Politecnico di Torino

Ingegneria del Software

Anno Accademico 2022/2023

Marco Lampis

28 febbraio 2023

Indice

0 Informazioni

La seguente dispensa è stata realizzata nell'anno accademico 2022-2023 durante il corso di *Ingegneria del Software*. Il materiale **non** è ufficiale e non è revisionato da alcun docente, motivo per cui non mi assumo responsabilità per eventuali errori o imprecisioni.

Per qualsiasi suggerimento o correzione non esitate a contattarmi o a eseguire una pull request su GitHub.

E' possibile riutilizzare il materiale con le seguenti limitazioni:

- Utilizzo non commerciale
- Citazione dell'autore
- Riferimento all'opera originale

E' per tanto possibile:

- Modificare parzialmente o interamente il contenuto

Questi appunti sono disponibili su GitHub al seguente link:

```
https://github.com/Guray00/polito_lectures
```

Figura 1: Repository GitHub

0.1 Contributi

La condivisione è alla base del successo di qualsiasi progetto, citando:

> *“Open source is about collaborating; not competing. ~ Kelsey Hightower”*

La seguente dispensa ha avuto il prezioso contributo di:

- Marco Lampis

1 Introduzione

Con **ingegneria del software** si fa riferimento a un processo che richiede un approccio sistematico diviso in più fasi come *sviluppo*, *operazione* (messa in funzione), *manutenzione* e infine *ritiro* (obsolescenza o rifacimento).

Un altra rappresentazione vede l'applicazione di *principi scientifici* in modo da trasformare un problema in una soluzione funzionante alla quale segue la manutenzione fino al termine del suo ciclo di vita.

Ogni software ha un proprio ciclo di vita che comincia quando il prodotto viene concepito e termina quando questo non è più disponibile per l'uso. Ciascuna fase prevede un set di input, un set di output e un numero di attività da svolgere.

Ogni prodotto di qualità richiede lo sviluppo attraverso un processo che trova le sue fondamenta in pratiche e principi dell'ingegneria del software. **La qualità del prodotto dipende dalla qualità del processo**.

Lo sviluppo può avvenire con più metodologie, alcuni esempi sono:

- **Waterfall**: incrementale o evolutivo
- **Unified**
- **Support activities**

A questi seguono alcune metodologie più recenti, che verranno analizzate prossimamente, come:

- **Agile development**
- **SCRUM**
- **Continuous integration**
- **Continuous delivery**

1.1 Waterfall

Lo sviluppo avviene attraverso più passaggi sequenziali:

1. **Definizione dei requisiti**: descrizione generale in termini business dei vincoli e delle funzionalità principali.
2. **Progettazione**: viene definita l'architettura e come gli elementi interagiscono tra loro con i collegamenti necessari e le decisioni relative al *make-or-buy*, ovvero se deve essere sviluppato internamente o delegato.
3. **Implementazione**: progetto di dettaglio, sviluppo e testing.
4. **Trasferimento**: invio del software al committente
5. **Operazione**

Una problematica è però relativa alla rigidità, in quanto i requisiti sono bloccati subito alla prima fase.

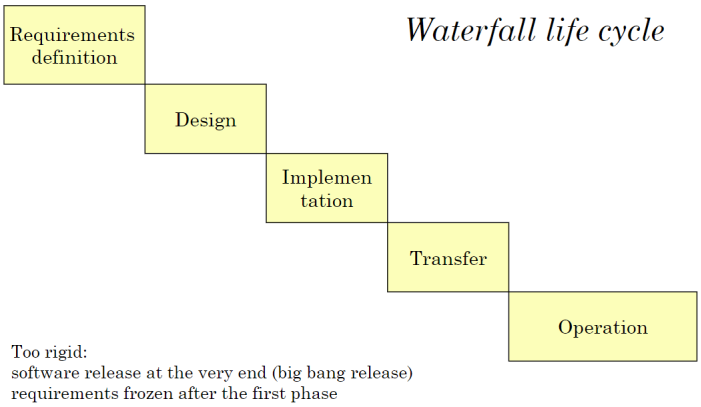


Figura 1.1: Waterfall

1.1.1 Incrementale

Una alternativa della metodologia waterfall è di tipo incrementale, dove l'implementazione e il rilascio sono dei "pezzetti" piuttosto che un unico passaggio.

Questo potrebbe causare un aumento dei costi.

1.1.2 Evolutivo

Le fasi vengono ripetute nel corso del tempo. E' utile quando i requisiti iniziali sono scarsi o non stabili. Permette agli utenti un funzionamento iniziale, a cui seguono ulteriori versioni come prototipi.

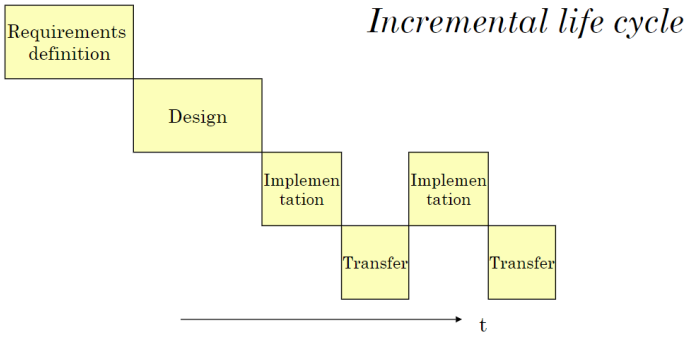


Figura 1.2: Ciclo di vita incrementale

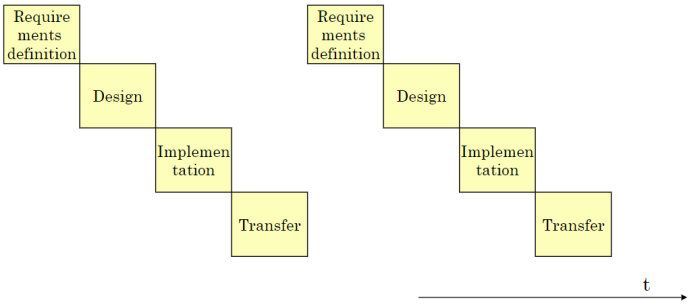


Figura 1.3: Ciclo di vita evolutivo

1.1.3 Prototyping

Il prototyping è una tecnica di sviluppo che consiste in una implementazione parziale del sistema in modo che i clienti, gli utenti e glli sviluppatori possano imparare di più riguardo i problemi e le relative soluzioni.

Gli approcci possono essere di due tipi:

- Usa e getta: può essere utilizzato per validare l'interfaccia utente, verificare se una particolare architettura soddisfa i requisiti e validare un particolare algoritmo.
- Evolutivo: Va avanti passo dopo passo fino a quando non sono soddisfatte tutte le richieste per il prodotto finale.

1.2 Unified Process

La strategia **Unified Process** combina gli approcci incrementali ed evolutivi dividendosi in 4 fasi:

- **Inception**: viene effettuato business modeling, ovvero si sviluppa mediante una modellazione dei requisiti tra persone di specializzazione differenti. Segue il risk assessment ovvero vedere se le scadenze possono essere rispettate, project schedule e cost estimate.
- **Elaboration**: analisi dettagliata dei requisiti, progettazione dell'architettura e validazione tramite prototipi oltre alla realizzazione del piano di sviluppo.
- **Construction**: implementazione supportata da diagrammi UML dettagliati.
- **Transition**: distribuzione.

Un tipo di attività non coincide con la fase specifica, anche se può essere quella più importante.

> Fa utilizzo del UML.

Il business modeling descrive il contesto in cui un sistema software deve operare e le relazioni con gli altri sottosistemi da un punto di vista business. Eì un ponte tra l'analisi business e l'analisi software.

1.3 Support activities

Le attività di supporto si dividono in:

- **Project management**: pianificazione, staffing, monitoring, controllo e leading.
- **Version control e configurazione della gestione**: definisce gli elementi del sistema, controllando le release e i cambiamenti.

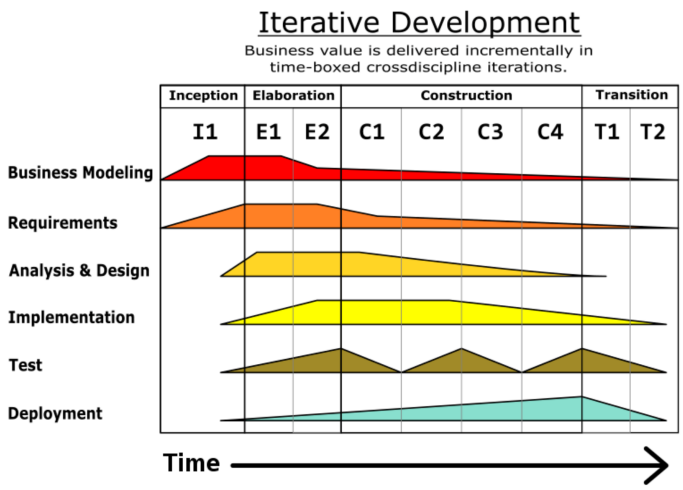


Figura 1.4: Unified process

- **Verification**: si preoccupa di controllare se il prodotto viene sviluppato correttamente. *Stiamo facendo il prodotto correttamente?*
- **Validation**: verifica che il prodotto soddisfi le richieste del cliente. *Stiamo facendo il prodotto giusto?*
- **Quality assurance**: verifica che gli standard e le procedure stabiliti vengano seguite.

1.4 Modelli Concettuali

Un modello è una rappresentazione astratta e rigorosa di un sistema reale, permettendo all'utente di considerare le proprietà importanti di un sistema.

Per lo sviluppo del software esiste il concetto del modello operazionale/operativo, in quanto il modello potrebbe essere direttamente mappato nel software, dunque un modello potrebbe essere direttamente il codice implementato oppure a partire dal modello può essere generato il codice *(model driven development)*.

1.4.1 UML

Con UML si fa riferimento al _Unified Modeling Language, realizzato per un alto numero di modelli di linguaggio. Non è una metodologia globale.

Uno standard è il OMG.

I diagrammi possono essere strutturali (class mdodels, object models) oppure comportamentali (casi d'uso,)

::note La differenza tra linguaggio e notazione è che il linguaggio è un insieme di regole sintattiche e semantica, mentre la notazione è un insieme di simboli che rappresentano un linguaggio. :::

1.4.2 Class Models

Il sistema da prende in considerazione è u formato da un insieme di oggetti i quali fanno riferimento a delle classi. Sono presenti dei link sistematici tra gli oggetti e rappresentati da link chiamati **relazioni** tra classi.

Le classi possono avere *proprietà* denominate **attributi**. Un class model mostra

Un object model è una realizzazione particolare di una class model in quanto mostra le relazioni tra gli oggetti per motivi illustrativi.

Alcuni esempi sono:

- Class model: tecnico
- Domain model: alto livello
- Information model: prospettiva dei dati

alcuni aspetti rilevanti sono:

- Relazioni: associative, composizione, inheritance, ricorsive
- Attributi normali e associativi
- Espressioni navigazionali
- Attributi necessari e relazioni necessarie
- Attributi derivati e relazioni derivate
- Invarianti e regole di validazione

Un esempio potrebbe essere il seguente:

> Nelle università sono presenti professori e studenti, con i primi che insegnano corsi e studenti che vi sono iscritti.

Un corso è insegnato da un solo professore, mentre gli studenti possono essere iscritti a più corsi. Inoltre, un professore può insegnare da 1 a 3 corsi, mentre i corsi sono sono tenuti da un solo docente. Gli studenti hanno da 10 a 50 corsi, ma ogni corso può avere n studenti.

Il modello è valido? In realtà no, perchè abbiamo indicato un minimo di 10 corsi per studente.

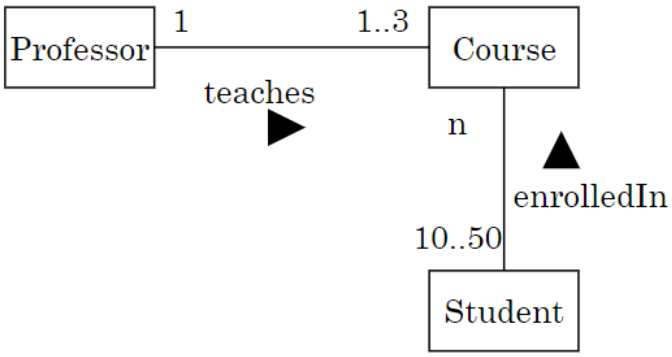


Figura 1.5: Esempio 1

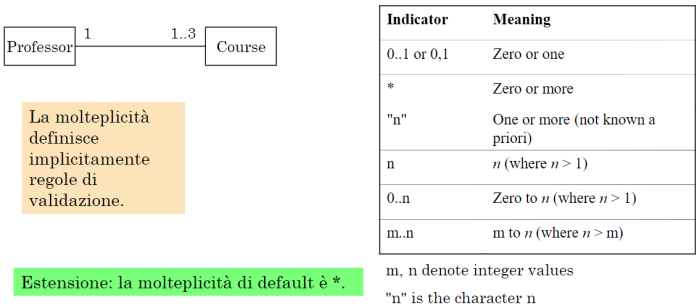


Figura 1.6: Molteplicità

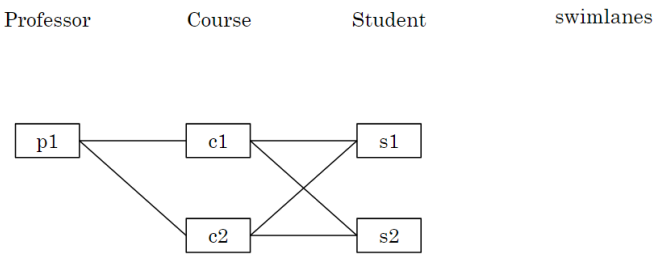
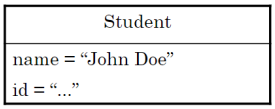


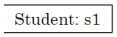
Figura 1.7: Object Model

Classes represent individuals, also called instances or objects.

An object model shows a number of objects along with their attributes and associations.



Simplified representation: the type is the header of a swimlane



Simplified representation: type and label.

The label may be the value of an attribute.

1.4.2.1 Composizione

La composizione viene indicata con un rombo pieno. Indica che un oggetto è composto da altri oggetti.

Quando un contenitore viene eliminato, tutti i suoi componenti vengono automaticamente rimossi a loro volta.

In questo esempio, una orderLine fa parte di un ordine.

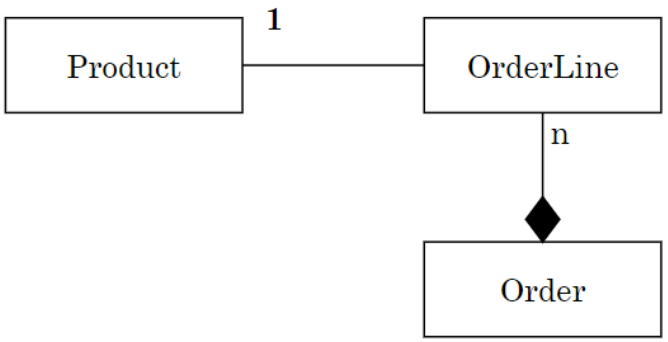


Figura 1.8: Composizione

Un oggetto può far parte di un solo contenitore per volta, ma può essere indicato come figlio di entrambi.

1.4.2.2 Aggregazione

Con il rombo vuoto si fa riferimento all'aggregazione. Indica che un oggetto è aggregato ad un altro oggetto.

Quando un aggregazione viene eliminata non vengono eliminati i suoi componenti.

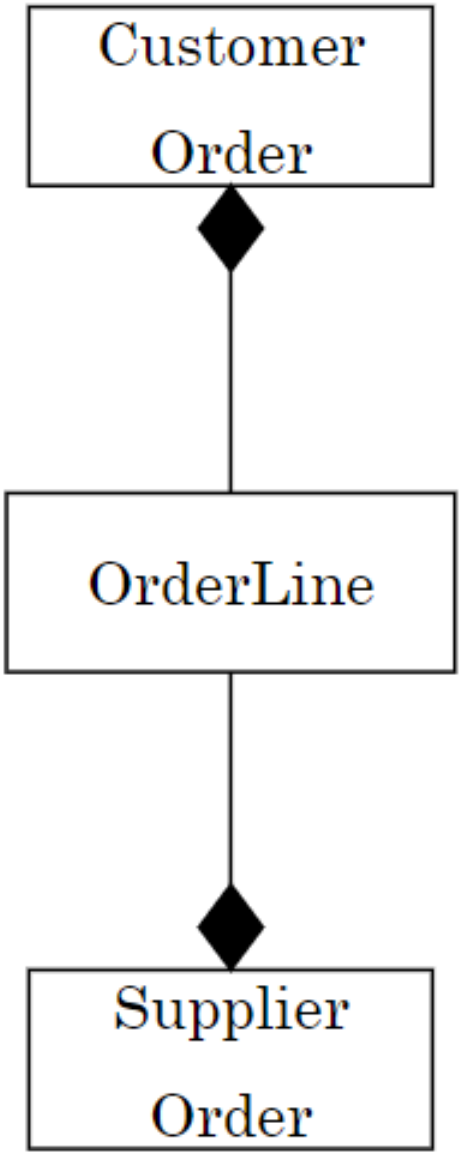


Figura 1.9: Composizione

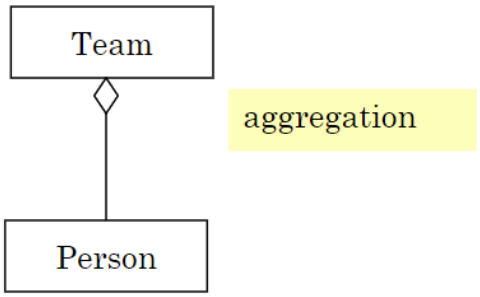
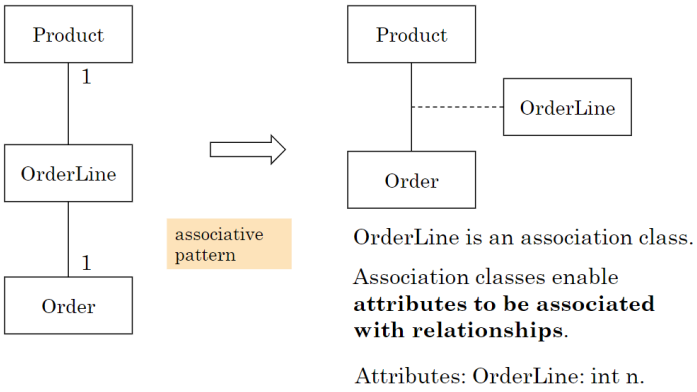


Figura 1.10: Aggregazione

1.4.2.3 Association class

Con classe associativa si fa riferimento a un collegamento…



1.4.2.4 Inheritance

L'ereditarietà viene indicata con una freccia a punta vuota. Indica che una classe è una specializzazione di un'altra classe.

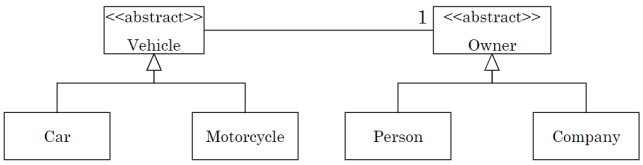
> Un veicolo ha un proprietario che può essere una persona o una compagnia e può essere una macchina o una moto.

Veicolo e Proprietario sono classi astratte, ovvero non possono essere inizializzate.

<<abstract>> is a *stereotype*

Class model

Inheritance



Object model