti aspetti di un progetto, compresa l'analisi e la progettazione orientata agli oggetti. **I casi d'uso sono testo.**

4.1 Attori, scenari e casi d'uso

- Un attore è qualcosa o qualcuno dotato di comportamento - Es. cassiere o sistema di pagamento
- Uno scenario (o istanza del caso d'uso) è una sequenza specifica di azioni e iterazioni tra il sistema e alcuni attori - Descrive una particolare storia nell'uso del sistema, si possono dividere in scenari di successo e di fallimento.
- Un caso d'uso è una collezione di scenari correlati, sia di successo che di fallimento, che descrivono un attore che usa un sistema per raggiungere un obiettivo specifico

Il **Modello dei Casi d'Uso** è l'insieme di tutti i casi d'uso scritti.

4.1.1 Perché i casi d'uso

Si tratta di un metodo semplice per descrivere i requisiti funzionali ed è direttamente comprensibile dai clienti. Inoltre mettono in risalto obiettivi degli utenti e il loro punto di vista. Molto utile per produrre la guida utente e per i test di sistema.

I casi d'uso sono requisiti funzionali Dato che essi indicano cosa deve fare il sistema, un caso definisce un contratto relativo al comportamento di un sistema.

4.2 Tipi di Attore

SuD = Sistema in discussione

- Attore primario: raggiunge obiettivi usando il SuD
- Attore finale: vuole che il SuD sia utilizzato affinché vengano raggiunti i suoi obiettivi (es cliente)
- Attore di supporto: es. servizio pagamento
- Attore fuori scena: ha un interesse nel comportamento del caso d'uso SuD, ma non è attore primario, finale o di supporto (es Governo interessato al pagamento delle imposte)

4.3 Tre formati comuni per i casi d'uso

- Formato breve - Riepilogo conciso di un solo paragrafo, normalmente relativo al solo scenario principale di successo
- Formato informale - Più paragrafi relativi anche agli altri scenari
- Formato dettagliato - Tutti i passi e le variazioni sono scritti nel dettaglio

Formato Dettagliato

Formato dettagliato	
SELEZIONE DEL CASO D'USO	DESCRIZIONE
Nome del Caso d'Uso	Inizia con un verbo
Portata	Il sistema che si sta progettando
Livello	"Obiettivo utente" o "sottofunzione"
Attore Primario	Nome dell'attore primario
Parti Interessate e Interessi	A chi interessa questo caso d'uso e che cosa desidera
Pre-condizioni	Che cosa deve essere vero all'inizio del caso d'uso (e vale la pena di dire al lettore)
Garanzia di successo	Che cosa deve essere vero se il caso d'uso viene completato con successo (e vale la pena di dire al lettore)
Scenario Principale di Successo	Uno scenario comune di attraversamento del caso d'uso, di successo e incondizionato
Estensioni	Scenari alternativi, di successo e di fallimento
Requisiti speciali	Requisiti non funzionali correlati
Elenco delle variabili tecnologiche e dei dati	Varianti nei metodi di I/O e nel formato dei dati
Frequenza di ripetizione	Frequenza prevista di esecuzione del caso d'uso
Varie	Altri aspetti, ad esempio i problemi aperti

Scrivere in uno stile essenziale, come per esempio

- L'amministratore si identifica
- Il sistema autentica l'identità

Scrivere casi d'uso in modo conciso e completo.

Scrivere casi d'uso a scatola nera, ovvero specificare che cosa deve fare il sistema, senza decidere come lo farà. Concentrarsi sulla comprensione di ciò che l'attore considera un risultato di valore.

4.4 Come trovare i casi d'uso

- Scegliere il confine di sistema (Es identificando gli attori esterni)
- Identificare gli attori primari
- Identificare gli obiettivi per ogni attore primario
- Definire i casi d'uso che soddisfano questi obiettivi

Per dettagli leggere Capitolo 7 pag 88 del Larman Link Bookshelf.

4.4.1 Verificare l'utilità dei casi d'uso

Ci sono diversi metodi

- Test del capo
- Test EBP (Elementary Business Process) - Capire se si tratta di un processo elementare e non troppo complesso
- Test della Dimensione - Un buon caso d'uso non dovrebbe essere troppo breve

4.4.2 Livello dei casi d'uso

I casi d'uso possono essere scritti a livelli diversi

- Livello di obiettivo utente
- Livello di sotto-funzione
- Livello di sommario

4.5 Diagramma dei Casi d'uso

usare notazioni diverse per gli attori umani e per quelli che sono sistemi informatici. Unire i vari attori e casi d'uso con associazioni rappresentate da una linea continua. La direzione viene utilizzata nel verso di chi dà inizio all'interazione.

Non viene associata una direzione se entrambe le parti possono dare inizio all'interazione.

Relazioni tra Casi d'Uso

- Include - Relazione tra un caso d'uso base ed un caso d'uso incluso nel caso base
- Extend - Connette un caso d'uso esteso ad un caso d'uso base, aggiunge varianti ad un caso d'uso base e viene inserito solo se la condizione d'estensione è vera.
- Generalization - Un caso d'uso genitore è una generalizzazione di un caso d'uso figlio. Viene eseguito se la condizione di generalizzazione è vera. Possono ereditare, aggiungere sovrascrivere le funzioni del loro genitore. Si può avere anche tra attori questa relazione.

Capitolo 5

Modellazione di dominio

Un caso d'uso o una caratteristica sono spesso troppo complessi per poter essere completati in una sola breve iterazione pertanto le varie parti o scenari possono essere distribuiti su diverse iterazioni.

5.1 L'elaborazione

L'elaborazione è la serie iniziale di iterazioni durante le quali, in un progetto normale:

- Viene programmato e verificato il nucleo, rischioso, dell'architettura software
- Viene scoperta e stabilizzata la maggior parte dei requisiti
- I rischi maggiori sono attenuati o rientrano

Alcune idee e best practice

- Eseguire iterazioni guidate dal rischio, brevi e timeboxed
- Iniziare presto la programmazione
- Progettare, implementare e testare in modo attivo le parti principali
- Effettuare test presto, spesso e in modo realistico
- Adattare in base al feedback proveniente da test, utenti e sviluppatori
- Scrivere la maggior parte dei casi d'uso e degli altri requisiti nel dettaglio

É importante definire le priorità in base al rischio, copertura e criticità.

- Rischio - Comprende tanto la complessità tecnica quanto altri fattori, come l'incertezza dello sforzo o l'usabilità
- Copertura - Indica che le iterazioni iniziali prendono in considerazione tutte le parti principali del sistema
- Criticità (Valore) - Le funzioni che il cliente considera di elevato valore di business

Prima di ogni iterazione viene aggiornata la classifica.

5.2 L'analisi a oggetti - Modellazione di Dominio

Nell'analisi orientata agli oggetti vengono modellati i seguenti aspetti

- Il Dominio informativo, ovvero le tipologie di informazioni che il sistema deve rappresentare e gestire
- Le interazioni fra attori e sistema, ovvero le funzioni
- Il comportamento del sistema, ovvero i cambiamenti nelle informazioni associati a ciascuna funzione

Vengono modellati nella come segue:

- Il dominio informativo è rappresentato mediante un modello oggetti (modello di dominio)
- Le funzioni del sistema sono rappresentate in termini delle operazioni che il Sistema è chiamato a svolgere (**operazioni di sistema**), insieme a una descrizione dell'ordine relativo in cui si possono richiedere queste operazioni **diagrammi di sequenza di sistema**
- Il comportamento è descritto come l'effetto prodotto dall'esecuzione di ciascuna operazione di sistema (**contratti** delle operazioni di sistema)

Modellazione di Dominio

Un modello di dominio è il modello più importante dell'OOA dato che:

- Descrive le classi concettuali e le relazioni tra esse
- Fonte di ispirazione per classi di progetto e di implementazione
- Sviluppato in modo iterativo ed incrementale
- Limitato dai requisiti dell'iterazione corrente

Si tratta di una rappresentazione visuale di classi concettuali o di oggetti del mondo reale, nonché delle relazioni tra di essi, in un dominio di interesse. Applicando UML, un modello di dominio può essere realizzato come uno o più diagrammi delle classi in cui non sono definite operazioni e mostrano:

- Classi concettuali o oggetti di dominio
- Associazioni tra classi concettuali
- Attributi di classi concettuali

Il modello di dominio non è una raffigurazione di oggetti software, si tratta di un dizionario visuale delle astrazioni significative, della terminologia del dominio e del contenuto informativo del dominio di interesse.

Un modello dei dati mostra dati che devono essere memorizzati in modo persistente.

Un modello di dominio descrive informazioni che devono essere gestite nel sistema in discussione e può contenere

- Classi concettuali senza attributi
- Classi concettuali che hanno un ruolo puramente comportamentale e non un ruolo informativo

Classi Concettuali

Una classe concettuale è un'idea, una cosa o un oggetto che può essere considerata in termini di:

- Simbolo - Un parola o immagine usata per rappresentare la classe concettuale
- Intenzione - La definizione della classe concettuale in linguaggio naturale

- Estensione - L'insieme degli oggetti descritti dalla classe concettuale

In UML, una classe è il descrittore per un insieme di oggetti che possiedono le stesse caratteristiche (attributi, operazioni, metodi, relazioni e comportamento).

Associazioni In UML, un'associazione è la relazione tra due o più classificatori che comporta connessioni tra le rispettive istanze.

Attributi In UML, un attributo è la descrizione di una proprietà di una classe. Creare un modello di dominio è molto utile per comprendere il dominio del sistema da realizzare e il suo vocabolario (analisi), è inoltre una fonte di ispirazione per lo strato del dominio.

5.3 Come creare un modello di dominio

- Trovare le classi concettuali
- Disegnarle come classi in un diagramma delle classi UML
- Aggiungere associazioni e attributi

Identificare le classi concettuali

- Riusare o modificare dei modelli esistenti
- Utilizzare un elenco di categorie comuni
- Identificare nomi e locuzioni nominali

Associazione Un'associazione è una relazione tra classi (più precisamente, tra istanze di queste classi) che indica una connessione significativa e interessante. In UML, una relazione semantica tra due o più classificatori che coinvolge connessioni tra le loro istanze.

Quando mostrare un'associazione Considerare l'inclusione delle seguenti associazioni in un modello di dominio

- Associazioni per cui la conoscenza della relazione deve essere conservata per qualche durata
- Associazioni identificate mediante l'elenco di associazioni comuni

Evitare di inserire troppe associazioni in un modello di dominio. Considera che le associazioni del modello di progetto andranno implementate nel software, ma considera che un modello di dominio non è un modello di progetto, infatti una associazione descrive una relazione significativa tra oggetti del mondo reale, non descrive caratteristiche di oggetti software.

Molteplicità

La molteplicità di un ruolo indica quante istanze di una classe possono essere associate a una istanze dell'altra classe. Essa dipende dal contesto

Aggregazione e composizione

Aggregazione Una associazione che rappresenta una relazione intero-parte

Composizione (aggregazione composta) Si tratta di una forma forte di aggregazione in cui:

- Una parte appartiene a un composto alla volta
- Ciascuna parte appartiene sempre a un composto
- Il composto è responsabile della creazione e cancellazione delle sue parti

Per esempio un computer e una stampante sono una aggregazione, dato che sono debolmente collegati. Mentre un albero e le sue foglie sono una composizione, dato che sono fortemente collegati.

- L'aggregato può in alcuni casi esistere indipendentemente dalle parti, ma in altri casi no
- Le parti possono esistere indipendentemente dall'aggregato
- L'aggregato è in qualche modo incompleto se mancano alcune delle sue parti
- É possibile che più aggregati condividano una stessa parte

Per la composizione

- Ogni parte può appartenere a un solo composto per volta
- Il composto è l'unico responsabile di tutte le sue parti: questo vuol dire che è responsabile della loro creazione e distruzione

- Il composto può rilasciare una sua parte, a patto che un altro oggetto si prenda la relativa responsabilità
- Se il composto viene distrutto, deve distruggere tutte le sue parti o cederne la responsabilità a qualche altro oggetto
- La composizione è transitiva e asimmetrica

Come identificare la composizione

Si consideri di mostrare una composizione quando:

- C'è un ovvio gruppo fisico o logico intero-parte
- La vita della parte è limitata dalla vita del composto
- Alcune proprietà dell'intero si propagano anche alle parti (es. cancellazione/creazione)

5.4 Attributi

Si tratta di una proprietà elementare degli oggetti di una classe a cui viene associato un valore.

Tipi di attributi appropriati nel modello di dominio

Gli attributi devono avere tipo semplice, elementare, corrispondente a un tipo di dati primitivo. Non devono avere tipo corrispondente a un concetto complesso del dominio.

Il tipo degli attributi deve essere un tipo di dato e in UML con tipo di dato si intende un insieme di valori in cui l'identità univoca non è significativa.

Esempio Numero di telefono è un valore, Persona non è un tipo di dato.

5.5 Considerazioni finali

Esiste un modello di dominio univocamente corretto? Non esiste un solo modello di dominio corretto, essendo tutti i modelli approssimazioni del dominio che si sta tentando di capire. Meglio chiedersi se il modello di dominio è utile.

É meglio evitare un grosso sfrozo in valutazione secondo l'approccio a cascata

per creare un modello di dominio completo e corretto. Non utilizzare più di alcune ore per ogni iterazione.

Capitolo 6

Diagrammi di Sequenza di Sistema (SSD)

Un diagramma di sequenza di sistema (SSD) mostra gli eventi di input e output dei sistemi in discussione. Per un particolare corso di eventi all'interno di un caso d'uso mostra:

- Gli attori esterni che interagiscono direttamente con il sistema
- Il sistema (a scatola nera)
- Gli eventi di sistema generati dagli attori

6.1 Che cosa sono gli eventi e le operazioni di sistema

Casi d'uso descrivono il modo in cui gli attori esterni interagiscono con il sistema. Un attore genera degli **eventi di sistema** per richiedere l'esecuzione di alcune operazioni di sistema. Le operazioni di sistema sono operazioni che il sistema deve definire per gestire gli eventi di sistema.

In UML un evento è qualcosa di importante o degno di nota che avviene durante l'esecuzione di un sistema. Un evento di sistema è un evento esterno al sistema, di input, di solito generato da un attore per interagire con il sistema. In UML un'operazione rappresenta una trasformazione oppure un'interrogazione che un oggetto o componente può essere chiamato eseguire.

Un'operazione di sistema è una trasformazione oppure un'interrogazione che il sistema può essere chiamato a eseguire.

Fondamentale per riassumere le interazioni tra attori e le operazioni iniziate da essi Disegnare un SSD per uno scenario principale di successo di ciascun caso chiuso, nonché per gli scenari alternativi più frequenti o complessi.

Applicazione UML; Eventi, operazioni e SSD

UML non definisce né eventi di sistema o operazioni di sistema né diagrammi di sequenza di sistema. UML definisce degli elementi o diagrammi chiamati semplicemente evento, operazione e diagrammi di sequenza. Sono considerati a scatola nera.

Perché disegnare un SSD

Perché gli eventi di sistema sono importanti:

- Il software deve essere progettato per gestire questi eventi
- Un sistema software reagisce a tre cose: Eventi esterni, temporali o guasti (eccezioni)

Risulta quindi fondamentale descrivere il comportamento in funzione di queste operazioni tramite

- I casi d'uso
- Diagrammi di sequenza di sistema
- Contratti delle operazioni di sistema

Servono proprio a spiegare **Che cosa fa** il sistema, senza spiegare come lo fa. L'SSD è fortemente in relazione con i casi d'uso.

Assegnare il nome a eventi e operazioni di sistema

Eventi di sistema e operazioni corrispondenti devono essere espressi a un livello astratto di **Intenzione** anziché di azione.

Iniziare il nome con un verbo.

Esempio addItem, enterItem, make... Vedremo che l'SSD coinvolgerà più sistemi (anche esterni).

6.2 SSD e Glossario

Dato che utilizzeremo termini concisi per rappresentare le operazioni e gli eventi sarà fondamentale aggiungere questi termini al Glossario e dettagliarne il significato.

6.3 Lo scopo di un SSD

Lo scopo è comprendere e non è adatto a tutti gli scenari. È adatto solo per quelli dell'iterazione corrente.

Gli SSD fanno parte del Modello dei casi d'uso e sono maggiormente creati durante l'elaborazione. Non sono un elaborato ufficiale di UP, anche se i creatori di Up ne riconoscono l'utilità.

Capitolo 7

Contratti delle operazioni di sistema

I contratti delle operazioni di sistema sono un modo per specificare il comportamento delle operazioni di un sistema. I contratti sono scritti in un linguaggio formale e preciso e contengono informazioni sulla preconditione, la postcondizione e l'effetto dell'operazione sul sistema. I contratti servono per descrivere il comportamento del sistema in maniera più dettagliata.

7.1 Le sezioni di un contratto

- Operazione - Nome e parametri (firma) dell'operazione
- Riferimenti - Casi d'uso in cui può verificarsi questa operazione
- Pre-condizioni - Ipotesi significative sullo stato del sistema o degli oggetti nel modello di dominio prima dell'esecuzione dell'operazione. Si tratta di ipotesi non banali che dovrebbero essere comunicate al lettore
- Post-condizioni - É la sezione più importante. Descrive i cambiamenti di stato degli oggetti nel modello di dominio dopo il completamento dell'operazione

7.2 Che cos'è un'operazione di sistema

Si tratta di un'operazione pubblica del sistema la cui esecuzione è richiesta da un evento di sistema. L'esecuzione di una operazione di sistema cambia lo stato del sistema stesso. L'interfaccia del sistema è l'insieme di tutte le operazioni di sistema.

Un contratto è relativo a una operazione di sistema e descrive il cambiamento dello stato di oggetti del modello del dominio causato dall'esecuzione dell'operazione di sistema.

7.3 Post-condizioni

- Descrivono i cambiamenti nello stato degli oggetti del Modello di dominio.
- Non descrivono azioni eseguite durante l'esecuzione

Sono dichiarazioni circa lo stato degli oggetti dopo l'esecuzione dell'operazione e sono scritte al passato.

7.3.1 Possibili cambiamenti di stato

- Creazione o cancellazione di oggetto (istanza di una classe)
- Cambiamento di valore di attributo
- Formazione o rottura di collegamento (istanza di un'associazione)

Come già detto in precedenza sono super importanti dato che descrivono i cambiamenti richiesti dall'esecuzione dell'operazione di sistema, senza però descrivere come ottenere questi cambiamenti. **Le post condizioni vanno espresse con un verbo al passato.**

Il cambiamento dello stato del sistema causato dall'esecuzione dell'operazione di sistema va espresso in termini di oggetti, collegamenti e attributi del dominio di interesse che il sistema ha necessità di conoscere e ricordare.

7.3.2 Aggiornamento del modello di dominio

Durante la creazione dei contratti è normale che emerga la necessità di registrare nuove classi concettuali, attributi o associazioni nel modello di dominio.

Ricorda Nei metodi iterativi e evolutivi, tutti gli elaborati dell'analisi e della progettazione sono considerati parziali e imperfetti, ed evolvono in risposta a nuove scoperte.

7.4 Utilità e scrittura contratti

Non sempre è necessario scrivere dei contratti, a volte i casi d'uso sono sufficienti. I contratti sono adatti alla rappresentazione di situazioni dettagliate o complesse, che non è opportuno descrivere nei casi d'uso. I contratti sono appunto complementari ai casi d'uso.

Operazioni per la scrittura

1. Identificare le operazioni di sistema dagli SSD
2. Creare i contratti per le operazioni più complesse, o che non sono chiare dai casi d'uso
3. Per descrivere le post-condizioni utilizzare le seguenti categorie
 - Creazione o cancellazione di oggetto
 - Modifica di attributo
 - Formazione o rottura di collegamento
4. Le post-condizioni descrivono cambiamenti di stato al passato
5. Ricordare di formare collegamenti necessari tra oggetti esistenti e quelli appena creati
6. Le pre-condizioni sono normalmente implicate dall'esecuzione di operazioni di sistema precedenti