INGEGNERIA DEL SOFTWARE: ORIGINE E DEFINIZIONI

- Seconda metà anni ’60 → Primi tentativi di grandi progetti software (sistemi operativi, e.g. IBM 360)
  - Si evidenziano le problematiche di gestione di questi grossi progetti → “crisi del software”→ troppo budget, ritardi di consegna
  - **Coniato il termine “ingegneria del software”** → ci si accorge che il metodo di implementazione di un prodotto software complesso doveva essere analogo a quello degli altri prodotti dell’ingegneria

- L’applicazione di conoscenze scientifiche e tecnologiche, metodi ed esperienza al progetto, l’implementazione, il collaudo e la documentazione del software.

Systems and software engineering – Vocabulary

- L’applicazione di un approccio sistematico, disciplinato, quantificabile, allo sviluppo, all’operazione ed alla manutenzione del software

IEEE Standard 610-12-1990

- Disciplina tecnologica e manageriale che riguarda la produzione sistematica e la manutenzione dei prodotti software,. . . sviluppati e modificati entro i tempi e i costi preventivati”

R. Fairley. Software Engineering Concepts. Maw-Hill, 1985.

PROGRAMMAZIONE vs INGEGNERIA DEL SOFTWARE

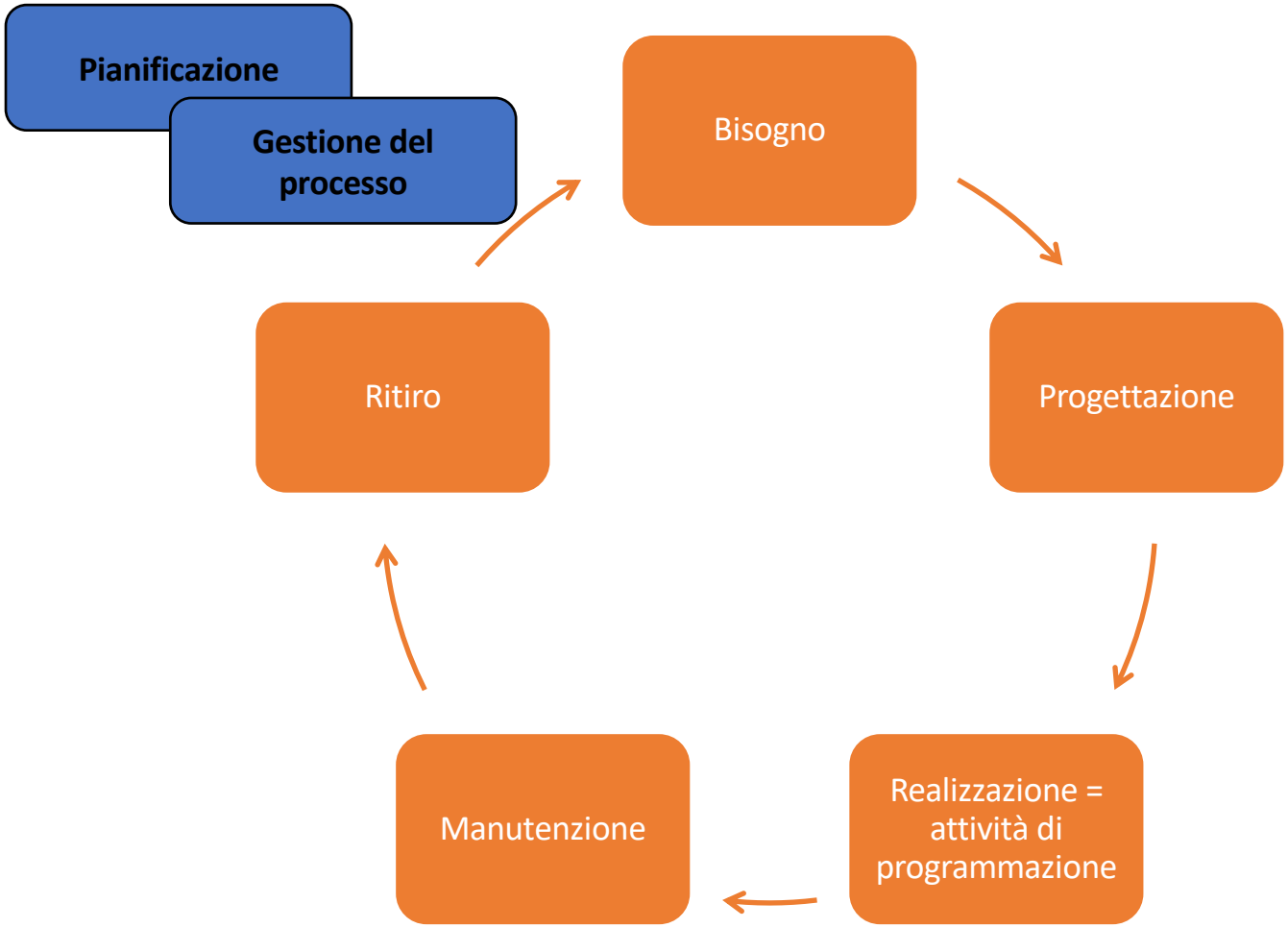
PROGRAMMAZIONE

- Soluzione stand alone
- Programma completo
- Monosviluppatore
- L'attività principale è lo sviluppo
- Competenze: linguaggi e programmazione

INGEGNERIA DEL SOFTWARE

- Sistemi complessi
- Componenti e/o moduli
- Multisviluppatore
- L'attività principale è la progettazione
- Competenze: approcci di progetto, gestione dei processi, definizione delle specifiche, comunicazione e interazione (con il committente/utente e con lo sviluppatore)

LE MACROFASI DEL CICLO DI VITA



ATTIVITÀ DI PRODUZIONE

- **Studio di fattibilità**
- **Acquisizione, analisi e specifica dei requisiti**
- **Progettazione dell'architettura**
- **Codifica e test dei moduli**
- **Integrazione e test del sistema**
- **Rilascio, installazione e manutenzione**
- **Attività di supporto**

OGNI ATTIVITÀ PRODUCE UN OUTPUT

STUDIO DI FATTIBILITÀ

- Stabilisce se:
  - Il prodotto può essere realizzato
  - È conveniente realizzarlo
  - Quali strategie possono essere adottate per realizzarlo
  - Valutare risorse/costi delle diverse alternative
- 3 parti:
  - Definizione del problema e analisi di dominio quanto più approfondito possibile
  - Definizione degli scenari di soluzione
  - Modalità di sviluppo delle alternative, costi e date di consegna

OUTPUT =
Documento di studio di fattibilità
che deve contenere la descrizione delle 3 parti

ACQUISIZIONE, ANALISI E SPECIFICA DEI REQUISITI

- REQUISITI = definizione di COSA l'utente richiede al software (proprietà e comportamenti richiesti)
- SPECIFICA DEI REQUISITI = descrizione precisa dei requisiti →
  - Descrive che cosa deve fare un sistema e quali proprietà deve avere
  - NON descrive come deve essere costruito
- Attività critica da cui dipende la correttezza/successo del sistema
- Deve tener conto dei vincoli imposti da altri sistemi cooperanti
- Devono essere identificati tutti gli stakeholder
- Devono essere analizzati i diversi punti di vista
- Devono essere armonizzate le contraddizioni

OUTPUT =
Documento di specifica dei requisiti, Versione preliminare del manuale utente, Piano di test

TIPOLOGIE DI REQUISITI

Requisiti dell'utente: descrizione del dominio applicativo e obiettivi dell'implementazione

- Quali utenti?
- Quali aspettative?
- Quali entità? Quali relazioni? (dati e relazioni tra i dati)
- Come interagiscono i dati col sistema? (controllo)

Requisiti funzionali

- Cosa farà il sistema?

Requisiti non funzionali

- Attributi di qualità (efficienza, scalabilità, etc)

Requisiti del processo di sviluppo e manutenzione

- Procedure di test e e verifica
- Priorità di sviluppo delle funzioni richieste
- Possibili cambiamenti che il sistema subirà

PROGETTAZIONE DELL'ARCHITETTURA

- Stabilisce **COME** deve essere fatto il sistema
- Costruzione di un modello = descrizione astratta di un sistema
- Descrive le componenti del sistema, le interfacce e le relazioni tra le parti
- Si parte da un'architettura di alto livello per poi scendere a livelli di dettaglio successivi
- Esistono linguaggi formali (ad es UML)

OUTPUT =
Documento di specifica del progetto

CODIFICA E TEST DEI MODULI

- Fase di sviluppo:
  - Trasparenza
  - Leggibilità (si possono seguire standard anche aziendali)
  - Modificabilità (è bene usare sistemi di versioning)
- Si effettua il debug
- Si effettua il test dei moduli
- Si può effettuare una verifica del codice per vedere se è conforme allo standard

OUTPUT =
Codice e documentazione del codice

INTEGRAZIONE E TEST DEL SISTEMA

- Attività in cui il sistema viene assemblato → a volte è inclusa nella precedente
- Vengono effettuati test di integrazione e test di sistema (alpha test)

OUTPUT =
Sistema completo per il rilascio

RILASCIO, INSTALLAZIONE E MANUTENZIONE

- Fase post-sviluppo
- Rilascio = consegna del prodotto
  - Ad un gruppo ristretto di utenti → beta test
  - A tutti gli utenti previsti
- Installazione = setup dell'architettura run-time
- Manutenzione (circa 60% del costo totale)
  - Correttiva
  - Adattiva
  - Perfettiva→ quella che si stima consumare le maggiori risorse (50% delle risorse di manutenzione)

REQUISITI FUNZIONALI E TESTING: ESEMPIO

Table 1. Use Case test for creating a new de-identified clinical form in WBB made by a doctor.

Use-Case: create a new de-identified clinical form.
Actors: Doctor (D), WebBioBank System (WBB).
Description: A doctor fills a de-identified clinical form into the system.
Precondition: the doctor must be authenticated and authorized by the system, with the IDBAC list of the right OU opened.
Workflow steps: 1. D: uploads the local xml to pair IDBACs to the local system patients; 2. WBB: visualize the name, surname and date of birth from IDBAC found into the xml; 3. D: selects the right IDBAC 4. WBB: shows the patient page with all the forms and the acquisition data; 5. D: clicks the form to fill; 6. WBB: shows the right blank form; 7. D: fills the form and clicks "save"; 8. WBB: display the "sign" option; 9. D: the doctor click the sign option, finalizing the form;
Extensions: 3a or 6a. D: selects the wrong IDBAC/Form; WBB: shows the wrong IDBAC/Form D: clicks on the "Go Back" button; WBB: return to step 1 (IDBAC de-identified); 9a. D: the doctor does not sign the document; WBB: shows the IDBAC list on step 1 and it displays +1 draft in the inbox;

Table 2. Use Case test results for Table 1.

Expected results: 1. Every System response (WBB) must be carried out correctly. 2. No IDBAC with personal information must remain cached in the grid form except between steps 2 and 3. a. Every personal detail must be only retained on the local database file, no browser must retain this information; b. This data cannot leave the local machine. 3. Signed form cannot be deleted or modified. 4. Saved but not signed forms can be modified.
Results: 1. Every System step was fulfilled, to test this scenario we used UPDRSIII and UDysRS forms; 2a. the system was tested with Google Chrome (ver. 62.0.3202) and Internet Explorer 11 as browsers, no information were visualized on all the other steps except 2,3; 2b. Wireshark (ver. 2.0.3) was used to test the internet/WLAN traffic during this test, no personal data was exchanged outside the local machine; 3. it's not possible to modify or delete the form, also, the "save" and the "sign" operation are logged with the system timestamp (server-side); 4. Saved forms can be modified but not deleted without Administrator's rights.

QUALITÀ DEL SOFTWARE: CLASSIFICAZIONI (1)

QUALITÀ INTERNE

- Relative alla stesura del codice
- Percepibili dagli esperti di programmazione (sviluppatori)

QUALITÀ ESTERNE

- Relative al prodotto finale (interfaccia esterna)
- Percepibili dagli utenti

Strettamente connesse tra di loro e interdipendenti

QUALITÀ DEL SOFTWARE: CLASSIFICAZIONI (2)

QUALITÀ DI PROCESSO

- Relative al processo di sviluppo (documentazione del processo)
- Percepibili dagli esperti di ingegneria del software

QUALITÀ DI PRODOTTO

- Relative all'artefatto (anche prodotto intermedio)
- Percepibili dal committente

PROPRIETÀ DEL SOFTWARE

	INTERNO	ESTERNO	PRODOTTO	PROCESSO
CORRETTEZZA	X		X	X
AFFIDABILITÀ			X	
ROBUSTEZZA	X		X	
PRESTAZIONI			X	
SCALABILITÀ			X	
EFFICIENZA	X		X	X
USABILITY	X		X	
VERIFICABILITÀ	X		X	X
MANUTENIBILITÀ	X		X	
RIUSABILITÀ		X	X	
PORTABILITÀ		X	X	
COMPRENSIBILITÀ			X	X
INTEROPERABILITÀ			X	
PRODUTTIVITÀ				X
TEMPESTIVITÀ				X
TRASPARENZA		X	X	X

CORRETTEZZA

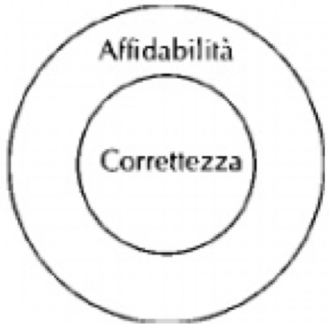
- Capacità del software di rispondere ai requisiti funzionali e non funzionali →

  a fronte di input corretti, il software produce il risultato atteso

- Dipende dal livello di definizione delle specifiche
- Le specifiche devono anche definire quando un input è corretto (*precondizioni*)
- Può essere valutata nella fase di test del prodotto → è uno degli obiettivi fondamentali della valutazione

AFFIDABILITÀ

- Probabilità che il software si comporti nel modo atteso in un certo intervallo di tempo
- Qualità “relativa” → se la conseguenza di un errore non è grave, anche un software non corretto può essere considerato affidabile
- Software corretti sono anche affidabili, non viceversa



- Molti software sono rilasciati con bug già noti

ROBUSTEZZA

- Capacità del software di funzionare anche in situazioni anomale non previste dai requisiti

- Caso classico→ errore dati in input
- Livelli di robustezza:
  - livello 0: anomalia non rilevata. Il software continua l'esecuzione e produce risultati errati/entra in cicli infiniti;
  - livello 1: anomalia rilevata. L'esecuzione del programma viene interrotta immediatamente;
  - livello 2: anomalia rilevata. Viene eseguito del codice opportuno per risolvere il problema/ presentare diagnostica.
- Il livello di robustezza dipende dalla capacità di prevedere e gestire le **eccezioni**.

USABILITY

- Facilità di utilizzo e di apprendimento da parte dell'utente

- Basata sulla valutazione di
  - Interfaccia utente
  - Facilità di configurazione
  - Capacità di adattamento al running environment
- È una misura soggettiva
- Deve essere specificato il tipo di utente

MANUTENIBILITÀ

EVOLVIBILITÀ

- Il software deve essere duttile
- Il software deve poter fornire nuove funzioni/modificare vecchie funzioni → le modifiche vanno progettate e documentate
- All'aumentare del tempo dal rilascio la capacità di evoluzione richiede interventi via via più complessi
- Facilitata dalla modularizzazione

RIPARABILITÀ

- I difetti del software devono essere corretti con una quantità di lavoro ragionevole
- Le parti "soggette a usura" devono essere accessibili (RAS: repairability, availability, serviceability)
- È utile l'utilizzo di parti standard

INTEROPERABILITÀ

- Capacità del software di coesistere e cooperare con altri sistemi

- Facilitato dall'utilizzo di interfacce aperte
- Sistema aperto: collezione estendibile di applicazioni sviluppate in modo indipendente ma che funzionano come un sistema integrato

INTEROPERABILITÀ E SANITÀ

INTEROPERABILITY =
Ability of different systems to work cooperatively allowing different users to share information and resources

REQUISITI NON FUNZIONALI E TESTING

Table 1 Non-functional requirements and evaluation scenarios

NON-FUNCTIONAL REQUIREMENT	DEFINITION	EVALUATION SCENARIOS
Development NFRs		
Maintainability	Ability of the architecture to support the changes needed by the software in the case if changing requirements	S1.1 – Introduction of a new OS version for the mHealth App
		S1.2 – Update of the current version of the EHR system
		S1.3 – Introduction of a new data protection rule
		S1.4 – Introduction of new health information exchange standard versions
Flexibility	Ability of the architecture to support the development of new software solution for a different case study	S2.1 – mHealth App developed for a different OS
		S2.2 – Introduction of a new EHR system
		S2.3 – Development of a mHealth App for monitoring a different pathology
Scalability	Reliability of the architecture in a changed workload	S3.1 – Increased number of monitored patients (multiple access)
		S3.2 – Increased complexity of the parameters to be reported to the EHR
Operational NFRs		
Reliability	Ability of the architecture to ensure the transmission of reliable data	S4.1 – Creation of a duplicate patient on the EHR side (same patient with two different patient IDs)
Fault-tolerance	Ability of the architecture to resist to system failures	S5.1 – Lack of internet connection on the mHealth App side
		S5.2 – Access system failure on the EHR side

DESIGN PATTERN

- Identificazione di un problema che ricorre spesso con diverse forme e varianti
- Definizione di una possibile soluzione in termini di organizzazione di classi/oggetti che generalmente si è rilevata efficace a risolvere il problema stesso.
- Non sono componenti software ma soluzioni

DESIGN PATTERN: CARATTERISTICHE

Nome

- riferimento mnemonico che permette di identificare il problema e la soluzione in una o due parole

Problema

- descrizione del problema e del contesto a cui il pattern intende fornire una soluzione

Soluzione

- elementi fondamentali che costituiscono la soluzione e le relazioni che intercorrono tra questi

Conseguenze

- possibili conseguenze che l'applicazione della soluzione proposta può comportare (possibili problemi di spazio o efficienza della soluzione, applicabilità con specifici linguaggi di programmazione)

CRITERI DI CLASSIFICAZIONE

- IN BASE AL DOMINIO
  - Oggetti: relazioni fra oggetti che possono modificarsi a tempo di esecuzione
  - Classi: riguardano relazioni tra classi e sottoclassi

- IN BASE ALLA FUNZIONE
  - Creazionali (Creational): forniscono meccanismi per la creazione di oggetti
  - Strutturali (Structural): gestiscono la separazione tra interfaccia e implementazione e le modalità di composizione tra oggetti per creare strutture dati complesse
  - Comportamentali (Behavioral): consentono la modifica del comportamento degli oggetti minimizzando la quantità di codice da cambiare

ESEMPIO: SINGLETON

Scopo

- Fare in modo che a run-time esista al più una sola istanza di una classe e fornire un punto globale di accesso a tale istanza

Motivazione

- É spesso importante avere una sola istanza di una classe a run-time. Si consideri ad esempio il caso del File System Manager o del Window Manager di un sistema.

Applicabilità

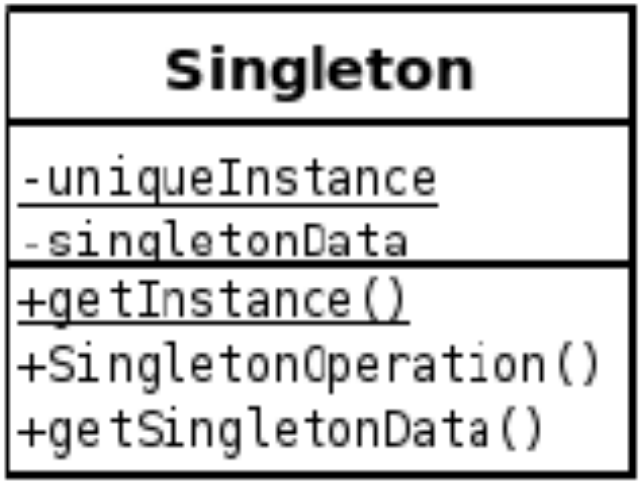
- il pattern singleton può essere usato quando
  - deve esistere una sola istanza di una classe e un punto di accesso noto agli utilizzatori
  - l'unica istanza deve poter essere estesa attraverso definizione di sottoclassi ed i client non devono essere modificati come conseguenza di ciò

Conseguenze

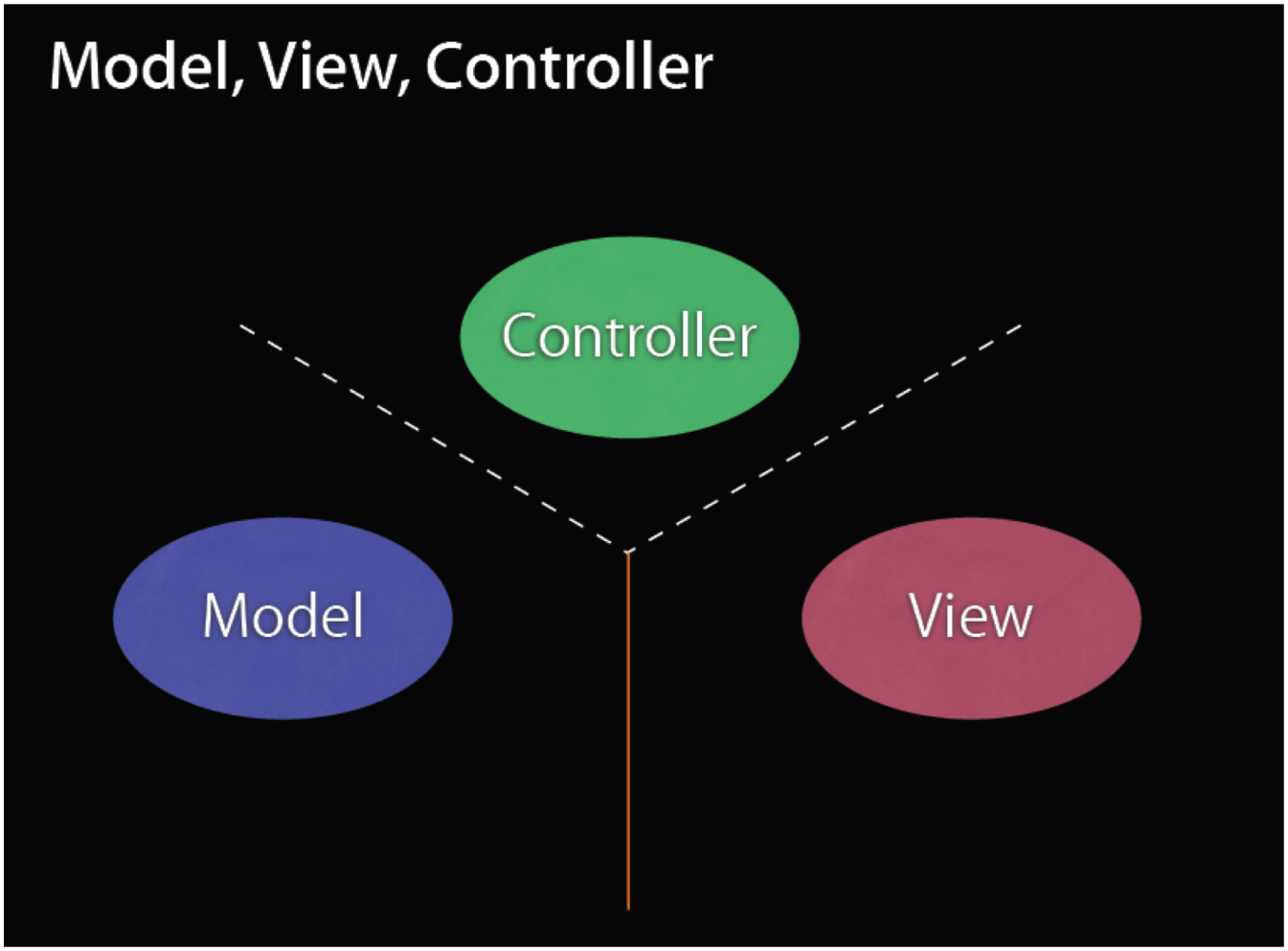
- controllo degli accessi all'istanza
- spazio dei nomi ridotto
- possibilità di estensione ad aver un numero n di istanze
- facile manutenzione

IMPLEMENTAZIONE

Singleton: definisce un'operazione Instance che permette ai client di accedere all'unica istanza disponibile della classe. La detta operazione deve essere un'operazione di classe



Il pattern MVC



Il pattern MVC

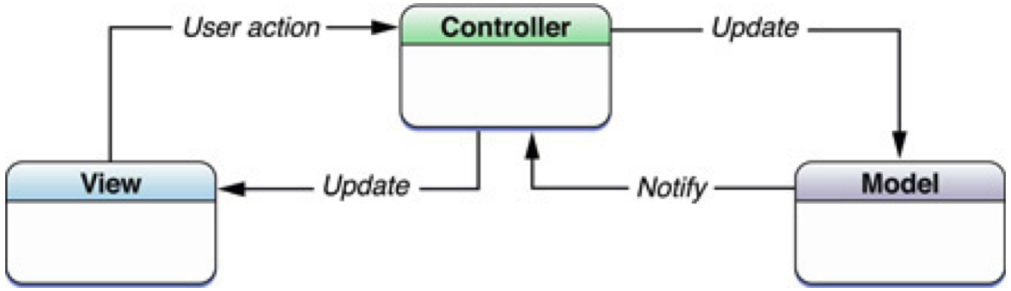
- **MVC** (Model View Controller) è un pattern architetturale molto diffuso nella programmazione orientata agli oggetti, legato tra i tanti linguaggi anche ad Objective-C.

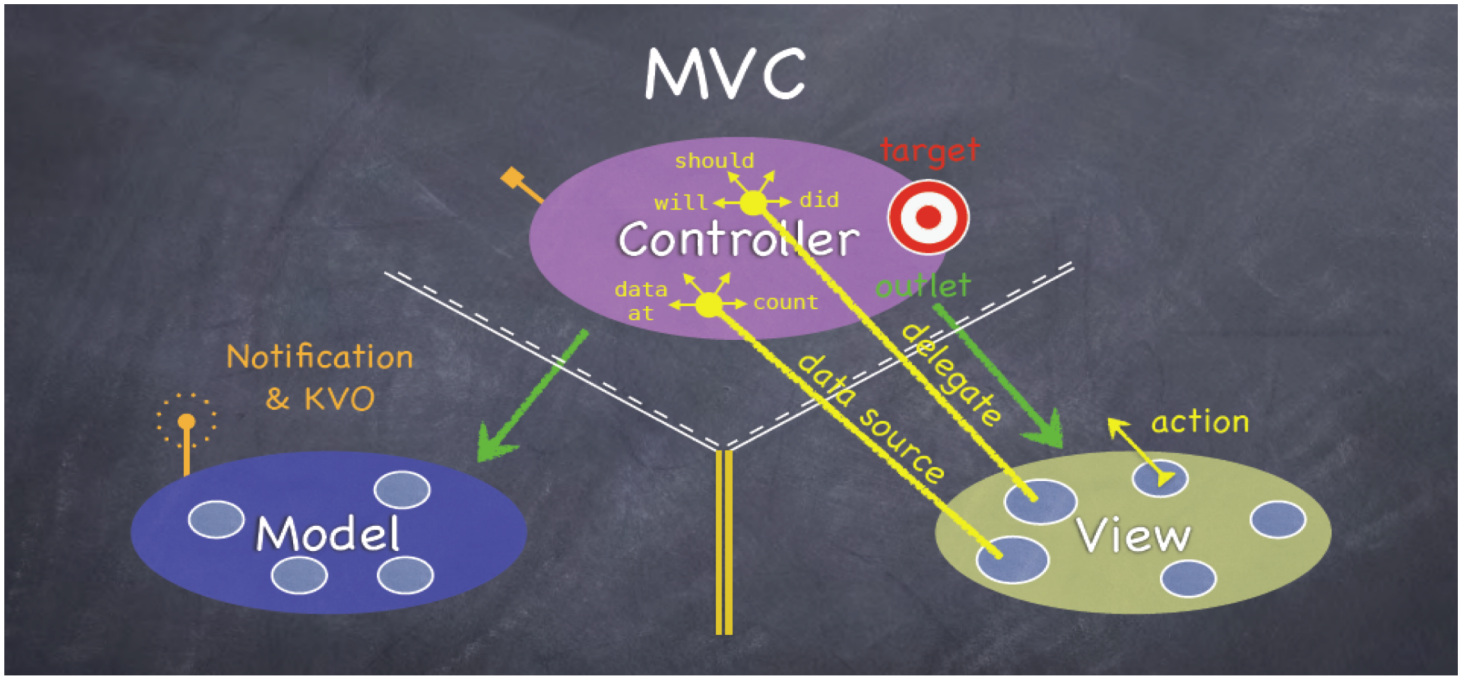
- il **model** (o modello) fornisce i metodi per accedere ai dati utili all'applicazione;
- la **view** (o vista) visualizza i dati contenuti nel model e si occupa dell'interazione con gli utenti;
- il **controller** (o controllore) riceve i comandi dell'utente (in genere attraverso la view) e li attua modificando lo stato degli altri due componenti

- La logica del programma (gestita dal controller), interfaccia utente (gestita dalla view) e i dati insieme alle proprietà (gestiti dal model) sono quindi separati tra loro e gestibili in modo del tutto indipendente.

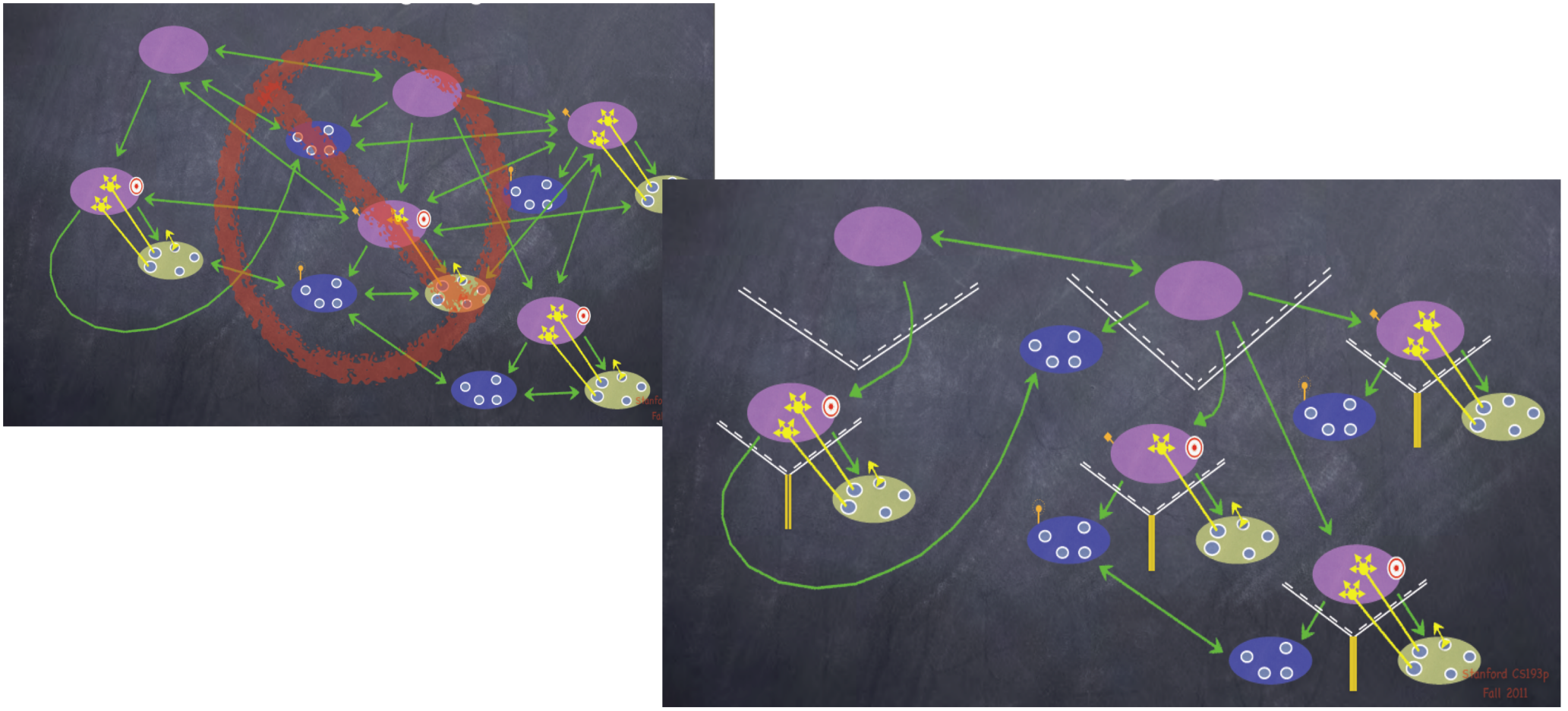
Il pattern MVC per le App

Software → il driver è a logica applicativa (funzionalità offerte)
Apps → il driver è la User Interface e le interazioni





Cooperazione tra gli MVC



By Paul Hegarty, Stanford University

“TAKE HOME MESSAGES” DAL’INGEGNERIA DEL SOFTWARE

- Concetto di MODULARITÀ → la suddivisione in moduli è un vantaggio per la fase di sviluppo e la sua gestione, il suo mantenimento e il suo miglioramento
- Definizione dei requisiti →
  - Metodologie di rappresentazione dei requisiti (ad es: UML)
  - Definizione dei requisiti non funzionali: proprietà del software
- Fase di sviluppo → design pattern