



02. Unified Process

Sviluppo di Applicazioni Software

Ferruccio Damiani

a.a. 2023/24

Università degli Studi di Torino - Dipartimento di Informatica

Attenzione!



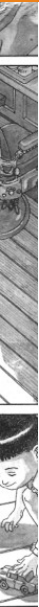
©2024 Copyright for this slides by Ferruccio Damiani. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<https://creativecommons.org/licenses/by/4.0/>

Si noti che

questi lucidi sono basati sul libro di testo del corso “C. Larman, *Applicare UML e i Pattern*, Pearson, 2016” e parzialmente sul materiale fornito da Matteo Baldoni, Viviana Bono, Claudia Picardi e Gianluca Torta dell'Università degli Studi di Torino.

Table of contents

- 
1. Object-Oriented Analysis/Design
 2. Unified Modeling Language
 3. Introduzione allo Unified Process
 4. Requisiti evolutivi

Lo sviluppo iterativo è alla base del modo in cui l'analisi dei requisiti orientata agli oggetti (**OOA**) e la progettazione orientata agli oggetti (**OOD**) viene applicata al meglio.

Per studiare l'OOA/D utilizzeremo **Unified Process**, un **processo iterativo** per lo sviluppo del software orientato agli oggetti.

UP è flessibile e può essere applicato usando un **approccio agile** come **Extreme Programming (XP)** e **Scrum**.

UP non è uno standard, è un processo di sviluppo che utilizza **UML**. La versione commerciale (IBM-Rational) è *RUP* (Rational Unified Process).



Object-Oriented Analysis/Design

Introduzione all'Object-Oriented Analysis/Design (OOA/D) con l'applicazione dell'Unified Modeling Language (UML) e dei **pattern**.

UML non è OOA/D: UML come strumento per **pensare** e **comunicare**.

OOD, progettazione guidata dalle **responsabilità**:

- Quali sono gli oggetti? Quali sono le classi?
- Cosa deve conoscere ciascun oggetto? Cosa deve saper fare?
- Come collaborano gli oggetti?

Euristiche, *best practice*, **pattern** aiutano in questo codificando principi di progettazione esemplari, coppie problema-soluzione.

Introduzione all'Object-Oriented Analysis/Design (OOA/D) con l'applicazione dell'Unified Modeling Language (UML) e dei **pattern**.

UML non è OOA/D: UML come strumento per **pensare** e **comunicare**.

OOD, progettazione guidata dalle **responsabilità**:

- Quali sono gli oggetti? Quali sono le classi?
- Cosa deve conoscere ciascun oggetto? Cosa deve saper fare?
- Come collaborano gli oggetti?

Euristiche, *best practice*, **pattern** aiutano in questo codificando principi di progettazione esemplari, coppie problema-soluzione.

La progettazione orientata agli oggetti

pone l'enfasi sulla definizione di oggetti software e del modo in cui questi collaborano per soddisfare i requisiti

L'OOD è fortemente correlata all'attività dell'**analisi dei requisiti**:

- **Casi d'uso**
- **Storie utente**

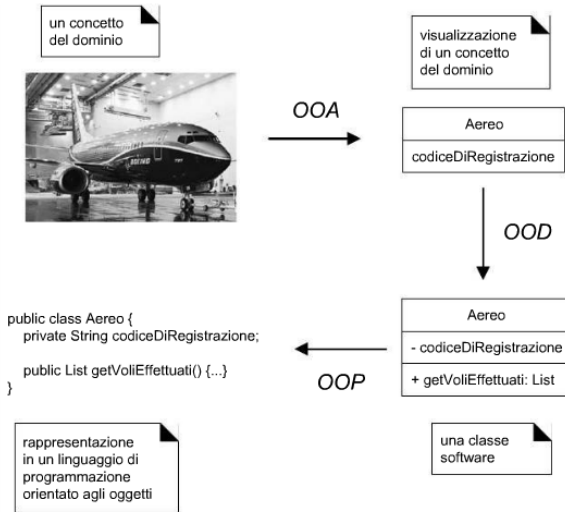
L'OOD è fortemente correlata all'attività dell'**analisi dei requisiti**:

- **Casi d'uso**
- **Storie utente**

L'analisi orientata agli oggetti

pone l'enfasi sull'identificazione e la descrizione degli oggetti, ovvero dei concetti nel dominio del problema

OOA, OOD e OOP



Un esempio: il gioco dei dadi

1. Definizione dei casi d'uso: storie scritte

Gioca una partita a dadi: il Giocatore chiede di lanciare i dadi. Il Sistema presenta il risultato: se il valore totale delle facce dei dadi è sette, il giocatore ha vinto; altrimenti ha perso.

Un esempio: il gioco dei dadi

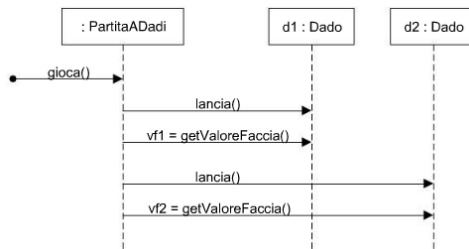
1. Definizione dei casi d'uso: storie scritte
2. Definizione di un modello di dominio: i concetti o gli oggetti significativi del dominio



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Un esempio: il gioco dei dadi

1. Definizione dei casi d'uso: storie scritte
2. Definizione di un modello di dominio: i concetti o gli oggetti significativi del dominio
3. Assegnare responsabilità agli oggetti e disegnare diagrammi di interazione: responsabilità e collaborazioni



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Un esempio: il gioco dei dadi

1. Definizione dei casi d'uso: storie scritte
2. Definizione di un modello di dominio: i concetti o gli oggetti significativi del dominio
3. Assegnare responsabilità agli oggetti e disegnare diagrammi di interazione: responsabilità e collaborazioni
4. Definizione dei diagrammi delle classi di progetto



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

L'analisi dei requisiti e l'OOA/D vanno svolte nel contesto di qualche processo di sviluppo

- Processo di sviluppo iterativo
- Approccio agile
- Unified Process (UP)



Unified Modeling Language

Unified Modeling Language (UML)

è un linguaggio **visuale** per la specifica, la costruzione e la documentazione degli elaborati di un sistema software.

UML è uno standard *de facto* per la notazione di diagrammi per disegnare o rappresentare figure relative al software, e in particolare al software OO.

L'utilizzo di UML può essere come “abbozzo” (diagrammi informali per esplorare e comunicare), come “progetto” o come “linguaggio di programmazione”.

La modellazione agile enfatizza l'uso di UML come “abbozzo”.

Unified Modeling Language (UML)

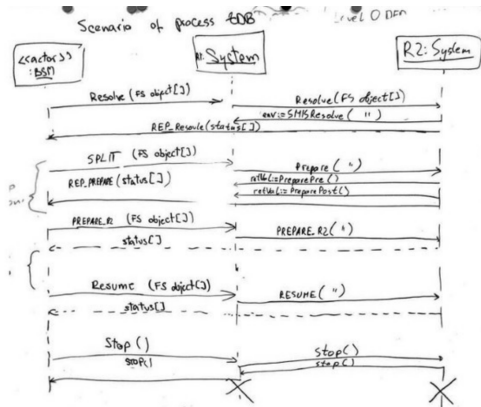
è un linguaggio **visuale** per la specifica, la costruzione e la documentazione degli elaborati di un sistema software.

UML è uno standard *de facto* per la notazione di diagrammi per disegnare o rappresentare figure relative al software, e in particolare al software OO.

L'utilizzo di UML può essere:

- come “abbozzo” (diagrammi informali per esplorare e comunicare),
- come “progetto“, o
- come “linguaggio di programmazione”.

La modellazione agile **enfatizza l'uso di UML come “abbozzo”**.



- **Punto di vista concettuale** (modello di dominio): la notazione dei diagrammi di UML è utilizzata per visualizzare concetti del modo reale (classe concettuale)



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

- **Punto di vista concettuale** (modello di dominio): la notazione dei diagrammi di UML è utilizzata per visualizzare concetti del modo reale (classe concettuale)
- **Punto di vista software** (diagramma delle classi di progetto): la notazione dei diagrammi delle classi di UML è utilizzata per visualizzare elementi software (classe software)



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Anni Sessanta e Settanta: nascita dei linguaggi OO (Simula, Smalltalk).

Nel 1988 **Bertrand Meyer** pubblica "Object-Oriented Software Construction".

Nel 1991 **Jim Rumbaugh** pubblica "Object-Oriented Modeling and Design" (OOA/D).

Nel 1991 **Grady Booch** pubblica "Object-Oriented Software Engineering" (OOA/D e Casi d'uso per i requisiti)

Nel 1994 Booch e Rumbaugh fanno nascere dalle rispettive proposte UML.

Rational Corporation: **Ivar Jacobson, Grady Booch e Jim Rumbaugh** (i "tre amigos") si concentrano sulla notazione piuttosto che su di un metodo comune.

L'**Object Management Group** (OMG), un consorzio per la standardizzazione industriale dei cocetti relativi all'OO, avvia la standardizzazione di **UML**.

Nel 1997 UML 1, nel 2004 UML 2.



Introduzione allo Unified Process

Unified Process (UP)

Unified Process (UP) - Booch, Rumbaugh, Jacobson

è un processo *iterativo* e *evolutivo* (incrementale) per lo sviluppo del software per la costruzione di sistemi orientati agli oggetti. Le iterazioni iniziali sono guidate dal **rischio**, dal **cliente** e dall'**architettura**.

Unified Process (UP)

UP incoraggia l'uso di pratiche agili introdotte da altre metodologie:

- Iterazioni corte e timeboxed
- Raffinamento di piani, requisiti, progettazione
- Gruppi di lavoro auto-organizzati che si coordinano in riunioni regolari (da Scrum)
- Programmazione a coppie e sviluppo guidato dai test (da eXtremeProgramming)
- Modellazione agile: l'obiettivo è la comprensione del software piuttosto che la documentazione dello stesso

Unified Process (UP)

Cosa c'è in UP:

- Un'organizzazione del piano di progetto per fasi sequenziali
- Indicazioni sulle attività da svolgere nell'ambito di discipline e sulle loro inter-relazioni
- Un insieme di ruoli predefiniti
- Un insieme di artefatti da produrre

Ruoli \Rightarrow Attività \Rightarrow Artefatti

Un progetto UP organizza il lavoro e le iterazioni in **quattro fasi**:

- **Ideazione** (*inception*): visione approssimativa, studio economico, portata, stime approssimative dei costi e dei tempi
Milestone: **obiettivi**
- **Elaborazione** (*elaboration*): Visione raffinata, implementazione iterativa del nucleo dell'architettura, risoluzione dei rischi maggiori, identificazione della maggior parte dei requisiti e della portata, stime più realistiche sulle loro inter-relazioni
Milestone: **architetturale**
- **Costruzione** (*construction*): Implementazione iterativa degli elementi rimanenti, più facili e a rischio minore, preparazione al rilascio
Milestone: **capacità operativa**
- **Transizione** (*transition*): Beta test, rilascio
Milestone: **rilascio prodotto**

Organizzazione temporale dei progetti in UP



Attenzione!

- **Ideazione:** non è una fase di requisiti bensì di fattibilità, in cui viene eseguita un'indagine sufficiente per decidere di proseguire con il progetto o di interromperlo
- **Elaborazione:** non è una fase di requisiti o di progettazione bensì una fase in cui viene implementata in modo iterativo l'architettura del sistema e vengono mitigati i rischi maggiori

Le attività lavorative in UP si eseguono nell'ambito di **discipline** (*Core Workflow*).

Disciplina

Una disciplina è un insieme di attività e dei relativi elaborati in una determinata area, come le attività relative all'analisi dei requisiti.

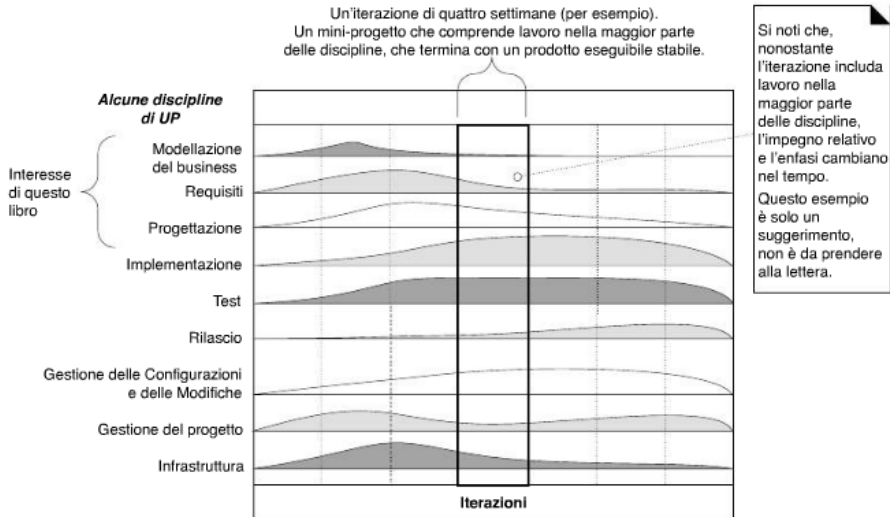
Elaborato

Elaborato (artefatti o *work product*, *product* in RUP) è il termine generico che indica un qualsiasi prodotto di lavoro: codice, schemi di base di dati, documenti di testo, diagrammi, modelli, ecc.

- **Modellazione del business.** Attività che modellano il dominio del problema ed il suo ambito.
- **Requisiti.** Attività di raccolta dei requisiti del sistema.
- **Progettazione** (*analysis and design*). Attività di analisi dei requisiti e progetto architeturale del sistema.
- **Implementazione.** Attività di progetto dettagliato e codifica del sistema, test su componenti.
- **Test.** Attività di controllo di qualità, test di integrazione e di sistema.
- **Rilascio.** Attività di consegna e messa in opera.

- **Gestione delle configurazioni e del cambiamento.** Attività di manutenzione durante il progetto.
- **Gestione progetto.** Attività di pianificazione e governo del progetto.
- **Infrastruttura** (*environment*). Attività che supportano il team di progetto, riguardo ai processi e strumenti utilizzati.

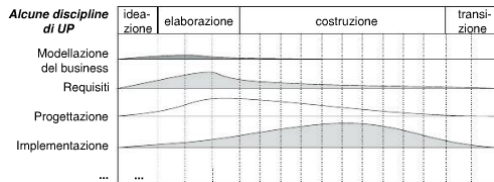
Le discipline di UP



Relazione tra discipline e fasi

Attenzione!

- Le **fasi** sono **sequenziali** e la fine di una fase corrisponde ad una *milestone*
- Le **discipline** (tipologie di attività) **non sono sequenziali** e si eseguono nel progetto in ogni iterazione
- Il numero di iterazioni dipende dalla decisione del manager di progetto e dai rischi del progetto



L'impegno relativo nelle discipline cambia a seconda delle fasi.

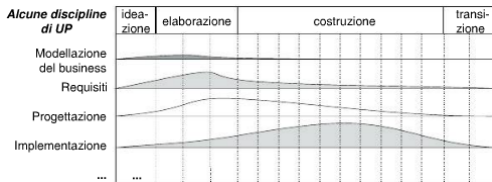
Questo esempio è solo un suggerimento, non è da prendere alla lettera.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Relazione tra discipline e fasi

Attenzione!

- Le iterazioni iniziali tendono in modo naturale a dare una maggiore enfasi relativa sui requisiti e sulla progettazione, mentre quelle successive lo faranno in misura minore
- Questo perché i requisiti e il progetto si stabilizzano attraverso un processo di feedback e adattamento



L'impegno relativo nelle discipline cambia a seconda delle fasi.

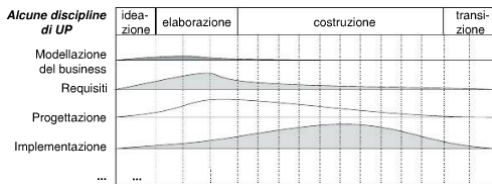
Questo esempio è solo un suggerimento, non è da prendere alla lettera.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Relazione tra discipline e fasi

Attenzione!

- Durante l'elaborazione le iterazioni tendono ad avere un livello relativamente alto di lavoro sui requisiti e la progettazione, sebbene prevedano anche un certo livello di implementazione
- Durante la costruzione, l'enfasi è maggiore sull'implementazione e minore sull'analisi dei requisiti



L'impegno relativo nelle discipline cambia a seconda delle fasi.

Questo esempio è solo un suggerimento, non è da prendere alla lettera.

©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

- UP usa solo UML come linguaggio di modellazione (ad esempio, non si usano i Data Flow Diagram)
- I diagrammi UML si usano con variabilità: se un diagramma non è necessario non si usa, però tale scelta si indica esplicitamente. Bisogna **personalizzare** UP
- I diagrammi si usano in UP seguendo le caratteristiche di **iterazione** ed **incremento** (incrementi definiti su uno stesso diagramma)
- UP dice **quando** usare un diagramma

- In UP quasi tutto (tra artefatti e pratiche) è **opzionale**, eccetto che lo sviluppo iterativo e guidato dal rischio, la verifica continua della qualità e naturalmente il codice
- La scelta delle pratiche e artefatti UP per un progetto si riassume in un documento chiamato **scenario di sviluppo** (artefatto della disciplina Infrastruttura)

Esempio di scenario di sviluppo

Tabella 2.1 Scenario di Sviluppo di esempio (i – inizio; r – raffinamento).

Disciplina	Pratica	Elaborato Iterazione →	Ideazione I1	Elaboraz. E1..En	Costr. C1..Cn	Transiz. T1..T2
Modellazione del business	modellazione agile workshop requisiti	Modello di Dominio		i		
Requisiti	workshop requisiti esercizio sulla visione votazione a punti	Modello dei Casi d'Uso	i	r		
		Visione	i	r		
		Specifica Supplementare	i	r		
		Glossario	i	r		
Progettazione	modellazione agile sviluppo guidato dai test	Modello di Progetto		i	r	
		Documento dell'Architettura Software		i		
		Modello dei Dati		i	r	
Implementazione	sviluppo guidato dai test programmazione a coppie integrazione continua standard di codifica	...				
Gestione del progetto	gestione del progetto agile riunioni Scrum giornaliere	...				
...						



Requisiti evolutivi

Che cosa sono i requisiti

Requisito

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

Sorgenti dei requisiti

I requisiti derivano da richieste degli utenti del sistema, per risolvere dei problemi e raggiungere degli obiettivi.

Tipi di requisiti

- **Requisiti funzionali.** I requisiti *comportamentali* descrivono il comportamento del sistema, in termini di *funzionalità* fornite ai suoi utenti.
- **Requisiti non funzionali.** Le proprietà del sistema nel suo complesso, come ad esempio *sicurezza*, *prestazioni* (tempo di risposta, throughput, uso di risorse), *scalabilità*, *usabilità* (fattori umani), ecc.

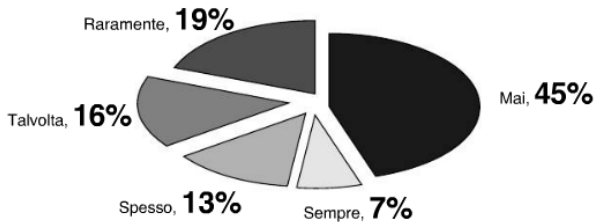
UP promuove un insieme di **best practice**, una delle quali è **gestire i requisiti**.

Nel contesto dei desiderata delle parti interessate, che sono poco chiari e che inevitabilmente cambiano, **un approccio sistematico per trovare, documentare, organizzare e tracciare i requisiti che cambiano di un sistema**.

In UP si iniziano programmazione e test quando è stato specificato solo il 10% o il 20% dei requisiti più significativi dal punto di vista del valore di business, del rischio e dell'architettura.

Requisiti in UP vs Requisiti 'a cascata'

Secondo uno studio, il 45% di queste caratteristiche specificate con un approccio a cascata non è mai stato utilizzato, e un altro 19% è stato raramente utilizzato.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Requisiti in UP vs Requisiti 'a cascata'

UP incoraggia un'acquisizione (un approccio sistematico per trovare) dei requisiti abile, attraverso tecniche quali:

- scrivere i casi d'uso con i clienti
- workshop dei requisiti a cui partecipano sia sviluppatori che clienti
- gruppi di lavoro con rappresentanti dei clienti
- dimostrazione ai clienti dei risultati di ciascuna iterazione, per sollecitare il feedback

Modello FURPS+, acronimo per:

- **Funzionale (F)**: requisiti funzionali e sicurezza
- **Usabilità (U)**: facilità d'uso del sistema, documentazione e aiuto per l'utente
- **Affidabilità (R - *reliability*)**: la disponibilità del sistema, la capacità di tollerare guasti o di essere ripristinato a seguito di fallimenti
- **Prestazioni (P)**: tempi di risposta, throughput, capacità e uso delle risorse
- **Sostenibilità (S)**: facilità di modifica per riparazioni e miglioramenti, adattabilità, manutenibilità, verificabilità, localizzazione, configurazione, compatibilità
- **altre (+)**: vincoli di progetto (risorse, hardware, ecc.), interoperabilità, operazionali, fisici, legali, ecc.

UP ha diversi elaborati (molti opzionali):

- **Modello dei Casi d'Uso:** scenari tipici dell'utilizzo di un sistema (requisiti funzionali, comportamento)
- **Specifiche Supplementari:** ciò che non rientra nei casi d'uso, requisiti non funzionali o funzionali non esprimibili attraverso casi d'uso (es. generazione di un report)
- **Glossario:** termini significativi, dizionario dei dati (requisiti relativi ai dati, regole di validazione, valori accettabili)
- **Visione:** riassume i requisiti ad alto livello, un documento sintetico per apprendere rapidamente le idee principali del progetto
- **Regole di Business:** regole di dominio, i requisiti o le politiche che trascendono un unico progetto software e a cui il sistema deve conformarsi (es. leggi fiscali dello stato)

Non si è capito lo sviluppo iterativo UP se

- Si cerca di definire tutti i requisiti del software prima di iniziare la progettazione o l'implementazione
- Si dedicano giorni o settimane a modellare con UML prima di iniziare a programmare
- Si pensa: ideazione = requisiti, elaborazione = progettazione, costruzione = implementazione (cioè, si adotta l'approccio 'a cascata')
- Si pensa che l'obiettivo dell'elaborazione sia quello di definire in maniera completa e dettagliata i modelli, che verranno tradotti in codice durante la costruzione
- Si pensa che la durata adeguata per una iterazione siano 3 mesi al posto di 3 settimane
- Si cerca di pianificare il progetto nei dettagli dall'inizio fino alla fine, e di prevedere in maniera speculativa tutte le iterazioni e cosa deve accadere in ognuna di esse