Corso di laurea in
Informatica

INGEGNERIA DEL SOFTWARE
M - Z

Anno accademico 2024/2025 - Docente: **ANDREA MARIO CALVAGNA**

Risultati di apprendimento attesi

1. Conoscenza e capacità di comprensione (knowledge and understanding): lo studente conoscerà in modo approfondito i meccanismi di programmazione ad oggetti, le caratteristiche di modularità del software, la gestione dello sviluppo del software.

2. Capacità di applicare conoscenza e comprensione (applying knowledge and understanding): lo studente saprà progettare, documentare e implementare sistemi software ad oggetti, saprà distinguere i ruoli dei partecipanti allo sviluppo del software, saprà implementare test per verificare la correttezza del software.

3. Autonomia di giudizio (making judgements): lo studente acquisirà la capacità di analizzare la modularità dei sistemi software.

4. Abilità comunicative (communication skills): lo studente imparerà a descrivere in linguaggio tecnico le componenti software.

5. Capacità di apprendimento (learning skills): lo studente potrà affrontare e risolvere problemi di progettazione e implementazione in ambiti realistici, studiando, valutando e utilizzando, sulla base dei fondamenti di questo corso, tecnologie innovative.

Modalità di svolgimento dell'insegnamento

Lezioni frontali.
Qualora l'insegnamento venisse impartito in modalità mista o a distanza potranno essere introdotte le necessarie variazioni rispetto a quanto dichiarato in precedenza, al fine di rispettare il programma previsto e riportato nel syllabus.

Prerequisiti richiesti

Basi di programmazione di algoritmi in C o C++ o Java.

Frequenza lezioni

La frequenza delle lezioni è indispensabile per la comprensione degli argomenti trattati.

Contenuti del corso

Processi di sviluppo del software: cascata, evolutivi e agili (spirale, XP, Scrum), tempistica delle attività svolte. Gestione dei requisiti del software: raccolta ed analisi. Notazione UML: diagrammi di casi d’uso, attività, stati, classi, sequenza e collaborazione. Gestione codice tramite Git.

Progettazione ad oggetti, ereditarietà e polimorfismo. Qualità del software. Design Pattern illustrati tramite obiettivi, contesto, problema, esempi, soluzione con diagrammi e codice: Singleton, Factory Method, Adapter, Bridge, Composite, Decorator, Facade, Chain of Responsibility, Mediator, Observer, State.

Tecniche di Refactoring.

Stream in Java e parallelismo.

Metriche sul software, ed evoluzione del software.

Tecniche di test per convalidare il software.

Testi di riferimento

1. Ravi Sethi - Software Engineering: Basic Principles and Best Practices - Cambridge University Press.
2. E. Gamma, R. Helm, R. Johnson, J. Vlissiders. Design Patterns – Elements of Reusable Object-Oriented Software. Pearson Addison-Wesley.
3. M. Fowler. UML Distilled. Pearson. 2010
4. R.-G. Urma, M. Fusco, A. Mycroft. Java 8 in Action: Lambdas, streams, and functional-style programming. Manning. 2015
5. K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley. 1999
6. M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley.
7. Dan S. Myers - Data Structures and Algorithms in Java - Cambridge University Press.

Programmazione del corso

	Argomenti	Riferimenti testi
1	Obiettivi dell'ingegneria del software. Caratteristiche del software: complessità, modificabilità, correttezza, specifiche, test	1
2	Caratteristiche del software: astrazioni (classi). Principio di singola responsabilità per classi e metodi.	1,2
3	Qualità del codice e anti-pattern (God class, Spaghetti Code). Progettazione ed implementazione di test.	1,2
4	Tecniche di Refactoring: estrai metodo, sostituisci temp con query, dividi variabile temp.	6
5	Design pattern Singleton. Creazione di istanze di classi. Metodi statici. Visibilità di classi, attributi e metodi. Concetti di information hiding, coesione, coupling. Concetti e meccanismi di ereditarietà e polimorfismo.	1,2
6	Design Pattern Factory Method (con varianti riflessiva, object pool, dependency injection).	2
7	Identificazione e progettazione classi, uso di interfacce, ereditarietà e polimorfismo. Compatibilità di tipi. Dispatch di chiamate di metodo. Override e overload.	1,2
8	La notazione UML per i diagrammi delle classi, di sequenza, di collaborazione.	3
9	Design pattern Adapter (versione Object e Class), e Façade.	2
10	Diagrammi UML degli stati. Design pattern State	2,3
11	Design pattern Observer e Publish-Subscribe.	2
12	Riconoscimento di design pattern dal codice e dai diagrammi UML delle classi	2

	Argomenti	Riferimenti testi
13	Design pattern MVC, Mediator, e Decorator.	2
14	Processi di sviluppo agili: introduzione alle pratiche del processo extreme programming (XP).	5
15	Processo XP: pratiche di pianificazione (story card e CRC), pair programming, design semplice, refactoring, piccole release, standard di codifica, cliente in sede, test. Cenni di processo Scrum.	5
16	Fasi per lo sviluppo del software: raccolta e analisi requisiti, progettazione, codifica, convalida, evoluzione.	1
17	Processi di sviluppo del software: cascata, evolutivi e a spirale.	1
18	Requisiti del software: raccolta, tipi di requisiti, linee guida per la scrittura dei requisiti.	1
19	Progettazione ed implementazione con Java Stream e parallelismo. Filter, map, reduce. Tipi Predicate, Function, Supplier, Optional.	4
20	Esempi di programmazione con Stream Java, e concorrenza con stream.	4
21	Design pattern Composite, Chain of Responsibility. Conseguenze sul codice dell'adozione di vari design pattern	2
22	Design pattern Bridge. La notazione UML dei diagrammi dei casi d’uso, e delle attività.	2,3
23	Evoluzione del software, categorie di cambiamenti e gestione dei cambiamenti. Leggi di Lehman.	1
24	Metriche sul software: complessità ciclomatica, linee di codice. Metriche per sistemi ad oggetti: suite di Chidamber e Kemerer. Fase di test. Test di componenti, metriche di copertura del codice. Test regressivi.	1

Verifica dell'apprendimento

Modalità di verifica dell'apprendimento

L'esame è strutturato in un compito scritto, consistente in varie domande a risposta multipla ed alcune domande aperte, e una prova orale.

The final examination is split in a written test, consisting of several multiple choice questions and some open questions, plus an oral interview.

Esempi di domande e/o esercizi frequenti

Disegnare il diagramma UML delle classi di un design pattern

Descrivere gli obiettivi di un design pattern

Dire quali sono le conseguenze (sul codice) di un design pattern

Descrivere il funzionamento di un design pattern

Implementare una classe di un design pattern

Implementare una selezione dati e una trasformazione dati tramite Stream Java

Descrivere il metodo filter (map, reduce) e mostrarne la sintassi d'uso e un esempio

Descrivere il tipo Predicate (Function, Supplier)