INGEGNERIA DEI SISTEMI SOFTWARE
M - 72939

a.a. 2019-2020

Antonio Natali - antonio.natali@unibo.it

https://github.com/anatali/iss2020LabBo

Link a lezioni

INGEGNERIA

Da https://it.wikipedia.org/wiki/Ingegneria:

- L'**ingegneria** è la disciplina, a forte connotazione tecnico-scientifica, che ha come obiettivo l'applicazione di conoscenze e risultati propri delle scienze matematiche, fisiche e naturali per produrre sistemi e soluzioni in grado di soddisfare esigenze tecniche e materiali della società attraverso le fasi della progettazione, realizzazione e gestione degli stessi.

Ingegneria del software

Da https://it.wikipedia.org/wiki/Ingegneria_del_software

- disciplina che si occupa dei processi produttivi e delle metodologie di sviluppo finalizzate alla realizzazione di sistemi software.
- si propone una serie di obiettivi legati all'evoluzione dello sviluppo del software (inteso come attività industriale) sia da un punto di vista tecnologico (per esempio attraverso la definizione di nuovi linguaggi di programmazione) che metodologico (per esempio il perfezionamento dei modelli di ciclo di vita del software).

INGEGNERE

- Realizza progetti
  - Da cui si realizzano prodotti
  - Partendo da una analisi del problema dato
  - Definito da un insieme di requisiti

PROBLEMA

Difficoltà che richiede un adattamento o un comportamento particolare, o di cui si impone il superamento.

PROBLEMATICA

Complesso dei temi presi in considerazione in rapporto a determinati rami del sapere o a determinati interessi.

INGEGNERE DEL SOFTWARE

MAIN GOAL: Implement a feature that matches specification and works efficiently, but at minimum development and maintenance cost, using minimum effort and in minimum amount of time (and optionally get maximum payment for doing so)

Per noi, un INGEGNERE che fa suo il motto:

1. there is no code without a project,
2. no project without problem analysis , and
3. no problem without requirements

See also:

https://www.quora.com/As-a-programmer-what-is-your-favorite-motto

OUR GOALS

- (Learn to) design, build (deploy/maintain) **software systems …**
- made of software **components** …
- that **interact** in a local environment and/or via Internet …
- whose **behavior** is expressed in some **programming language** (Java, Kotlin, Python, JavaScript/Node, C, C++, …) …
- with the aim to solve problems in application domains (e.g. IOT) that demands distributed, heterogeneous systems based on (micro)services and Domain Driven Design
- by following a '**hands on**' approach (learning by doing/by example)
- working **in teams** of (3 person each)
- according to **agile**, **incremental** and **model-driven** development

Software systems

- Not ‘simply’ algorithms , but … ???

- … we will discuss and ‘discover’

Software component

- At language level:
  - Function
  - Object
  - Coroutine
  - Actor
  - Agent
  - …
- At architecture (layered, subsumption, hexagonal …) level:
  - Views – Models – Controllers / DAO(DataAccesObjects)
  - (Micro)Services
  - Plugins
  - API
  - …

Component Interaction

- Produre-call
- Message-passing (local or remote)
- Synchronous / Asynchronous
- …

- Request-response (HTTP – REST)
- CoAP
- TCP / UDP / …
- …

Programming language

- Looking at … :
  - https://en.wikipedia.org/wiki/List_of_programming_languages
  - https://en.wikipedia.org/wiki/List_of_programming_languages_by_type
  - https://en.wikipedia.org/wiki/Comparison_of_programming_languages
  - https://en.wikipedia.org/wiki/History_of_programming_languages
- … we could get lost

Towards technology-independence (being technology-aware)

- means not being biased towards any particular platform or software language.
- It's about selecting the right technology that fits the solution, not fitting the solution around the technology.

Distributed systems

- A *distributed system* is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.
- The components interact with one another in order to achieve a common goal.
- ….

Domain Driven Design

- an approach to software development for complex needs by connecting the implementation to an evolving model (A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain).

Agile software development

- Requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).
- **Scrum** è un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo.
- https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Italian.pdf

Model Driven Software Development

- The approach centers itself on building models of a software system.

MODEL

- A model is an abstraction of the system being studied from different perspectives: Structure, Interaction, Behavior (External)
- https://www.martinfowler.com/bliki/ModelDrivenSoftwareDevelopment.html
- https://en.wikipedia.org/wiki/Model-driven_engineering

Activities

- Il **laboratorio** costituisce una parte essenziale del corso e la frequenza è pressochè indispensabile.
- Gli studenti sono tenuti a progettare e costruire in modo incrementale sistemi software, che verranno valutati insi eme al docente, in modo da acquisire una retroazione immediata sul lavoro svolto (**autovalutazione**) , che potrà essere utilizzata per modificare/migliorare quanto sviluppato.
- La **valutazione finale** consiste nelle presentazione e discussione degli artefatti prodotti in relazione alla costruzione di un sistema software (dello stesso tipo di quello discusso nei CaseStudy proposti durante il corso) che rispetti i requisiti pubblicati l'ultima settimana di lezione.
- Questi artefatti possono essere prodotti in modo individuale oppure, preferibilmente, attraverso un **lavoro cooperativo** svolto in gruppi di 2/3 studenti.

Application0

- Design and build a software system to control a DDR (differential drive robot) so that:
  - The robot is able to explore in a sistematic and autonomus way a room
  - By showing (in real time) a picture of the explored space

Teaching and Assessment methods

The examination will performed in two-phases.

- The first phase starts by publishing a set of requirements and ends with the production of a prototype of a software system satisfying the requirements, toghether with a project site. This phase can be delevoped both in individual way or in a team.
- The second phase consists of a individual discussion of the work, in which the student is invited to present the relevant project choices with reference to the theroretical/practical concepts learned during the course.

ESEMPI

TemaFinale1 - TemaFinale2 - TemaFinale3

WORFLOW

- Beyond 'traditional' programming models: from Java to Kotlin and Node
- Towards heterogenous distributed systems by means of DDR Robots based on Arduino and Raspberry
- Software design and development : from bottom-up to top-down

---

- From problems to projects: the role of requirement/problem analysis and of test planning. The need of MDSD.
- The issue of distribution: need of a methodology (design patterns), of proper tools/frameworks and DDD.
- From 'programs' to Software Architectures based on (micro)services: the need of 'technoloy independence' (e.g. from network protocols UDP,TCP,HTTP,CoAP, framerworks, etc. …)

- Beyond UML: introduction of a custom metamodel / language
- Using executable models for agile, incremental software development
- Towards automatic code generation: software factories

WORK TO DO

- Acquire a paper-notebook and a pencil
- Fill our workflow template with your data/photo and print it on a SINGLE SHEET
- Download and install tools reported in LabBo2020StartUp
- Attempt to fill (at your best) the template with reference to Application0 and print it on a SINGLE SHEET

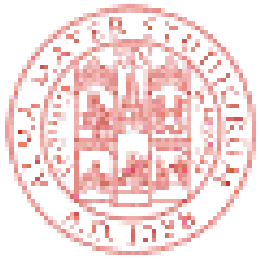
-

- Evaluate the possibility to build a DDR
- Read
  - https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Italian.pdf

Domande (su un sistema)

- Cosa deve fare → *committente*
- Cosa deve essere → *analista*
  - Quale ***struttura-interazione-comportamento*** **sono necessari** tenendo conto dei ***vincoli che sorgono dai requisiti e dal problema in sè***
- Cosa è opportuno che sia → *analista*
  - Tenendo conto delle tecnologie, dell'andamento del mercato, delle risorse umane, economiche, temporali, etc.
- *Come è fatto* → *progettista*
  - Quale ***struttura-interazione-comportamento*** ha il sistema finale tenendo conto dei vincoli dall'analisi e dei criteri (*pattern*) usati per risolvere le forze (anche contrastanti) in gioco

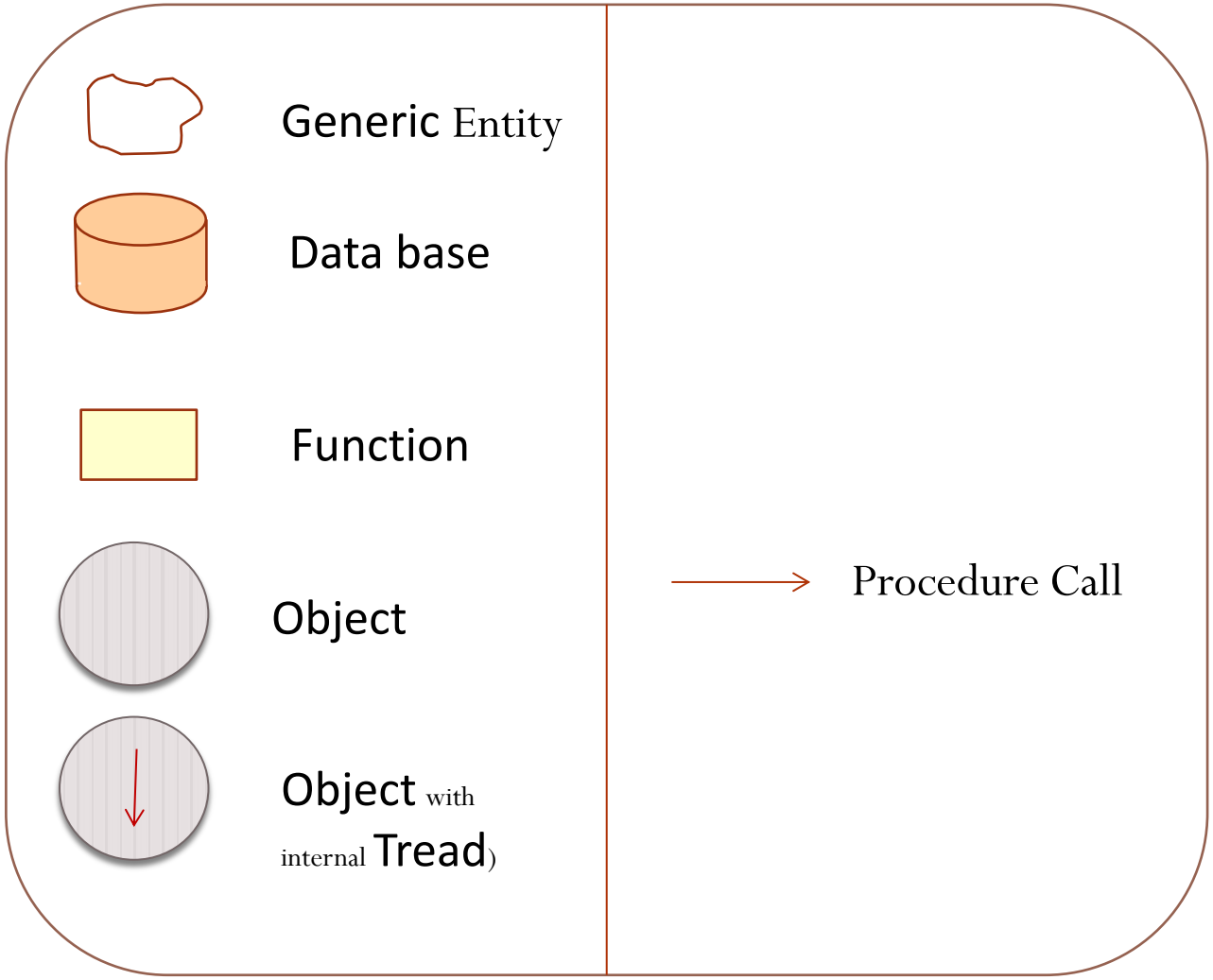
Workflow

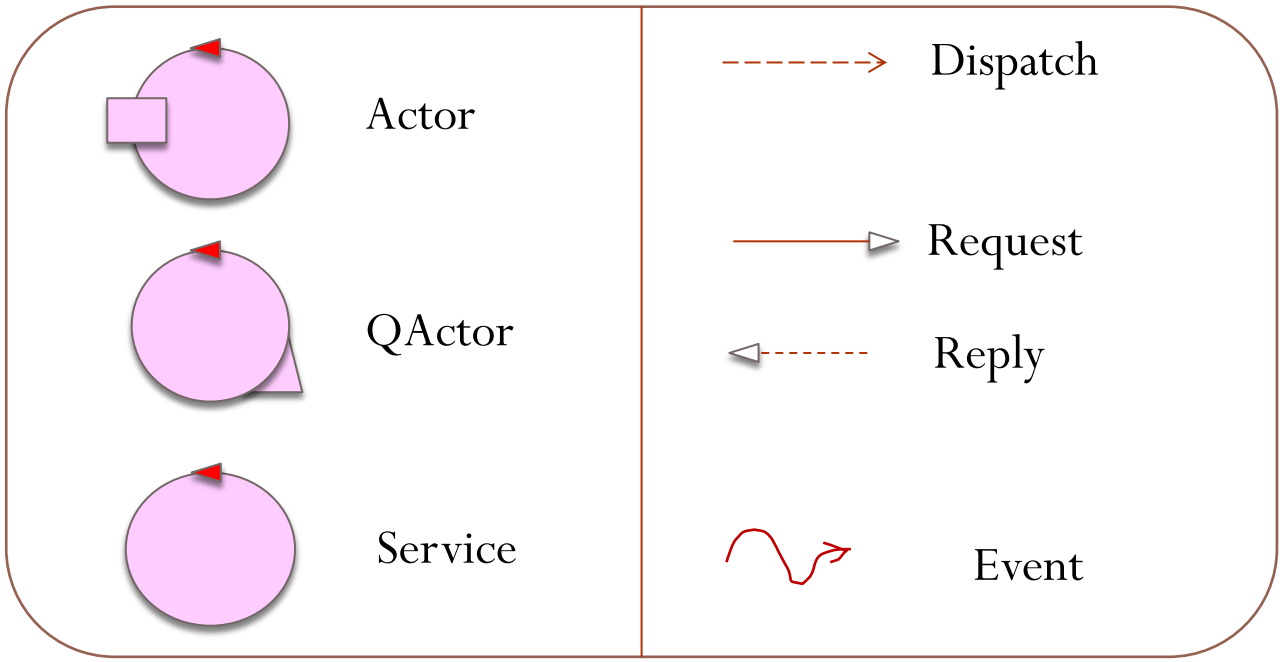


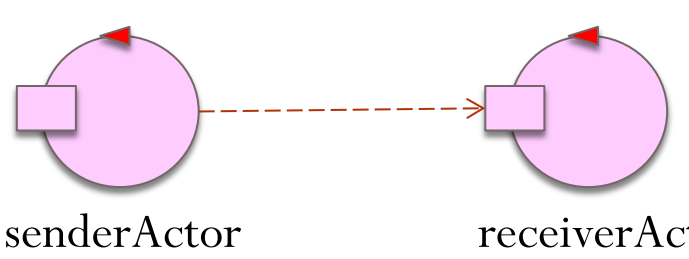
- Sviluppiamo l'analisi dei requisiti e l'analisi del problema
  - *Esprimendo fatti rilevanti attraverso modelli essenziali*
- Discutiamo in modo sistematico avvalendoci di modelli basati su meta-modelli custom
  - *Usiamo i modelli come se fossero il nuovo codice sorgente* costruendo generatori di codice usando Xtext (pg. 206)
  - Realizziamo in modo automatico la schematic part avvelendoci dei design pattern per sistemi distribuiti
- *Impostiamo piani di collaudo ancor prima di avere iniziato la fase di progettazione*
- Realizziamo un primo prototipo di prodotto e interagiamo con il committente

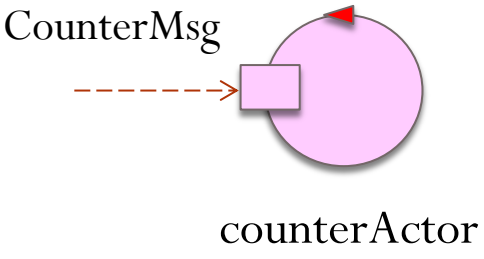
Non c'è … senza analisi

- NON iniziamo la fase di progetto prima di avere assestato la fase di *analisi dei requisiti e l'analisi del problema*, con relativa peer-review, al fine di
  - Individuare i principali sottosistemi
  - Capire quali tecnologie sono necessarie
  - Valutare i punti critici, i rischi e i costi
  - Pianificare l'uso delle risorse e i tempi
  - Distribuire il lavoro tra le persone


Generic Entity
Data base
Function
Object
Object with internal Tread)
Procedure Call


Actor
QActor
Service
Dispatch
Request
Reply
Event


senderActor
receiverActor


CounterMsg
counterActor