ALMA MATER STUDIORUM A.D. 1088

UNIVERSITÀ DI BOLOGNA

Scuola

Ingegneria e Architettura

Corso di studio (laurea magistrale)

Ingegneria e Scienze informatiche

Sede didattica

Cesena

A.A.

2017/2018

Insegnamento

Paradigmi di Programmazione e Sviluppo

C.I. Sviluppo di Sistemi Software

Titolare

Prof. Mirko Viroli

Sommario

Course introduction

Homepage del corso: http://mirkoviroli.apice.unibo.it

Tutor: Dott. Roberto Casadei roby.casadei@unibo.it

Portale APICE: http://apice.unibo.it/xwiki/bin/view/Courses/PPS1718

Moodle: https://elearning-cds.unibo.it/course/view.php?id=12038

Books

Advanced OO programming techniques

- "Effective Java", Second Edition, Joshua Bloch
- "Design Patterns": Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
- "Java 8 Lambdas", Richard Warburton

Functional programming

- "Functional Programming in Scala", Paul Chiusano and Runar Bjarnason
- "Programming in Scala", Bill Venners, Lex Spoon, and Martin Odersky
- "Functional Programming Patterns in Scala and Clojure", Michael Bevilacqua-Linn
- "Programming Scala: Scalability = Functional Programming + Objects", Alex Payne and Dean Wampler

Logic programming

- "The Art of Prolog", Second Edition, Leon S. Sterling and Ehud Y. Shapiro

Software Engineering

- "Fundamentals of Software Engineering", Giorgio Ghezzi, Dino Mandrioli and Mehdi Jazayeri
- "Clean Code", Robert Cecil Martin
- "The Art of Agile Development", James Shore
- "Scrum and XP from the Trenches: How We Do Scrum", by Henrik Kniberg
- "Test Driven Development: By Example", Kent Beck
- "JUnit in Action", Second Edition, Petar Tahchiev, Felipe Leme, Vincent Massol and Gary Gregory
- "Testing in Scala", Daniel Hinojosa

Software

Language frameworks

- Java (https://www.java.com)
- Scala (https://www.scala-lang.org)
- tuProlog (http://tuprolog.apice.unibo.it)

IntelliJ as IDE (**https://www.jetbrains.com/idea**)

- Java support by default
- Scala support via plugin (https://plugins.jetbrains.com/idea/plgin/1347-scala)

Other development & management tools

- Gradle build tool (https://gradle.org)
- SBT build tool (http://www.scala-sbt.org)
- Git dvcs (https://git-scm.com)

On our languages and paradigms

L'uso di JVM come framework tecnologico:

- permette di guadagnare in termini di coerenza;
- spiana la strada verso le integrazioni;
- ricerca un buon livello di operatività.

Un'alternativa possibile è costituta da .NET.

Per quanto riguarda l'object-orientation la preferenza – per questo corso – ricade su Java che, essendo oggi di gran lunga il più popolare linguaggio OO, rappresenta una scelta ovvia per i corsi Universitari.

Come software per studiare il paradigma funzionale, tra le possibili alternative, si utilizzerà Scala che è il secondo linguaggio JVM-based per popolarità ed è un perfetto strumento di studio e allenamento in quanto molto sofisticato e avanzato.

Infine si userà tuProlog per studiare il paradigma logico.

Exam

Durante il corso saranno dettagliati tre *assignments* opzionali:

i. code re-engineering
ii. Scala
iii. Prolog

le cui soluzioni dovranno essere consegnate durante le attività di laboratorio e che costituiranno materiale di discussione durante l'esame.

Sarà invece obbligatorio un progetto finale da scegliere, negoziare, sviluppare e documentare appropriatamente.

Il voto d'esame costituisce il 50% del voto del corso integrato.

L'esame consiste nella valutazione del progetto e nella sua discussione e altre domande, incluse eventuali discussioni degli *assignments*.

Due elementi chiave della valutazione sono:

1. L'operatività con i linguaggi, le tecniche e gli strumenti
2. La "conoscenza" e l'"abilità di utilizzo" delle tecniche di programmazione avanzate.

The philosophy of this course

Si tratta essenzialmente di un corso di programming-oriented software engineering.

Segue approcci moderni che si concentrano più sul codice che sui processi e alla documentazione dei processi.

Quindi: good coding practice, techniques, paradigms, tools and methods.

Relativamente al software engineering, le parole chiave sono: qualità, principi, design/architettura, specifiche, verifiche, processi/gestione, strumenti.

Learning preconditions

Strong preconditions (da recuperare rapidamente): buona conoscenza del paradigma Object-Oriented e utilizzo fluente di Java.

Weak preconditions: basi di software engineering (UML, processi), esperienza di programmazione Java, minima conoscenza del paradigma funzionale (es. Java 8 lambdas).

Part A: on quality software production

Module 01: quality programs (mechanisms and criteria)

Topic: the need of a software process

Lo sviluppo di un software è caratterizzato da una serie di problemi legati alla complessità delle applicazioni, alla difficoltà di realizzare prototipi per pezzi di software troppo piccoli, alla facile generazione di errori e alla relative difficile identificazione tramite processi di debug, al tempo necessario per rilasciare software corretto. Questi e altri motivi rendono facile il fallimento dei progetti software.

Quali sono dunque le qualità software che portano a progetti di successo? Quali sono i processi di sviluppo e quali le tecniche di programmazione che promuovono la qualità?

Alcuni pasi di sviluppo software:

- Requirement engineering (elicitation, analysis, specification): conoscere il problema raccogliendo, analizzando, capendo, organizzando e studiando le necessità
- Design: definire gli elementi chiave della soluzione (un sistema software) e procedere per raffinamenti incrementali dall'architettura ai dettagli
- Implementation: scegliere le tecnologie e sviluppare il software, scrivere codice
- Post-implementation: test, rilascio e manutenzione.

L'articolo "Frequently Forgotten Fundamental Facts about Software Engineering" di R.Glass esplora diversi aspetti dei processi di sviluppo software:

- le **persone**: "good programmers are up to 30 times better than mediocre programmers"
- la **complessità**: "for every 10-percent increase in problem complexity, there is a 100-percent increase in the software solution's complexity"
- gli **strumenti**: "learning a new tool or technique actually lowers programmer productivity and product quality initially"
- la **qualità** e **l'affidabilità**: "software that a typical programmer believes to be thoroughly tested has often had only about 55 to 60 percent of its logic paths executed"
- l'**efficienza**: "efficiency is more often a matter of good design than of good coding"
- la **manutenzione**: "maintenance typically consumes about 60 percent average of software costs"
- l'**analisi dei requisiti** e il **design**: "one of the two most common causes of runaway projects is unstable requirements"
- le **revisioni** e le **ispezioni**: "rigorous reviews are more effective, and more cost effective, than any other error-removal strategy"
- il **riuso**: "reuse-in-the-large remains largely unsolved"
- la **capacità di stima**: "because estimates are so faulty, there is little reason to be concerned when software projects do not meet cost or schedule targets"
- la **ricerca**: "many software researchers advocate rather than investigate".

Un software di qualità aiuta a:

- adattare meglio il software all'instabilità dei requisiti
- riduce i costi di manutenzione
- supporta l'individuazione e la rimozione degli errori.

Per raggiungere la qualità è necessario capire come definire un software di qualità (criteri e metriche), utilizzare pratiche di programmazione note e organizzare al meglio l'intero processo di sviluppo.

[CONTINUARE DA SLIDE 7 MODULO 01]

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 02: techiques for quality (patterns, testing, refactoring, abstraction)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Part B: functional programming

Module 03: the functional style (Java streams, lambda, Scala)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 04: mixing with OO part 1 (OO in Scala)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 05: mixing with OO part 2 (advanced mechanisms)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Part C: software development methodologies and techniques

Module 06: software methodologies (overview and focus on agile's, Scrum)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 07: test-driven development (Junit, Scalatest, ScalaCheck)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 08: advanced development tools (continuous integration)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Part D: logic programming

Module 09: basic logic programming and Prolog (unification, resolution, programming)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 10: coding and practice (few algorithms, tuProlog integration)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Part E: advanced programming and design patterns

Module 11: OO and functional patterns (in Java and Scala)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home

Module 12: internal DSLs (in Prolog and Scala)

Topic

Technical description

Theory, concepts, techniques

Complement techincal description

Guided lab experience

Work at home