Introduzione corso di Ingegneria del Software

Ingegneria del Software

Francesca Arcelli Fontana

Università degli Studi di Milano-Bicocca
DISCo – Dipartimento di Informatica Sistemistica e Comunicazione
Milan, Italy

27, Settembre 2021

AA 2021/2022

Organizzazione del corso

- Lezioni ed esercitazioni :

  Francesca Arcelli e Gabriella Nifosi

  Lunedi': 13.30-15.30

  Mercoledi':12.30-14.30

- Esercitazioni saranno al lunedi ed iniziano 11 ottobre (prima dell'11-10 al lunedi ci sarà lezione)

- Laboratorio: Ilaria Pigazzini- date comunicate su Moodle
  - Giovedì: 8.30-11.30

NOTA:

- I **laboratori** saranno al GIOVEDI', ma non tutti i giovedì: controllate le date su Moodle.
- Inizieranno dopo il 12 ottobre.

Lezioni prossime due settimane

- Lunedi 27-9:introduzione corso
- Mercoledi 29-9: ripasso autonomo diagrammi UML con materiale che verrà indicato ad inizio lezione via webex.
- Lunedi 4-10: lezione
- Mercoledi 6-10: lezione
- Lunedi 11-10: inizio esercitazioni (tutti i lunedi fino al 20-12)
- Mercoledi 13-10: lezione via webex (le successive lezioni, salvo problemi, saranno tutte in presenza).
- Da giovedi 21-10 iniziano i Lab in presenza.

Materiale didattico

- Il materiale didattico sarà disponibile nella piattaforma di e-learning
- Corso di Ingegneria del Software, III anno, LT
  - http://informatica.elearning.unimib.it/

Contenuti del corso (I)

- Aree dell'ingegneria del software: model-driven software engineering, component-based software engineering, aspect-oriented software engineering, distributed software engineering,….
- Architetture software: Progettazione di architetture software. Pattern architetturali. Diagramma dei package, dei componenti e di deployment.
- Applicazione dei design pattern nello sviluppo del software. Testo del Larman e slide
  - Modellazione di un framework di persistenza con i design pattern. - Testo del Larman
  - Sviluppo di un sistema di gestione di risorse con i design pattern. Testo del Larman

Contenuti del corso

- Principi, tecniche e strumenti per lo sviluppo di software
- Pattern ed esempi della loro applicazione nello sviluppo del software: Design Pattern e Pattern Architetturali
- Continuous Integration pipeline
- Problemi architetturali e loro soluzione
- Best practices in Java -Lab
- Valutazione della qualità del codice
- Nozioni base per la gestione dei progetti
- Esempi progetti software,le problematiche affrontate
- Utilizzo di GitHub in Lab.
- Utilizzo d Sonarqube e Understand in Lab.

Obiettivi del corso

- Acquisire conoscenze più avanzate di sviluppo di un progetto software
- Conoscere ed applicare i pattern durante lo sviluppo del software
- Imparare best practices in Java
- Imparare i principali design principles
- Valutare la qualità del software
- Problematiche del refactoring di progetti di medie/grandi dimensioni
- Pianificare la gestione delle risorse di un progetto: diagrammi Gantt e PERT. Gestione dei rischi, gestione della qualitá.

Alcune tematiche dei LAB

- Java Best Practices
- Controllo di versione con Git
- Verifica della qualità del codice con SonarQube
- Impostazione complessiva di un progetto software: controllo di versione, collaborazione, build, definizione e verifica degli standard di qualità
- Detection Code Smells e valutazione qualità del software con Understand
- GitHub and SonarQube: *we can empower and integrate each other inside a pipeline for a continuous integration process*. --DevOps
- …Travis8 (tool): is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.
- SonarCloud9 (tool): *is the cloud service offered by SonarSource and based on SonarQube; it allows to host the SonarQube analysis of a project by importing the code directly from Github*.

Testi di riferimento del corso

- I. Sommerville, Ingegneria del Software, Pearson, 9 ed, 2011 (solo alcuni contenuti).
- C. Larman, Applicare UML e i Pattern – analisi e progettazione orientata agli oggetti, Pearson, 3 ed, 2005- solo ultimi capitoli
- Joshua Bloch, Effective Java: Programming language Guide, Addison Wesley 2001.
- **Lippert, Refactorings in Large Software Projects-How to Successfully Execute Complex Restructurings**

- Articoli o capitoli di libri.

Esame

- 1 Task assegnato in aula: presentazione di un Pattern Architetturale
- Valutazione dell'attività di Laboratorio: 0-4 o 0—5 punti.
- Esame finale: Progetto assegnato ad un gruppo di 3 / 4 studenti. Circa 4/6 settimane di tempo per realizzarlo.
- Per il progetto si farà uso di Git, Sonarqube,...
- Proposte di stage interni ed esterni.

Interventi esterni

- Seminari esterni con <u>obbligo di frequenza</u>

Scorso anno:

- Ing. Gualtiero Bazana, ALTEN

  Presidente mondiale ISTQB (Inter.Software Testing Qualification Board)

  «Analisi quantitativa dell'efficacia delle varie tecniche di software engineering»

- …altri interventi Camillo Rui di Clusit, e di Cybaze
- Presentazione di **stage/tesi di studenti** che si sono laureati recentemente:

Riconoscimento di pattern architetturali, di nuovi architectural smells, migrazione verso micro servizi..

Why Software Engineering?

From….History

Software Crisis [Context]

- **Initially**, the focus on the development and innovation of hardware
- Computer programming was considered as a state of art known to a gifted few
- Programmers followed their own personal style and did not follow any standardized approach
- Design was implicitly performed in one's mind, and documentation was often non-existent

Software Crisis [Statistics]

- A study by the Controller General of the United States **(1979)** found that:
  - 2% of software worked on delivery
  - 3% worked only after some corrections
  - 45% were delivered, but never successfully used
  - 20% were used, only after major modification/rework
  - 30% were paid for, but never completed/delivered

Software Crisis [Statistics]

- In a survey done by IBM in **1994**, it was reported that
  - 55% of systems cost exceeded the initial estimated budget
  - 68% of software overran delivery schedules
  - 88% of the software had to be substantially modified
- Failed projects are not used:
  - 75% of large projects are operating failures or not used at all
- Public safety can be affected:
  - Sw controls transportations, life support systems, nuclear plants

Software Crisis [Why?]

- Critical problems of software development – challenges of large software:
  - Distributed and real time systems
    - Distributed software is complex and has more paths of failure
  - Human comprehension
    - No single person can completely comprehend a large project
  - Growth in hardware complexity
    - Hw complexity doubles every 18 months
  - Lack of standardization
    - Interfaces and methods for sw construction can differ
  - Lack of measurment
    - Time and cost are hard to estimate (need of experience)

Software Crisis [Why?]

- Critical problems of software development - absence of a structured and methodical approach caused problems like:
- Errors made in one phase of development carried over to subsequent phases, causing rework and therefore delay
- Difficulty in estimating effort and measuring progress caused schedule slippage and budget overruns
- Assuring quality of delivered software in terms of reliability and meeting the customers' requirements became difficult

Software Engineering

- First defined in 1968 by the NATO Science Committee
  - 50 sw experts meet to solve the sw crisis
  - First definition of software engineering:

The application of a

- systematic: Establish standard practices
  Allows reapeatable results
- quantifiable Allows estimation of time, cost

Allows comparison of systems

Approach to the development, operation, and maintenance of software.

Distributed Software Engineering

Component-based Software Engineering

Model-driven Software Engineering

Software Engineering

Aspect-Oriented Software Engineering

Reverse Software Engineering

Empirical Software Engineering

Software development

- Problems
- Solutions

Symptoms of Software Development Problems

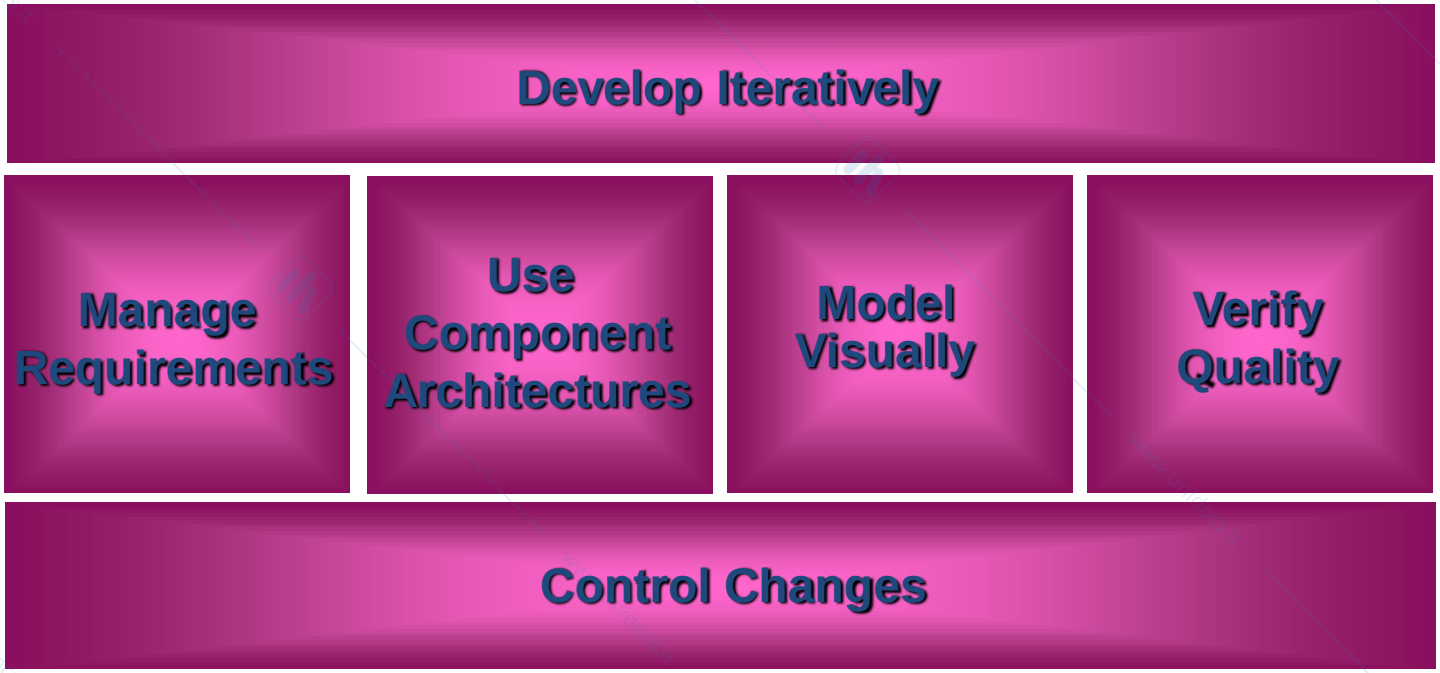
- Inaccurate understanding of end-user needs
- Inability to deal with changing requirements
- Modules that don't fit together (integration)
- Software that's hard to maintain or extend (brittle)
- Late discovery of serious project flaws (integration)
- Poor software quality (architecture, risks unanticipated…)
- Process not responsive to Change (Gantt Charts…)
- Unacceptable software performance
- Team members in each other's way, unable to reconstruct who changed what, when, where, why (software architecture, …
- …and we could go on and on…

Need a Better Hammer!

- We need a process that
  - Will serve as a framework for large scale and small projects
  - ➔ Adaptive – embraces 'change!'
    - Opportunity for improvement not identification of failure!
  - Iterative (small, incremental 'deliverables')
  - Risk-driven (identify / resolve risks up front)
  - Flexible, customizable process (not a burden; adaptive to projects)
  - Architecture-centric (breaks components into 'layers' or common areas of responsibility…)
  - **Heavy** user involvement

- Identify best ways of doing things – a better process – acknowledged by world leaders…

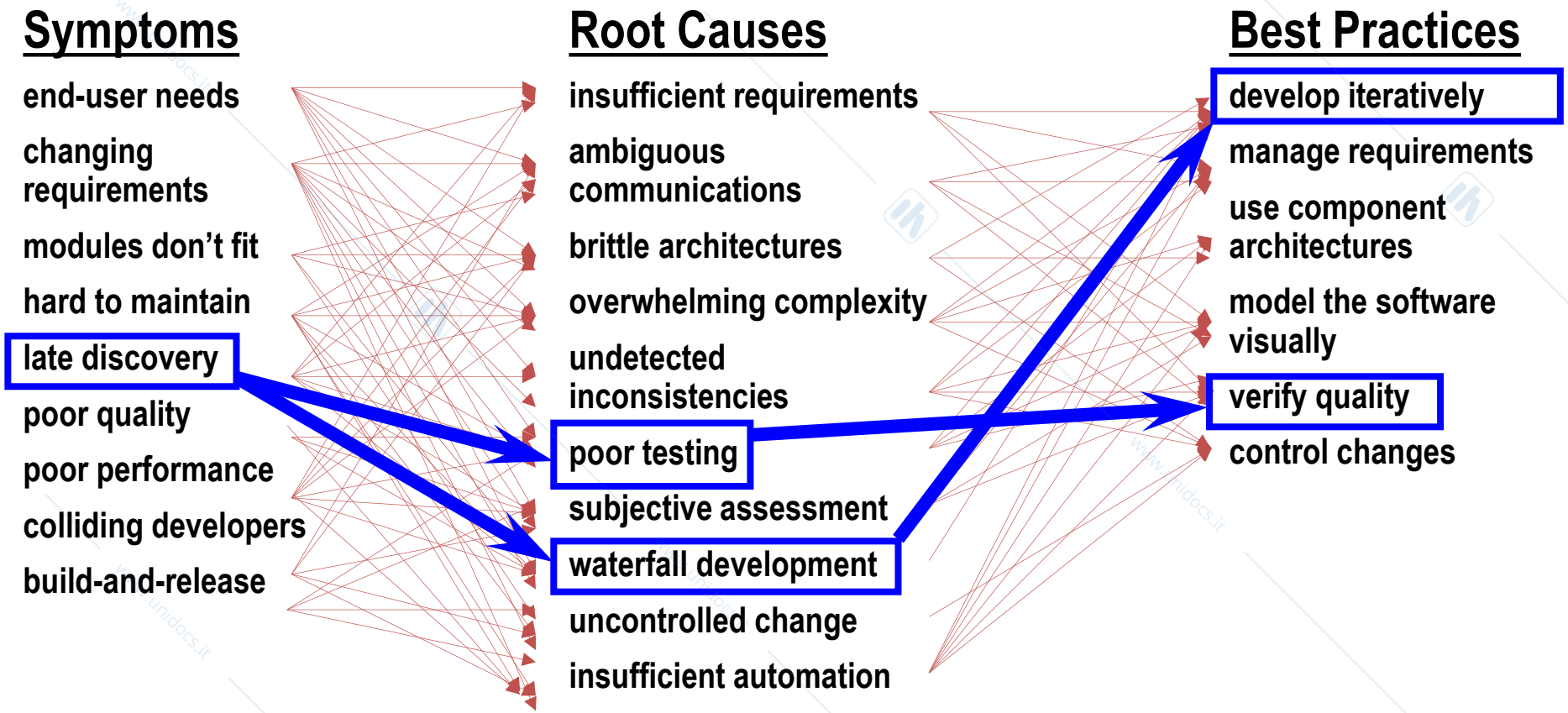
Best Practices of Software Engineering



Know these!

Which is most significant one?

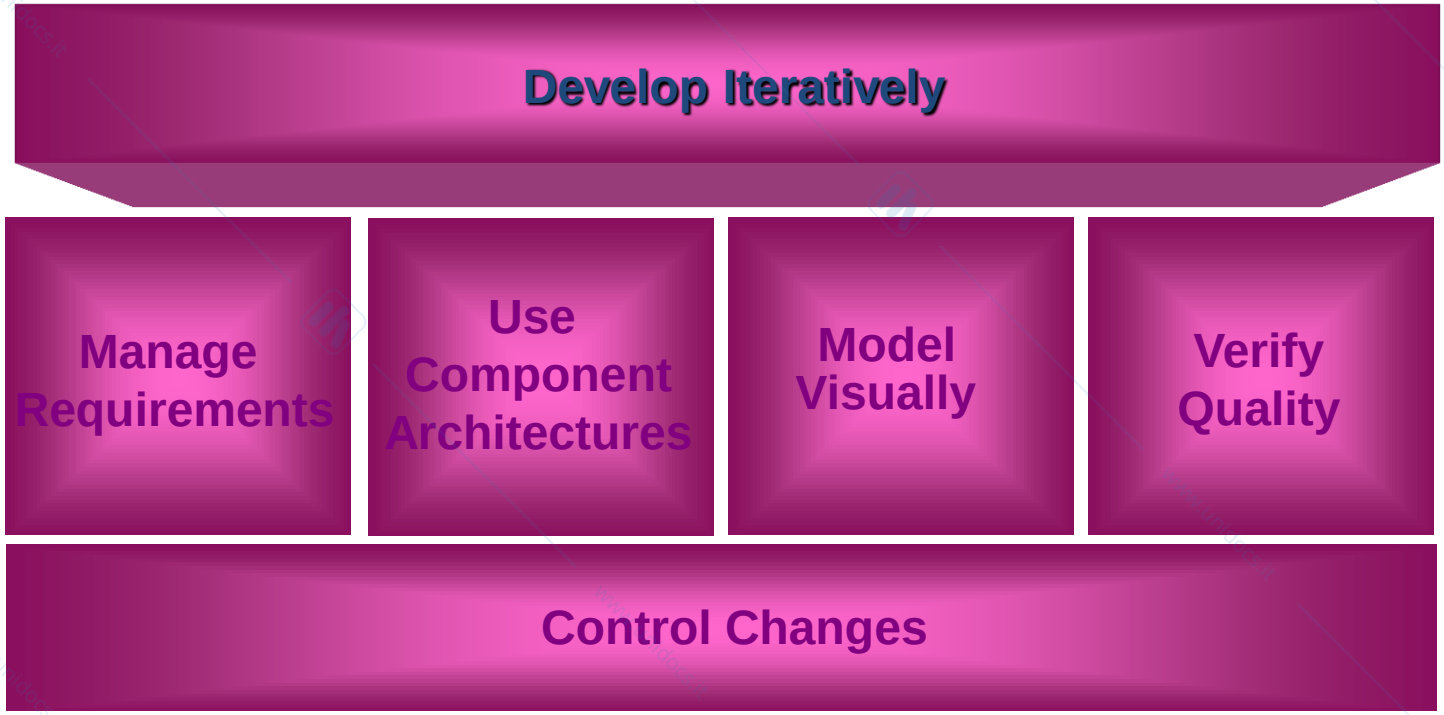
Addressing Root Causes Eliminates the Symptoms



Symptoms of problems can be traced to having Root Causes.
Best Practices are 'practices' designed to address the root causes of software problems.

30

Practice 1: Develop Software Iteratively



Considered by many practitioners to be the most significant of the six