# INFORMATICA III
# Parte B - Progettazione e Algoritmi

## Introduzione al corso

**Patrizia Scandurra** patrizia.scandurra@unibg.it
Università degli Studi di Bergamo
Corso di laurea magistrale in Ingegneria Informatica

# Contatti

- **Email:** patrizia.scandurra@unibg.it
- **Sito web**: http://cs.unibg.it/scandurra/INF3ProgAlg2017.html
- **Link Dropbox per materiale didattico:**

**https://www.dropbox.com/sh/o8sqf43hgwwrpdw/AAC8L6CogSRh-MVer3tPk6NUa?dl=0**

- **Ricevimento:**
  - Edificio B, terzo piano, ufficio 3
  - **??Venerdì?? mattina 8.45 - 11.45**
    o su appuntamento

# Natura

- **Lezioni frontali e di esercitazione**
  - si dovranno svolgere alcuni esercizi
  - entreremo nei dettagli del codice
- **Non informativo sulle tecnologie recenti**
  - Le tecnologie cambiano rapidamente, ma i principi rimangono
- **Prerequisiti**: fondamenti di informatica, ingegneria del software e notazione UML, programmazione Java, elementi di analisi matematica su serie e successioni

# Obiettivi

Fornire agli studenti

- conoscenze teoriche
- metodi e strumenti di sviluppo utili

per **progettare ed implementare "in grande" ed "in piccolo"una applicazione software** attraverso un **processo di sviluppo**

- **"agile"**
- **orientato alle componenti** e alle **architetture software**
- che punta all'efficienza degli **algoritmi e strutture dati** impiegate

Impiegheremo linguaggi di programmazione OO come **Java** e librerie fornite da terze-parti; utilizzeremo IDE e tool di sviluppo vari basati principalmente su **Eclipse**

# Architettura software

- Informalmente, un'**architettura software** è l'organizzazione del sistema, costituita dalle varie parti che lo compongono (*sottosistemi* e *componenti*), dalle relazioni tra di loro e con l'ambiente

- Definire un'architettura SW significa mappare funzionalità su componenti
  - Es. Modulo di Interfaccia utente, modulo di accesso al DB, modulo di gestione della sicurezza, etc..

- L'architettura di un sistema software viene definita in fase di *System Design* e vari principi di *good design* e *design pattern* ne guidano il progetto e l'evoluzione

# Argomenti (modulo B)

**Design, analisi e sviluppo di codice algoritmo**:

- **Complessità algoritmica**: complessità spazio-tempo e notazione asintotica
- **Tecniche di progettazione di algoritmi**: strategia incrementale, divide et impera, greedy, programmazione dinamica
- **Design e realizzazione di algoritmi e strutture dati** (in Java): strutture dati e algoritmi elementari (liste, pile, code), algoritmi di ordinamento, alberi (alberi binari di ricerca, B-tree, RB-tree), applicazione degli alberi per il processamento di documenti XML, tabelle hash, grafi e cammini minimi

(Prima parte, *design in piccolo*)

# Argomenti (modulo B)

**Design e sviluppo di architetture SW**:

- **Processi di sviluppo agili:** modellazione, sviluppo e testing con AMDD (Agile Model-driven Development)
- **Analisi e specifica dei requisiti SW:** use case modeling, descrizione dei casi d'uso e degli scenari
- **Component-based development**: componenti, architetture software, design pattern (stili) architetturali, architetture SW per applicazioni service-oriented e Cloud-based, architetture SW per sistemi self-adaptive, il modello a componenti della piattaforma Android
- **Design e sviluppo di GUI e componenti grafiche**: JFC/Swing e Java2D, gestione eventi, MVC e componenti Swing di base
- **Deployment di un'applicazione SW**: Java convention, deployment e subversioning
- **Analisi statica e strutturale del SW:** definizione/calcolo di metriche di qualità del software con stan4j, refactoring del codice
- **Analisi dinamica di un'applicazione SW:** unit testing con Junit, copertura del codice con il plug-in Eclipse ECLEMMA

(Seconda parte, *design in grande*)

# Argomenti non trattati

- Tecniche di analisi di architetture SW dal punto di vista di *requisiti non funzionali* (performance, availability, reliability, ecc..)

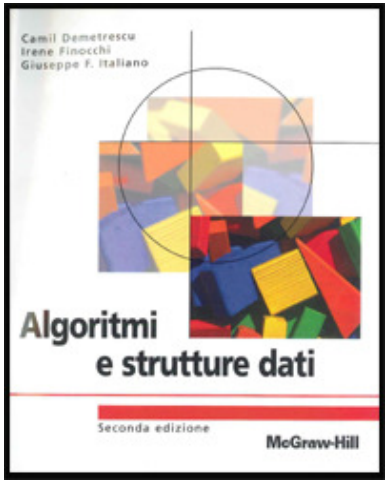
  Esempio: Analisi delle performance di una architettura SW

- Project management

# Materiale didattico (parte B)

- **Lucidi e dispense** distribuite attraverso la cartella Dropbox

- **C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati**, **McGraw-Hill**, ISBN: 978-88-386-6468-7, seconda edizione, Gennaio 2008
  http://www.ateneonline.it/demetrescu2e/homeA.asp




*Per approfondimenti*:

- **C.A.Shaffer: A practical Introduction to data structures and algorithm analysis**.
  http://people.cs.vt.edu/~shaffer/Book/Java3e20100119.pdf
- **Software Architecture in Practice (3rd Edition)** SEI Series in Software Engineering, Addison Wesley, 5 October 2012
- **The Object Primer 3rd Edition: Agile Model Driven Development with UML 2**, Cambridge University Press, 2004 ISBN 0-521-54018-6

# Modalità d'esame (parte B)

- L'esame consta di:
  - una **prova scritta** che verte sulla prima parte del corso
  - **progettino software** (**prova orale**)
    - Sono ammessi team di sviluppo composti da gruppi di al max 3 persone
    - L'idea del progettino software è libera ma va concordata con il docente
    - Il progettino SW prodotto dovrà esibire certe caratteristiche obbligatorie (vedi lucidi successivi)
- Il giorno dell'appello coincide con la prova scritta
- Superata la prova scritta, consegnerete al docente il progetto SW (codice + documentazione) tramite un cloud storage (ad es. Dropbox) prima della data fissata per la prova orale

# Distribuzione del codice del progetto software

- Al di fuori dell'IDE: installazione e funzionamento stand-alone con un installer o con uno script o con Java web start. Usate fatjar se volete mettere più jar in un unico Jar
- In linea di principio, non deve richiedere al docente l'installazione di SW ausiliario per farlo funzionare
  - Se necessario un DB, preferite Oracle JavaDB (soli 2.6 MB)
- Il codice deve essere sviluppato (e consegnato al docente) in un (unico) progetto di Eclipse e deve includere librerie e risorse necessarie per compilare/eseguire i file e i casi di test
  - I link alle risorse devono essere *relativi* in modo che spostando il progetto su vari PC esso continui a funzionare
  - Se utilizzate altri progetti ausiliari, anche questi vanno inclusi

# Documentazione del progetto software

- Sintetico manuale utente e di installazione

Documenti da produrre in ogni fase incrementale (almeno 3!) del processo agile AMDD:

- Documento di specifica dei requisiti SW
- Documento per il design dell'architettura SW e degli algoritmi
- Documenti per l'analisi dinamica
  - Documento che definisce gli *unit test* e report di copertura prodotto con Junit/Eclemma
  - Eventuale report per altre forme di testing, ad es. testing black-box di interfacce grafiche
- Documento per l'analisi statica e strutturale
  - *Quality report* prodotto con STAN4J

http://stan4j.com/sample-report.html

# Modalità di valutazione (1)

**Voto finale: media aritmetica tra voto p. scritta e voto p. orale**

**Parte scritta**: voto tra 0 e 30; **18 = sufficiente**
(**BARRIERA per la prova orale**)

**P. Orale (discussione progetto SW):** Il voto è ottenuto valutando il progetto SW su ognuna delle seguenti caratteristiche "pesate":

- ***Interesse del problema affrontato e dimensione del progetto*** **10%**
  - Premio l'originalità, difficoltà, l'interdisciplinarietà, fondamenti teorici, l'interesse potenziale, dimensione del progetto, uso di librerie, ecc.
- ***Bontà delle soluzione proposta:*** **60%**
  - *Architettura SW (20%) e qualità del codice*: stile di programmazione, opportuna decomposizione del design in componenti e package (valutabile tramite analisi statica e strutturale), impiego di design pattern, ecc..
  - Algoritmi e strutture dati (20%): strutture dati e algoritmi user-defined, uso di JFC
  - *Validazione (20%)*: risultati ottenuti ad esempio dal testing e dal piano di test, grado di automazione del testing (che sia ripetibile), dalla copertura con Emma, calcolo delle metriche, ecc..

# Modalità di valutazione (2)

- ***Modalità di sviluppo e documentazione*** **20%**
  - *Valutazione della modellazione agile*: uso di UML per la modellazione dei casi d'uso (requisiti funzionali), per il design dell'architettura (diagramma delle componenti e di deployment), diagramma delle classi, piani di test, uso di metodi formali (Reti di Petri, Abstract State Machines) per specificare il comportamento delle componenti più "critiche", ecc..
  - *Valutazione dello sviluppo agile*: uso dei metodi (testing, copertura, refactoring, ecc..), strumenti (tool e Plug-in Eclipse) e tecnologie (librerie, APIs, ecc..) spiegati a lezione e nei tutorial
  - *Valutazione della documentazione* a corredo

- ***Presenza attiva in aula e lab*** (svolgimento degli esercizi proposti) **10%**

# Progettare in “piccolo” ed in “grande” con una metodologia agile

Introduzione

# Metodologia agile

Differenziazione di metodi e modelli di sviluppo:

- **Metodologie pesanti** per i vecchi metodi come il *Modello a cascata*
- **Metodologie iterative** per i metodi come il *Modello a spirale*
- **Metodologie agili** (o leggere) che coinvolgono quanto più possibile il committende, ottenendo in tal modo una elevata reattività alle sue richieste
  - *Agile alliance*, una organizzazione no-profit creata allo scopo di diffonderle
  - Esistono un certo numero di tali metodologie (XP, SCRUM, AMDD,ecc..)
    - Adotteremo **AMDD = versione "agile" del Model Driven Development**

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Dialogo con l’utente che espone le sue necessità, interpretazione di massima delle specifiche e primo documento di analisi.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Definizione delle specifiche per mezzo di tecniche formali. Stesura di un documento di analisi dei requisiti da sottoporre all’utente.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

**Progettazione in grande:**
Si analizza la suddivisione del software per componenti funzionali e lo scambio dati tra questi ultimi.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

**Progettazione in piccolo**: Per ogni componente si fa un’analisi di dettaglio e si definisce l’algoritmo.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Scelta del linguaggio di programmazione e sviluppo delle singole parti con azione di correzione degli errori (prototipi).

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Di norma un team diverso da quello di progetto. A questo punto va fatta anche l’integrazione tra le parti.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle pr*
   2. *Documentazione*
   3. *Manutenzione*

Può essere scritta anche durante lo sviluppo, man mano che si completa il software.

# Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle pr*
   2. *Documentazione*
   3. *Manutenzione*

Effettuata dopo la distribuzione del prodotto, con l’aiuto degli utenti: nuove release e/o versioni.

# The infamous (software) design/development process tree-swing comic