PROGRAMMAZIONE DEI CALCOLATORI ELETTRONICI

INTRODUZIONE ALL'INGEGNERIA DEL SOFTWARE

Roberto Nardone, Luigi Romano

INTRODUZIONE ALL'INGEGNERIA DEL SOFTWARE

- Introduzione al ciclo di vita del software
  - La notazione UML
  - Le fasi del ciclo di vita
  - I modelli del cliclo di vita
- Riferimenti Utili

CICLO DI VITA DEL SOFTWARE

- Quando si realizza un prodotto software, bisogna svolgere una serie di passi predefiniti, o meglio una strategia che permetta di ottenere il prodotto rispettando i tempi e mantenendo alta la qualità.
  - Questa serie di passi o attività rappresenta **il ciclo di vita del software**
  - In genere ogni attività viene descritta attraverso la notazione **UML**

DESCRIVERE IL CICLO DI VITA DEL SOFTWARE – LA NOTAZIONE UML

- L'Unified Modeling Language (UML) è un linguaggio di modellazione grafico che serve per descrivere e progettare sistemi software, in particolare quelli che hanno un paradigma object oriented (OO).
- Motivazione
  - I linguaggi di programmazione spesso hanno bisogno di ulteriori astrazioni per descrivere il loro funzionamento
- L'UML è uno standard gestito dalla Object Management Group (OMG)
- Sito ufficiale: https://www.uml.org/

COME USARE L'UML

- L'UML può essere visto come un linguaggio di programmazione **descrittivo**.
  - Al contrario di un linguaggio di programmazione **prescrittivo** in cui esiste un ente che dichiara cosa è legale o meno e quale significato dare alle espressioni.
  - In un linguaggio di tipo descrittivo le regole utilizzate si desumono guardando a come le persone utilizzano tale linguaggio per descrivere un sistema.
    - In questo senso l'UML somiglia più a un linguaggio parlato che non a un linguaggio di programmazione.
    - Aziende diverse adotteranno differenti convenzioni, in base alle loro esigenze di progettazione
    - Inoltre lo standard è così complesso che si offre a interpretazioni multiple
  - Lo scopo è quello di spiegare in modo esaustivo un processo software attingendo risorse dall'UML senza essere legati da particolari formalismi.

COME USARE L'UML

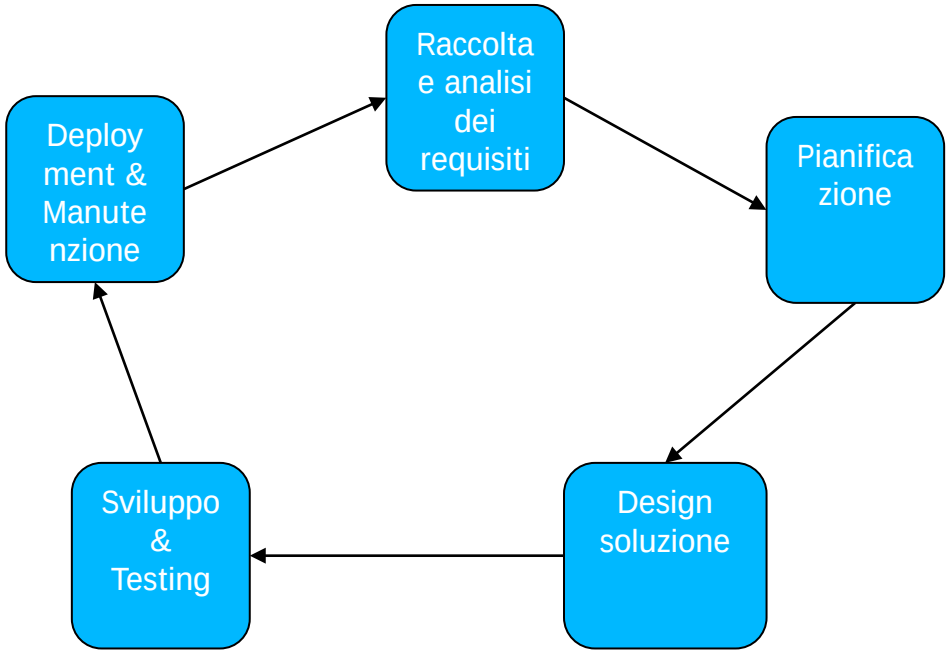
- L'UML si mantiene astratto rispetto al particolare linguaggio utilizzato per sviluppare
  - Osservando l'UML si può desumere grosso modo cosa fa il codice sorgente, senza necessariamente conoscere il tipo di linguaggio di programmazione utilizzato
  - Non esistono regole standard per la mappatura tra un modello UML e un qualsiasi linguaggio di programmazione.
- Spesso l'UML non è sufficiente per spiegare alcuni concetti
  - In tal caso si possono adottare costrutti aggiuntivi per descrivere un sistema software.

I DIAGRAMMI UML PIÙ COMUNI

- I seguenti rappresentano i diagrammi più comuni che andremo a rappresentare e a utilizzare per la descrizione delle attività di ingegneria del software
- Class diagram
- Sequence diagram
- Use case diagram
- Activity diagram

ATTIVITÀ DEL CICLO DI VITA DEL SOFTWARE

- Generalmente le attività del ciclo di vita del software sono le seguenti



RACCOLTA E ANALISI DEI REQUISITI

- I requisiti vengono raccolti e definiti tra gli attori coinvolti nel processo di creazione del software (stakeholder)
  - Clienti, utenti finali, manager del team di sviluppo, a volte il team di sviluppo stesso
- I requisiti possono essere di vario tipo:
  - **FR**- Funzionale – (riferito ai comportamenti del sistema in relazione a suoi input e a situazioni specifiche)
  - **NF**- Non Funzionale - (riferito ai vincoli sulla qualità del sistema come le performance, compatibilità, disponibilità…)
  - **SR** – Architetturale e di sistema - (requisiti a livello architetturale, ovvero come dovrebbe essere costruito il sistema al fine di venire incontro alle esigenze specificate in FR e NF)
- Per la descrizione di essi viene spesso utilizzata la notazione UML per i casi d'uso -> **Use Case diagram**

RACCOLTA DEI REQUISITI - USE CASE

- Un caso d'uso viene usato generalmente per descrivere i requisiti funzionali
- Esso è composto da un insieme di scenari e ciascuno di essi descrive come eseguire la stessa funzionalità ma in modo diverso (Es: acquistare un prodotto)
- Viene sottolineata l'interazione tra un attore e il sistema e tra il sistema e se stesso
- Di solito si comincia con lo scrivere lo scenario principale attraverso una lista numerata di passi da seguire per raggiungere l'obiettivo.
  - Un elemento della lista può essere esso stesso un caso d'uso incluso in quello principale
  - Gli altri scenari rappresentano variazioni o estensioni di quello principale

RACCOLTA DEI REQUISITI - USE CASE

- **Acquisto di un prodotto**
  - Scenario principale
    1. Il cliente naviga tra i prodotti e sceglie il prodotto da acquistare
    2. Il cliente va al check out
    3. Il cliente compila le informazioni per la spedizione
    4. Il sistema presenta il prezzo completo di spese di spedizione
    5. Il cliente compila le informazioni della carta di credito
    6. Il sistema autorizza il pagamento
    7. Il sistema conferma la vendita e invia la conferma all'email del cliente

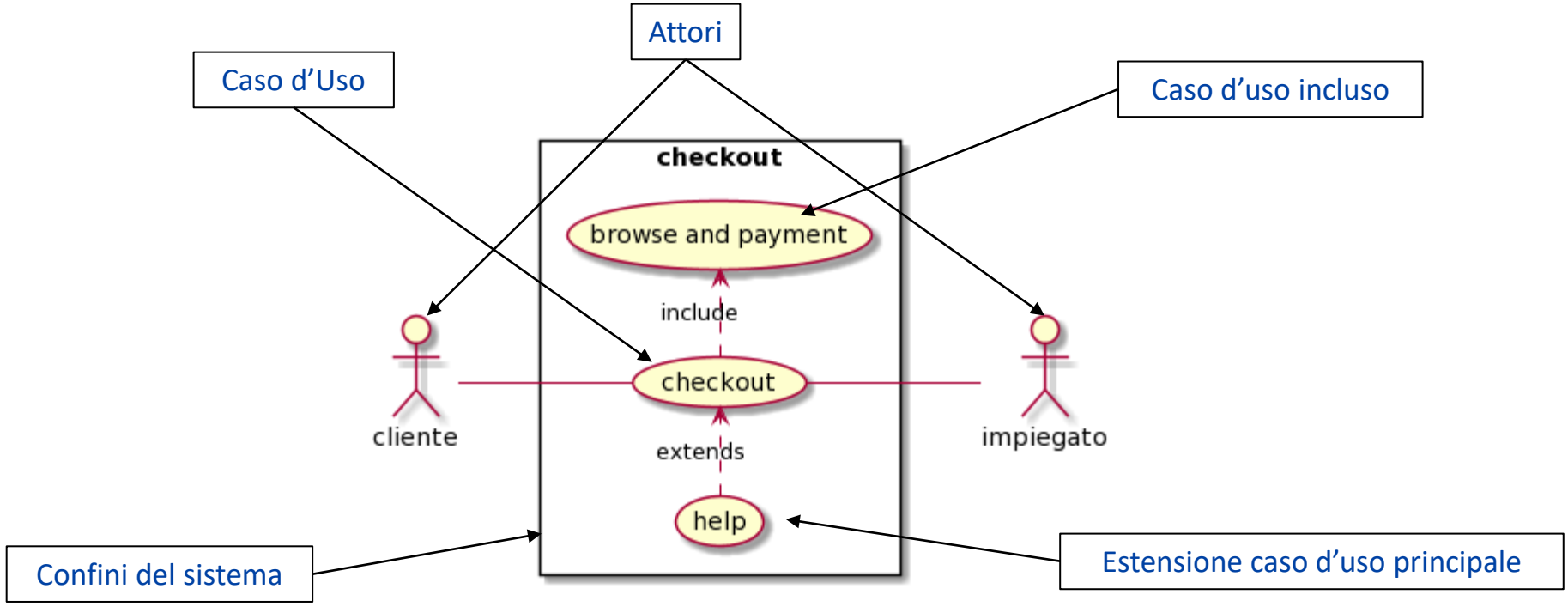
Caso d'uso incluso

- **Estensioni**
  - Estensione 6a: Il sistema non riesce a autorizzare il pagamento
    1. Il cliente può reinserire i dati della sua carta di credito

Estensione Caso d'uso

RACCOLTA DEI REQUISITI - USE CASE DIAGRAM

- Contiene l'insieme dei casi d'uso
- Mostra quali sono gli attori che portano a termine i casi d'uso e le relazioni tra i casi d'uso stessi (Es: rel. di include)



TEMPLATE PER LA DESCRIZIONE DEI REQUISITI UTENTE

Tipo	• **FR**- Funzionale – (riferito ai comportamenti del sistema in relazione a suoi input e a situazioni specifiche) • **NF**- Non Funzionale - (riferito ai vincoli sulla qualità del sistema come le performance, compatibilità, disponibilità…) • **SR** – Architetturale e di sistema - (requisiti a livello architetturale, ovvero come dovrebbe essere costruito il sistema al fine di venire incontro alle esigenze specificate in **FR** e **NF**)
ID Sequenziale	Identifica il requisito all'interno del caso d'uso. E' di tipo sequenziale
UR Riferimento	Riferimento del requisito utente. Costruito nel seguente modo: <**Tipo**> <**ID caso d'uso**>-<**ID sequenziale**>
Descrizione	Descrizione testuale sommaria del requisito.
Ruolo utente finale	Il ruolo dell'utente finale nei confronti del requisito, ad esempio l'amministratore di sistema, il manager, l'operatore, etc.
Priorità	• **Must Have** (requisito obbligatorio, senza di esso il sistema perde la sua funzionalità principale e non ha senso), • **Should have** (requisito richiesto, ma se non viene implementato adesso bisogna implementarlo prima o poi), • **Could have** (requisito non necessario per il funzionamento principale ma comunque utile), • **Won't have** (utile ad averlo in futuro)
Metodo di verifica	• **Ispezione** – ispezione a alto livello del sistema per verificare che i requisiti richiesti ci sono. Ad esempio verifica che è stato creato un bottone per una funzionalità richiesta o che sono stati inseriti i nuovi campi a una *form* in accordo con le nuove funzionalità. • **Dimostrazione –** Il sistema viene utilizzato nel modo in cui è stato pensato per essere usato. Il fine è quello di verificare che il comportamento ottenuto dal sistema è uguale a quello atteso • **Test –** Verifica sistematica del sistema attraverso l'utilizzo di un set predefinito di input. Confronto tra i risultati ottenuti e quelli attesi per quel set di input • **Analisi** – analizza come la variazione di input al sistema influenzi l'output prodotto in termini di performance o di disponibilità e prevedendo quale possa essere un valore di soglia prima della caduta del sistema

LA DESCRIZIONE DEI REQUISITI UTENTE: UN ESEMPIO

Requisiti per il caso d'uso Use Case ID: 00*

Tipo	ID Sequenziale	UR riferimento	Descrizione	Ruolo Utente finale	Priorità	Metodo di verifica
FR	00	FR 00-00	Il sistema deve avere in input due numeri e deve dare in output la loro somma	Manager	*Should Have*	Test: • (3, 5) -> 8; • (0, 2)-> 2;
NF	01	NF 00-01	Il sistema deve essere retro-compatibile con gli input delle versioni precedenti	Manager	*Must have*	Dimostrazione

*Nota: i requisiti possono essere spostati su altri casi d'uso, ES: per Use Case ID 01 avrei rispettivamente FR 01-00 e NF 01-01

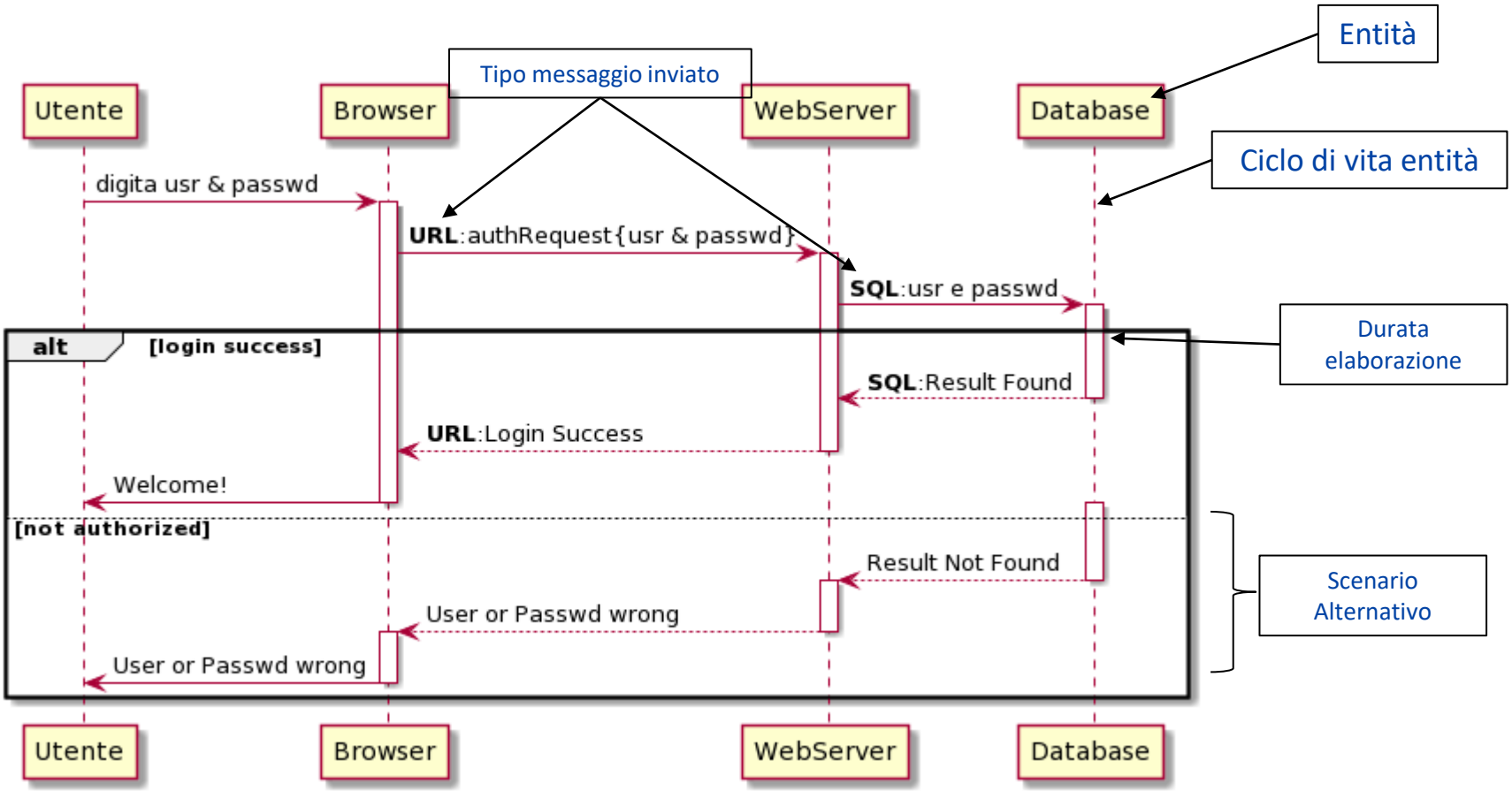
PIANIFICAZIONE

- I requisiti vengono ulteriormente definiti e analizzati dal team di sviluppo
- Si studia la loro fattibilità e la loro complessità
- Si stimano i tempi per rilasciare il prodotto richiesto dal cliente

DESIGN

- Si fanno le scelte architetturali
  - Si basano sui requisiti raccolti durante la fase precedente
  - andranno poi a definire come sarà sviluppato il software
  - Es:
    - archiettura cloud o classica
    - Database relazionale o non relazione
    - Linguaggio di programmazione da utilizzare
  - E’ possibile utilizzare **Sequence diagram** per descrivere e spiegare le interazioni tra le entità

FASE DI DESIGN – SEQUENCE DIAGRAM

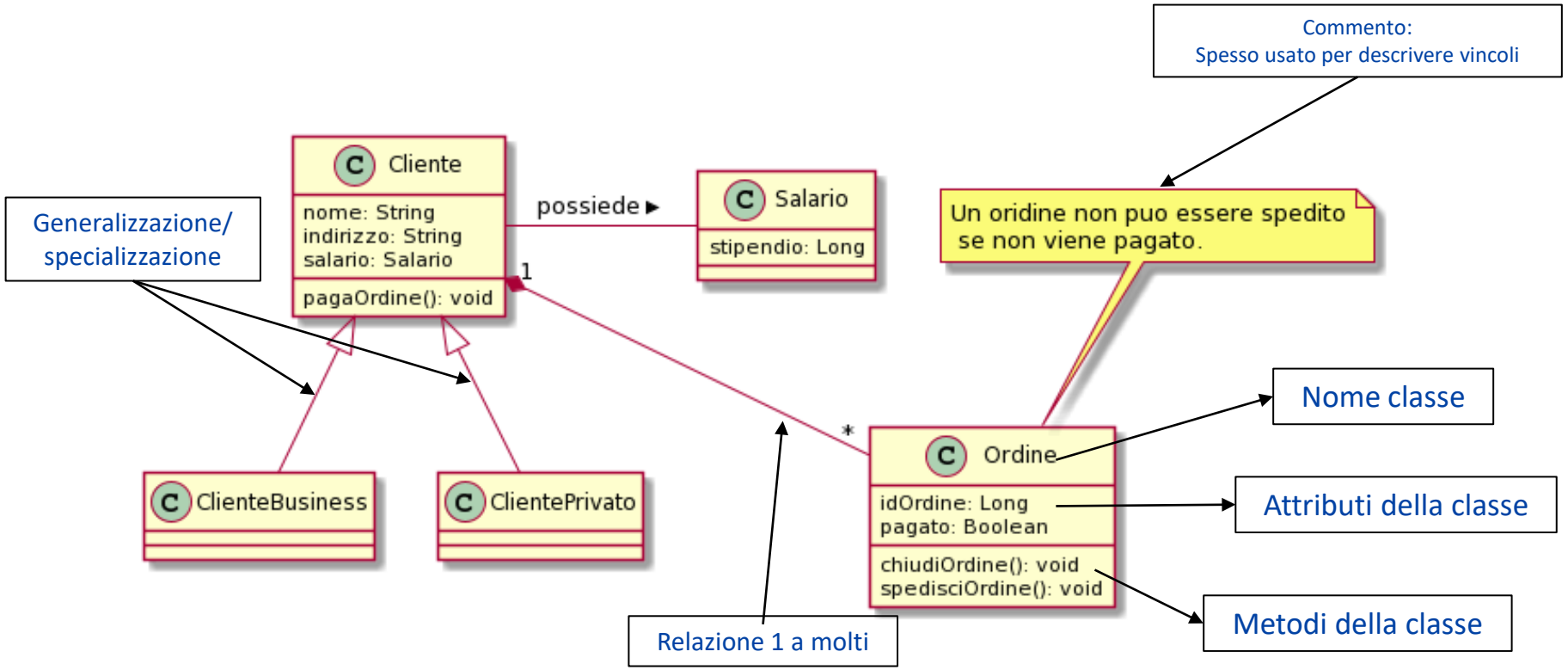


SVILUPPO & TESTING

- È l'attività in cui i componenti definiti in fase architetturale vengono codificati a basso livello
- All'implementazione corrisponde un'attività di testing volta a verificare se il sistema si comporta nel modo giusto e rispetta i requisiti stabiliti nelle fasi precedenti

SVILUPPO - CLASS DIAGRAM

- Un class diagram descrive i tipi di oggetti nel Sistema e le relazioni statiche esistenti tra di essi.



TESTING

- Il software viene testato per individuare errori che sono stati commessi inavvertitamente durante le varie fasi del ciclo di vita del software
- il test richiede uno sforzo elevato e deve essere condotto in maniera sistematica
  - Lo sviluppatore potrebbe non prevedere alcuni errori del software, per cui è importante definire una strategia per coprire il maggior numero di casi possibile
- Una strategia deve includere test a basso livello per **verificare** che un modulo sia stato realizzato correttamente e test ad alto livello per **validare** le principale funzioni rispetto ai requisiti dell'utente
- Ancora una strategia di test può essere di tipo **white box** o **black box**

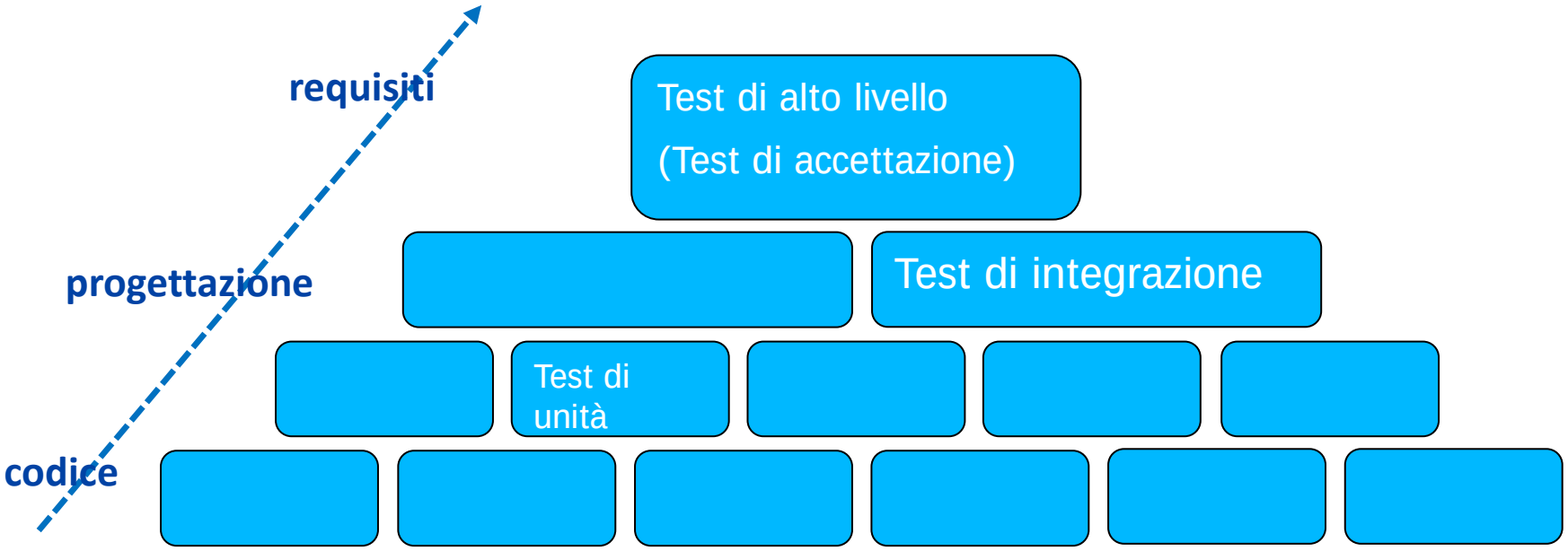
VERIFICA VS VALIDAZIONE

- **Verifica**: insieme di attività che assicurano che il software realizzi correttamente una determinata funzione
  - *Stiamo costruendo il prodotto nel modo giusto?*
- **Validazione**: altro insieme di attività che assicurano che il software costruito rispetti i requisiti del cliente
  - *Stiamo costruendo il prodotto giusto?*

WHITE BOX VS BLACK BOX

- **White box**:
  - Sfrutta la struttura di controllo del processo per ricavare i casi da testare
  - Garantisce che i cammini dell'algoritmo vengono esaminati
  - Esegue i rami vero o falso delle condizioni logiche
  - Controlla i cicli all'interno di un algoritmo
- **Black box**
  - Sono orientati al controllo del comportamento e si concentrano sui requisiti del software
  - Funzioni errate per certi input
  - Errori nelle prestazioni

I PASSI DEL TESTING

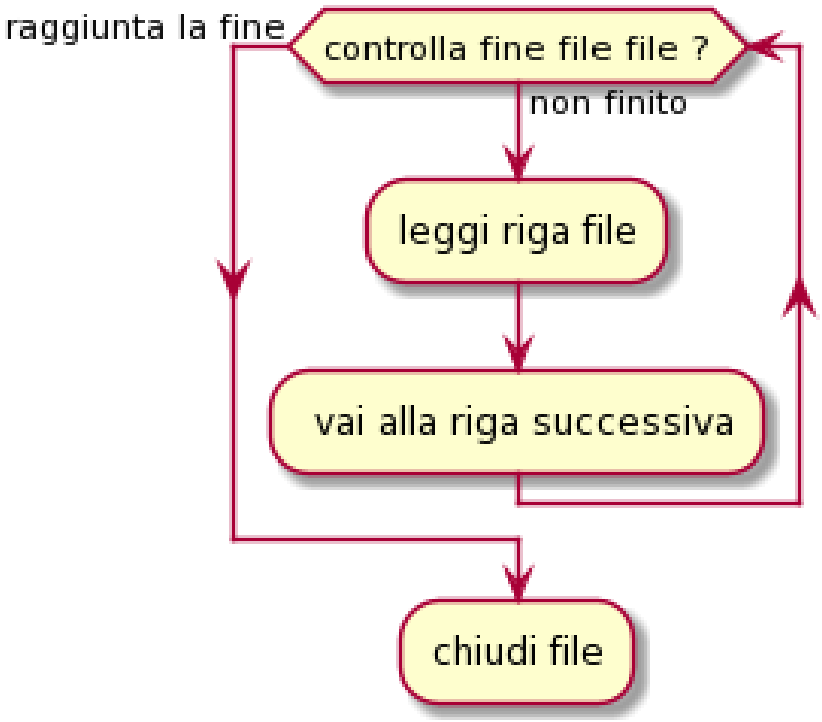


TEST DI UNITÀ

- Si concentra sulla più piccola unità di progettazione software
  - Ad esempio un metodo di una classe
- Vengono esaminati i casi limite (*boundary test*)
  - Es: ultimo elemento di un array, ultima iterazione di un loop, valori minimi e massimi…
- Viene esaminata l'integrità delle strutture dati locali all'interno di un metodo
- Vengono esercitati tutti i cammini nel flusso dell'algoritmo
  - Un cammino di flusso di un algoritmo può essere rappresentato in UML tramite un *activity diagram*

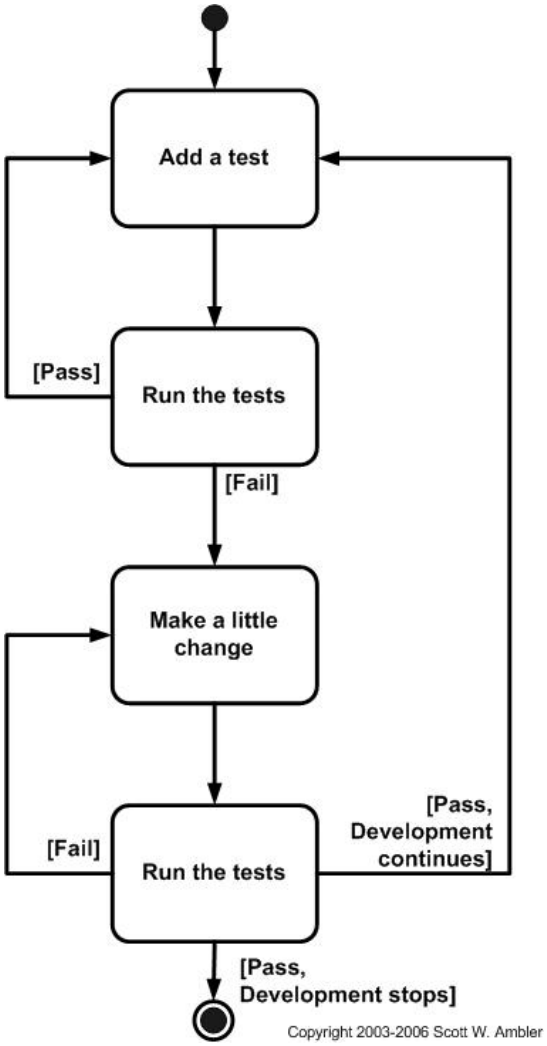
TEST SUL FLUSSO DI DATI – ACTIVITY DIAGRAM

- Gli activity diagram vengono utilizzati per descrivere la logica di un processo come ad esempio il flusso di un algoritmo o una procedura per la soluzione di un problema
- È di tipo **white box**



- I cammini di flusso da testare in questo caso sono
  - raggiunta fine file
  - file non finito

TEST DRIVEN DEVELOPMENT

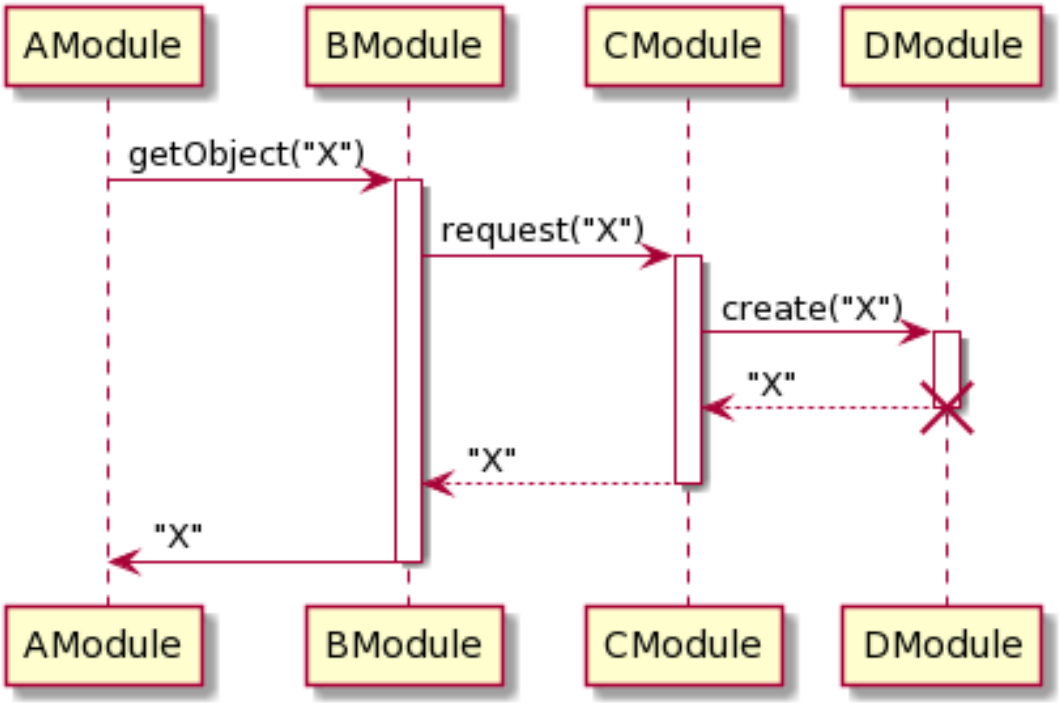


TEST DI INTEGRAZIONE

- Testa l'interfacciamento tra i moduli test, in quanto essi potrebbero non funzionare correttamente quando vengono aggregati
  - La loro integrazione potrebbe produrre effetti non previsti
    - Se l'interfaccia di un modulo è stata cambiata un altro modulo che la utilizzava potrebbe non funzionare correttamente
    - L'integrazione di un modulo con un altro potrebbe percorrere flussi in un modulo per cui esso non è stato testato
- La letteratura suggerisce un approccio **incrementale** al test di integrazione
  - Se aggregassi tutti i moduli insieme ci sarebbe una situazione caotica
  - integrazione top down
  - Integrazione bottom up
  - Test di regressione
    - Si dovrebbe effettuare ogni volta un modulo viene aggiunto o modificata la sua interfaccia

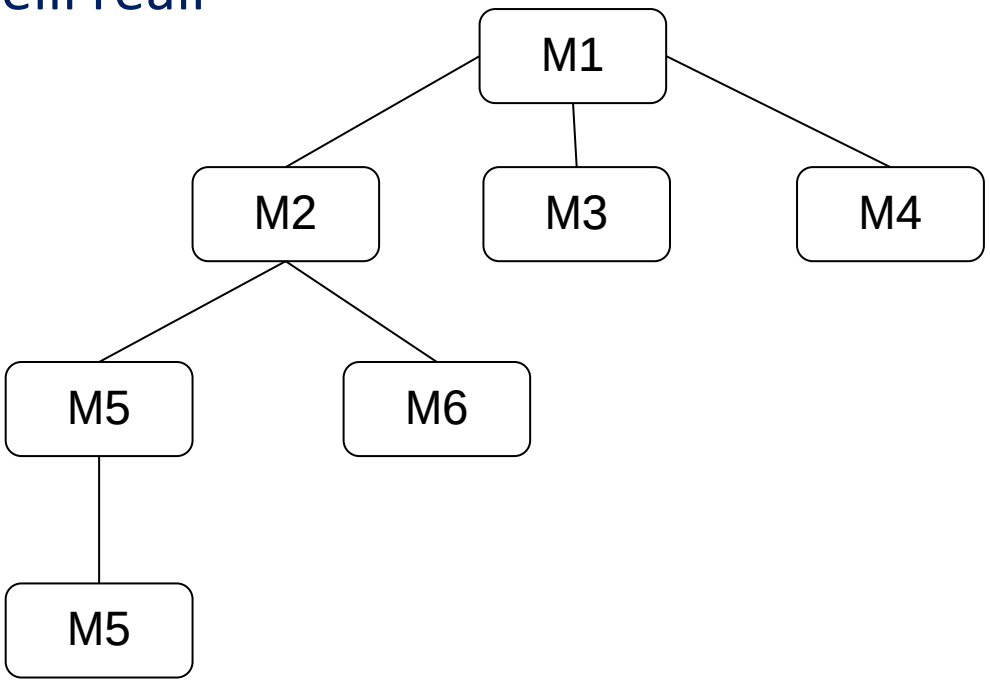
TEST DI INTEGRAZIONE – SEQUENCE DIAGRAM

- Tipicamente un sequence diagram serve per descrivere relazioni tra entità, in questo caso moduli
  - I moduli A, B, C e D comunicano tra loro mediante i metodi *getObject*, *request* e *create*. Cosa succederebbe se cambiassero le loro interfacce?



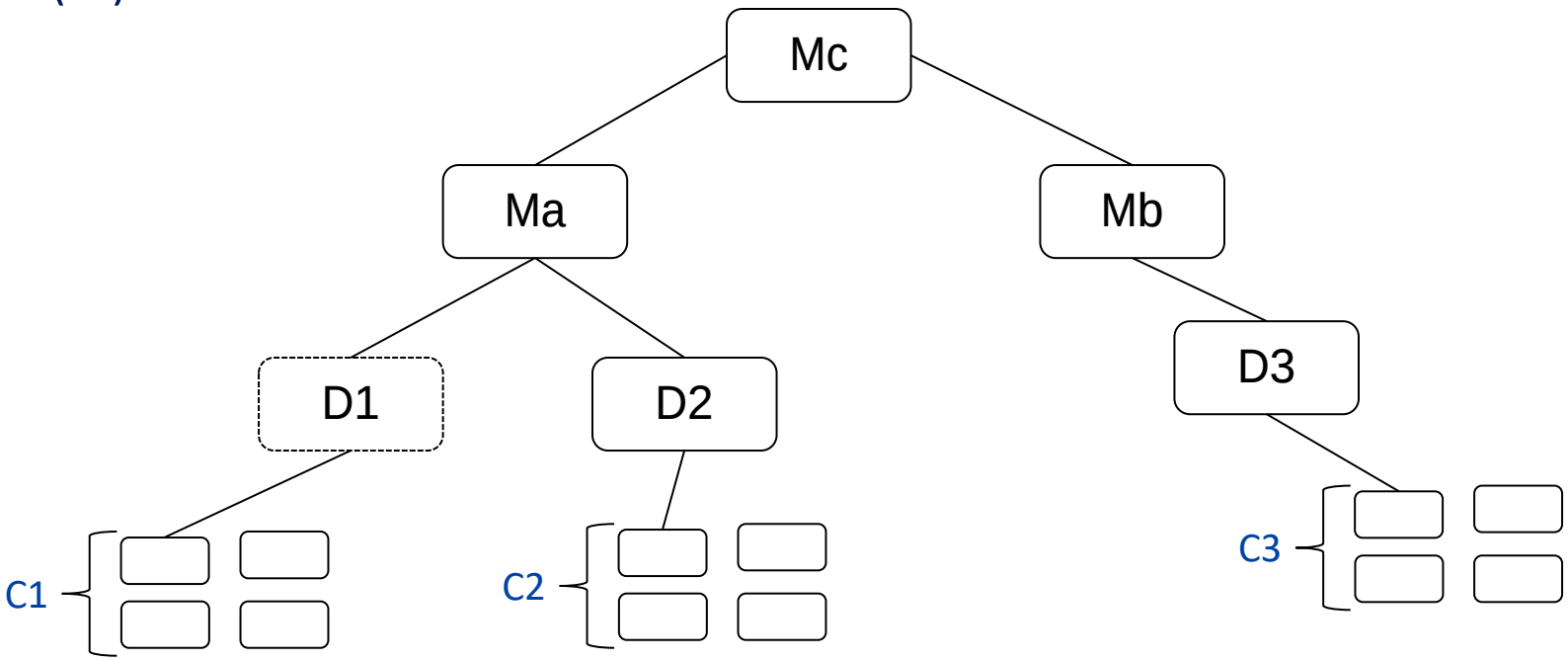
TEST DI INTEGRAZIONE – INTEGRAZIONE TOP DOWN

- Si parte dal modulo principale (M1) e si scende nell'albero delle dipendenze procedendo per sostituzione dei moduli fittizi (mock) con quelli reali



TEST DI INTEGRAZIONE – INTEGRAZIONE BOTTOM UP

- I moduli a basso livello vengono raggruppati per cluster (C) che realizzano una data funzionalità e vengono testati da moduli pilota (D) che coordinano gli ingressi e le uscite dei casi di prova
- In seguito i moduli D vengono rimossi e cluster aggregati ai moduli principali (M)



TEST DI ACCETTAZIONE

- Approccio di tipo **black box**
- E’ un test di **validazione**, serve per verificare che il prodotto rispetti i requisiti funzionali

SUDDIVISIONE IN CLASSI DI EQUIVALENZA

- Le classi di equivalenza suddividono il dominio dell'input in classi di dati.
  - Una classe di dati può individuare un certo tipo di errori
- Una classe di equivalenza rappresenta un insieme di stati validi o non per una condizione di ingresso
  - Es: se in input devo inserire un intervallo di valori, avrò due classi di equivalenza non valide per i dati che sono a sinistra e destra dell'intervallo e una classe di equivalenza che identifica l'intervallo di valori validi

TEMPLATE PER LA DESCRIZIONE DEI TEST

Informazioni generali			
Categoria	Categoria a cui appartiene il test		
Data del test	Data di esecuzione del test	**Testbed**	Piattaforma su cui è stato condotto il test
Descrizione del test	Descrivere cosa si propone di fare questo test		
Test case Id	Id del test case	**Risultato**	PASS/NOT PASS
Test			
Requisiti da testare	Indicare la lista dei requisiti utente che si intende validare in questo test, ad es: • **FR 00-00** • **NF 00-01**		
Ruoli e responsabilità	Indicare il responsabile del test		
Passi da seguire	Descrivere attraverso una sequenza di passi numerata come si esegue il test		
Obiettivi del test	Fornire una descrizione informale di cosa si vuole testare. Dovrebbe essere un sunto dei requisiti dichiarati al campo «Requisiti da testare»		
Risultato atteso	Cosa ci si aspetta come risultato. Con riferimento a **NF 00-01** ad esempio, l'utente inserisce un input della vecchia versione e ci si aspetta che esso funzioni senza problemi nella nuova versione del software		
Requisiti del test			
Hardware	Indicare l'hardware utilizzato per condurre il test, ad esempio: • Spazio sul disco • Processore • memoria		
Software	Indicare il software utilizzato per condurre il test, ad esempio: • Versione delle librerie • Versione di java		
Risultato attuale			
Valutazione risultato	Mostrare con una serie di screenshot o con un breve filmato la conduzione del test		

RILASCIO DEL SOFTWARE & MANUTENZIONE

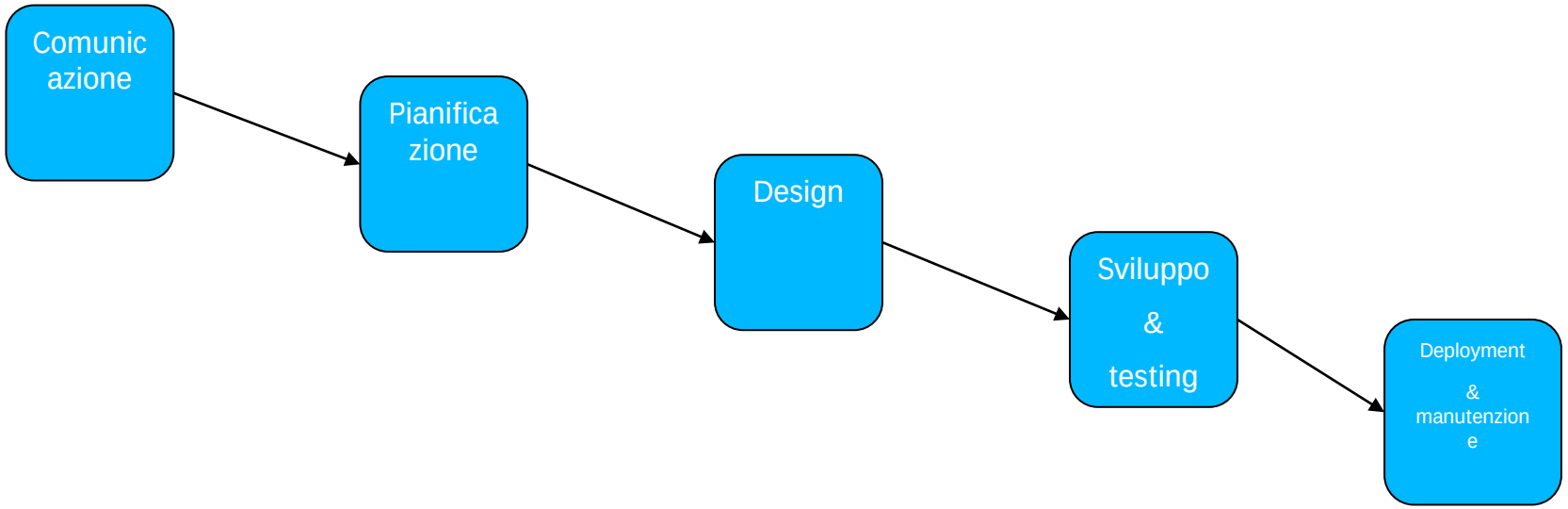
- Una volta che il software viene rilasciato per l'utente finale
- Il software come tanti altri prodotti è soggetto a manutenzione
  - Gli utenti finali scoprono errori che devono essere risolti
  - Il codice diventa obsoleto, alcune librerie esterne possono non essere più supportate per cui è opportuno aggiornarle
  - Spesso l'utente finale o il team preposto al testing scopre un errore nel sistema

METODOLOGIA DI SVILUPPO DEL SOFTWARE

- Sono state definite diverse metodologie per lo sviluppo del software
- Ogni metodologia viene poi adattata per poter essere calata nel contesto di un determinato progetto software
- I principali modelli di sviluppo software sono
  - Modello a cascata (*waterfall*)
  - Modello a spirale
  - Modello agile

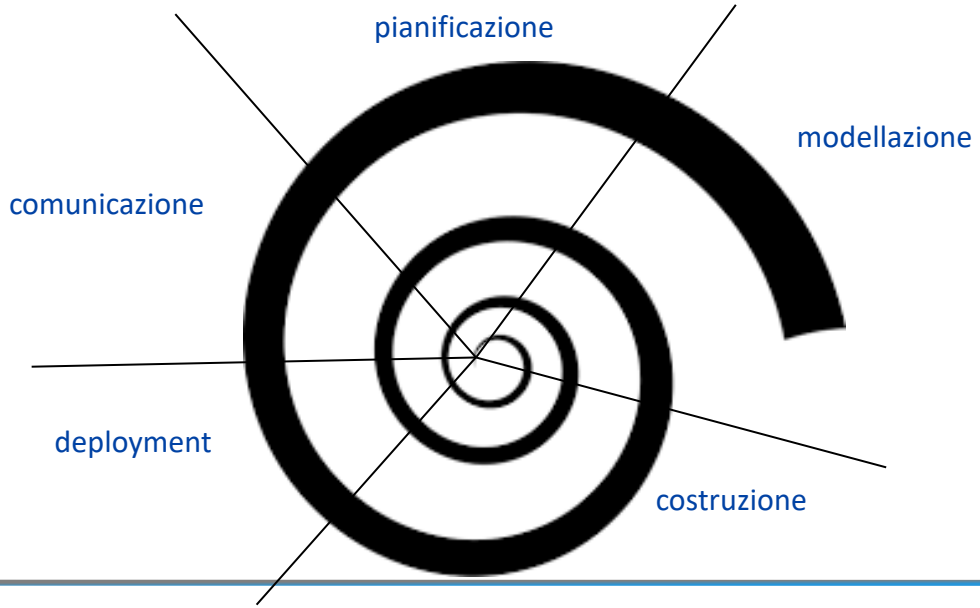
Modello a cascata (waterfall model)

- Il più vecchio e classico tra i paradigmi
- Riproduce linearmente le fasi del ciclo di vita del software
- Ha un approccio poco incline ai cambiamenti (è difficile tornare indietro una volta completata una fase) ed è sequenziale (non può essere parallelizzato)
- Viene usato quando i requisiti di un problema sono abbastanza noti a priori



MODELLO A SPIRALE

- Quando i requisiti sono troppo vaghi in genere si ricorre allo sviluppo di un **prototipo**
  - Un **prototipo** serve a far comprendere sia lato cliente/utente che lato sviluppatore quali sono i requisiti da soddisfare
- In un modello a spirale il software viene sviluppato attraverso una serie di release evolutive che partono da un prototipo o un modello cartaceo e permettono il rilascio di versioni via via più sofisticate del software
- Ogni ciclo della spirale rilascia una versione secondo il paradigma a cascata



MODELLAZIONE AGILE

- È un modello avanzato di processo di sviluppo
  - La metodologia agile più popolare è la Scrum
  - Combina elementi statici del ciclo di vita del software con la dinamicità di adattamento alle esigenze del cliente
  - Fornisce al team di sviluppo più capacità di auto gestirsi e più responsabilità
    - Per ogni ciclo di sviluppo (sprint) il team valuta quali e quanti compiti può prendersi in base alle persone che lo compongono
    - Un team è composto dagli sviluppatori e da uno scrum master che ha il compito di dirigere il team e interfacciarsi con l'ambiente esterno nel caso di problemi
  - Durante lo sprint si effettuano le seguenti riunioni
    - *Planning*: si tiene all'inizio di ogni sprint con l'obiettivo di pianificare e concordare il software da rilasciare per quello sprint.
      - Alla riunione partecipa il team di sviluppo e chi è incaricato di spiegare i requisiti (product owner)
    - *Daily*: riunione giornaliera tenuta dai membri del team in cui ognuno spiega cosa ha fatto il giorno prima e se ha avuto problemi e quali sono i suoi intenti per quella giornata
    - *Review*: il team mostra ai manager e al product owner una demo del software rilasciato e concordato in fase di planning
    - *Retrospective*: a fine sprint il team fa il punto della situazione su cosa è andato bene, cosa è andato male e cosa c'è da migliorare

RIFERIMENTI UTILI

- R. Pressman - Principi di ingegneria del software
- https://www.uml.org/
  - Sito ufficiale UML
- UML Distilled: A brief Guide to the Standard Object Modeling Language
  - Una guida per l'UML
- https://plantuml.com/
  - Utile per scrivere diagrammi UML

CONTACTS