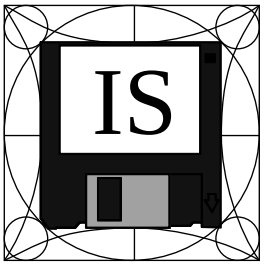
Progettazione *software*



Corso di Ingegneria del Software

V. Ambriola, G.A. Cignoni
C. Montangero, L. Semini

Con aggiornamenti di: T. Vardanega (UniPD)

Contenuti

- La progettazione
- Progettazione architetturale
- Progettazione di dettaglio
- Qualità della progettazione
- Approfondimento: viste multiple

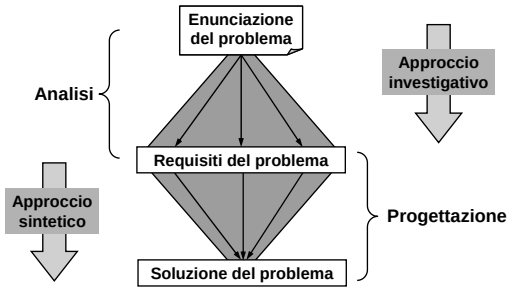
Progettare prima di produrre

- Progettazione precede produzione
  - Approccio industriale e metodo ingegneristico
  - Costruzione *a priori*
  - Prima analisi poi progettazione
- Perché progettare
  - Per governare la complessità del prodotto
  - Per organizzare e ripartire le responsabilità di realizzazione
  - Per produrre in economia
  - Per garantire controllo di qualità

Dall'analisi alla progettazione – 1

- Analisi: quale è il problema e la cosa giusta da fare?
  - Comprensione del dominio
  - Discernimento di obiettivi, vincoli e requisiti
  - Approccio investigativo ⟵
- Progettazione: come farla giusta?
  - Descrizione di una soluzione soddisfacente per tutti i portatori di interesse
  - Il codice non esiste ancora
  - Prodotti: l'architettura scelta e i suoi modelli logici
  - Approccio sintetico ⟵

Dall'analisi alla progettazione – 2



Dall'uso alla progettazione

- Ristrutturazione di prodotto
  - In corso di manutenzione
    - Dopo molte modifiche interne
    - Cambio di piattaforma (HW, S/O, DB) o tecnologia (linguaggio, librerie, ecc.)
- Ripensamento di un sistema
  - Sviluppo che parte direttamente dalla progettazione
    - Analisi a posteriori invece che a priori
    - Requisiti già noti e sostanzialmente stabili
  - Servono interventi, anche radicali, su architettura e codice
  - Re-ingegnerizzazione e riuso

Obiettivi della progettazione

- Soddisfare i requisiti di qualità
- Definire l'architettura del prodotto
  - Impiegando componenti con specifica chiara e coesa
  - Realizzabili con risorse date e costi fissati
  - Struttura che facilita eventuali cambiamenti futuri
- L'architettura è essenziale al raggiungimento degli obiettivi di progetto
  - Diventa essa stessa obiettivo

Progettazione SW

- Attività del processo di sviluppo
  - Procede dall'analisi dei requisiti
  - Produce una descrizione della struttura interna e dell'organizzazione del sistema
    - Fornisce la base della realizzazione
  - Architettura SW
    - Organizzazione del sistema per decomposizione in componenti e interfacce
    - Prima visione logica poi di dettaglio
    - Il livello di dettaglio deve essere sufficiente a guidarne la realizzazione parallela

Punti di vista

- Viste diverse dello stesso sistema
  - Committente: visione d'insieme del prodotto
  - Fornitore: confini del lavoro commissionato
  - Analista: vincoli e rischi tecnologici
  - Progettista: confini dell'attività di commessa
  - Architetto: evoluzione e riuso nel lungo periodo

Definizioni di architettura – 1

- La nozione di "architettura *software*" appare la prima volta nello stesso evento in cui venne riconosciuta la disciplina del *software engineering*
- Fino alla fine degli anni '80 il termine "architettura" viene applicato prevalentemente al sistema fisico
- Nel 1992 D. Perry and A. Wolf propongono
  - {*elements*, *forms*, *rationale*} = *software architecture*
  - *element* = *component* | *connector*

Definizioni di architettura – 2

- B. Boehm, et al., '95
  - A software system architecture comprises
    - A collection of software and system components, connections, and constraints
    - A collection of system stakeholders' need statements
    - A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements

Definizioni di architettura – 3

- P. Kruchten: The Rational Unified Process, '99
  - The set of significant decisions about the organization of a software system
  - The selection of the structural elements and their interfaces by which the system is composed
    - Together with their behavior specified in the collaborations them
  - The composition of these structural and behavioral elements into progressively larger subsystems
  - The architectural style that guides this organization

Definizioni di architettura – 4

- La decomposizione del sistema in componenti
- L' organizzazione di tali componenti
  - Definizione di ruoli, responsabilità, interazioni
- Le interfacce necessarie all'interazione tra le componenti tra loro e con l'ambiente
- I paradigmi di composizione delle componenti
  - Regole, criteri, limiti, vincoli
  - Hanno impatto sulla manutenzione futura

ANSI/IEEE Std 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems

Modelli e metamodelli

- Un modello è una semplificazione della realtà
  - Per ragionare più facilmente sul sistema (problema e soluzione)
  - Usiamo dunque uno o più modelli per descrivere architetture
  - Secondo il punto di vista di uno specifico *stakeholder*
- Nel *software* usiamo modelli per esprimere la semantica statica e dinamica di un sistema
  - Secondo il punto di vista dei vari *stakeholder*
- Il linguaggio che serve per esprimere modelli è fissato in un metamodello
  - Che ne specifica gli elementi lessicali e le regole grammaticali
  - Altrimenti l'interpretazione dei modelli sarebbe arbitraria!

Qualità da ricercare – 1

- Sufficienza
  - È capace di soddisfare tutti i requisiti
- Comprensibilità
  - È comprensibile ai portatori di interesse
- Modularità
  - È divisa in parti chiare e ben distinte
- Robustezza
  - È capace di sopportare ingressi diversi (giusti, sbagliati, tanti, pochi) dall'utente e dall'ambiente

Qualità da ricercare – 2

- Flessibilità
  - Permette modifiche a costo contenuto al variare dei requisiti
- Riusabilità
  - Sue parti possono essere utilmente impiegate in altre applicazioni
- Efficienza
  - Nel tempo, nello spazio e nelle comunicazioni
- Affidabilità
  - È altamente probabile che funzioni bene quando utilizzata

Qualità da ricercare – 3

- Disponibilità
  - Necessità di poco o nullo tempo di manutenzione fuori linea
- Sicurezza rispetto a intrusioni (*security*)
  - I suoi dati e le sue funzioni non sono vulnerabili a intrusioni
- Sicurezza rispetto a malfunzionamenti (*safety*)
  - È esente da malfunzionamenti gravi

Qualità da ricercare – 4

- Semplicità
  - Ogni parte contiene solo il necessario e niente di superfluo
- Incapsulazione (*information hiding*)
  - L'interno delle componenti non è visibile dall'esterno
- Coesione
  - Le parti che stanno insieme hanno gli stessi obiettivi
- Basso accoppiamento
  - Parti distinte dipendono poco o niente le une dalle altre

Progettazione *software*

Semplicità

- **William Ockham (1285 – 1347/49)**
  - ***"Pluralitas non est ponenda sine necessitate"***
    - Le entità usate per spiegare un fenomeno non devono essere moltiplicate senza necessità
- **Principio noto come "il rasoio di Occam"**
  - **Adottato da Isaac Newton (1643 – 1727) nella fisica**
    - "*We are to admit no more causes of natural things than such that are both true and sufficient to explain their appearances*"
      - Quando hai due soluzioni equivalenti rispetto ai risultati scegli la più semplice
  - **E poi anche da Albert Einstein (1879 – 1955)**
    - "*Everything should be made as simple as possible, but not simpler*"

Progettazione *software*

Incapsulazione

- **Le componenti sono "*black box*"**
  - **I suoi clienti ne conoscono solo l'interfaccia**
- **La loro specifica nasconde**
  - **Gli algoritmi e le strutture dati usati all'interno**
- **Benefici**
  - **L'esterno non può fare assunzioni sull'interno**
  - **Cresce la manutenibilità**
  - **Diminuendo le dipendenze aumentano le opportunità di riuso**

Progettazione *software*

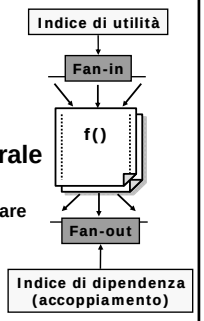
Coesione

- **Proprietà interna di singole componenti**
  - **Funzionalità "vicine" devono stare nella stessa componente**
    - La modularità spinge a decomporre il grande in piccolo
    - La ricerca di coesione aiuta sia a decomporre che a porre un limite inferiore alla decomposizione
  - **Va massimizzata**
- **Benefici**
  - **Maggiore manutenibilità e riusabilità**
  - **Minore interdipendenza fra componenti**
  - **Maggiore comprensione dell'architettura del sistema**

Progettazione *software*

Accoppiamento

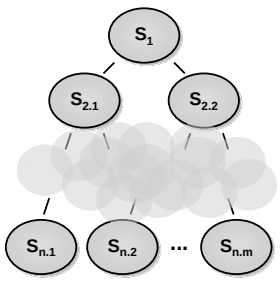
- **Proprietà esterna di componenti**
  - **Quanto M componenti si usano fra di loro**
  - **U = M × M** **massimo accoppiamento**
  - **U = ∅** **accoppiamento nullo**
- **Metriche: *fan-in* e *fan-out* strutturale**
  - **SFIN è indice di utilità ⇒ massimizzare**
  - **SFOUT è indice di dipendenza ⇒ minimizzare**
- **Buona progettazione**
  - **Componenti con SFIN elevato**



Progettazione *software*

Decomposizione architetturale

- ***Top-down* ↓**
  - **Decomposizione di problemi**
  - **Stile funzionale**
- ***Bottom-up* ↑**
  - **Composizione di soluzioni**
  - **Stile *object-oriented***
- ***Meet-in-the-middle* ↓↑**
  - **Approccio intermedio**
    - Il più frequentemente seguito



Progettazione *software*

Riuso

- **Capitalizzare sottosistemi già esistenti**
  - **Impiegandoli più volte per più prodotti**
  - **Ottenendo minor costo realizzativo**
  - **Ottenendo minor costo di verifica**
- **Problemi**
  - **Progettare per riuso è più difficile**
    - Bisogna anticipare bisogni futuri
  - **Progettare con riuso non è immediato**
    - Bisogna minimizzare le modifiche alle componenti riusati per non perderne il valore
- **Puro costo nel breve periodo**
  - **Risparmio (quindi investimento) nel medio termine**

Framework

- Un insieme integrato di componenti SW già sviluppate
  - Nel mondo pre-OO erano chiamate librerie
  - Sono *bottom-up* perché fatti di codice già sviluppato
  - Sono *top-down* perché impongono uno stile architetturale
- Forniscono la base riusabile di diverse applicazioni entro uno stesso dominio
  - Molti importanti esempi nel mondo J2EE
  - Spring (http://www.springsource.org/about) e
  - Struts (http://struts.apache.org/) per Web Apps
  - Swing per GUI, ecc.

Ricetta generale

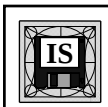
- Dominare la complessità del sistema
  - Organizzare il sistema in componenti di complessità trattabile
    - Secondo la logica del "***divide et impera***"
    - Per ridurre le difficoltà di comprensione e di realizzazione e permettere lavoro individuale
  - Identificare schemi architetturali utili al caso e componenti riusabili
- Riconoscere le componenti terminali
  - Quelle che non richiedono ulteriore decomposizione
    - Il beneficio che ne otterremmo è inferiore al costo
    - Eccessiva esposizione di dettagli causa accoppiamento
- Cercare un buon bilanciamento
  - Più semplici le componenti più complessa la loro interazione

Schemi architetturali

- Soluzioni fattorizzate per problemi ricorrenti
  - Metodo tipico dell'ingegneria classica
  - La soluzione deve riflettere il contesto
    - La soluzione deve soddisfare il bisogno e non viceversa!
  - La soluzione deve essere credibile (dunque provata altrove)
- Esempi
  - Modello di cooperazione di tipo cliente-servente
  - Comunicazione a memoria condivisa o scambio di messaggi
  - Comunicazioni sincrone (interrogazione e attesa)
  - Comunicazioni asincrone (per eventi)

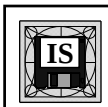
Design pattern

- Soluzione progettuale a problema ricorrente
  - Definisce una funzionalità lasciando gradi di libertà d'uso
    - Ha corrispondenza precisa nel codice sorgente
  - Il corrispondente architetturale degli algoritmi
    - Che invece specificano procedimenti di soluzione
  - Concetto promosso da C. Alexander (un vero architetto)
    - *The Timeless Way of Building*, Oxford University Press, 1979
    - Rilevante nel SW a partire dalla pubblicazione di "*Design Patterns*" della GoF
- A livello sistema servono *pattern* architetturali

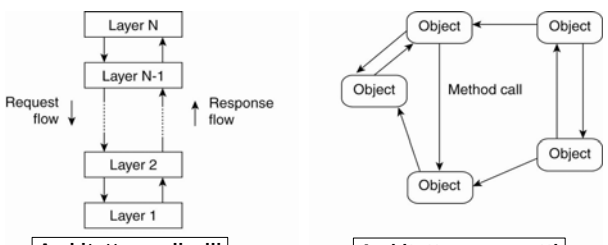


Pattern architetturali

- Architettura "*three-tier*"
  - Strato della presentazione (GUI)
  - Strato della logica operativa (*business logic*)
  - Strato dell'organizzazione dei dati (*database*)
- Architettura produttore-consumatore (*pipeline*)
- Architettura cliente-servente
  - Con cliente complesso ("*fat client*")
    - Meno carico sul servente ma scarsa portabilità
  - Con cliente semplificato ("*thin client*")
    - Maggior carico di comunicazione ma buona portabilità
- Architettura "*peer-to-peer*"
  - Interconnessione senza *server* intermedio



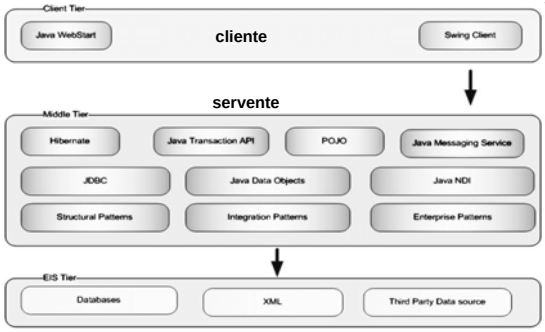
Stili architetturali



Architettura a livelli **Architettura a oggetti**

Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

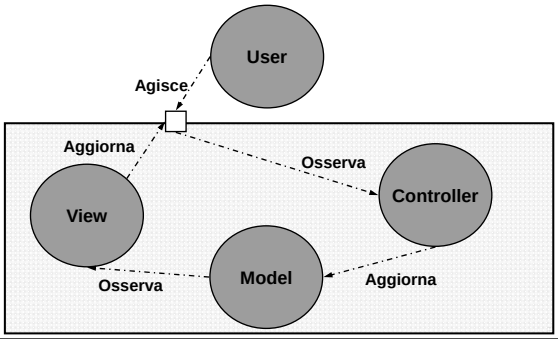
Architettura a livelli



Architettura MVC – 1

- ***Model***
  - Incorpora il modello dei dati e le operazioni su di essi
- ***View***
  - Incorpora la logica della presentazione
  - Riceve input dall'utente e mostra in output l'esito delle azioni conseguentemente svolte dal Controller sul Model
- ***Controller***
  - Incorpora la logica di controllo del sistema

Architettura MVC – 2



Linguaggi di descrizione architetturale

- **Descrizione degli elementi**
  - Componenti, porte e connettori
    - P.es. diagramma delle classi in UML
- **Descrizione dei protocolli di interazione**
  - Tra componenti tramite connettori
- **Supporto ad analisi**
  - Consistenza (analisi statica ad alto livello)
  - Conformità ad attributi di qualità
  - Comparazione tra soluzioni architetturali diverse

Progettazione di dettaglio: attività

- **Definizione delle unità realizzative (moduli)**
  - Un carico di lavoro realizzabile dal singolo programmatore
  - Un "sottosistema" definito
    - Un componente terminale (non ulteriormente decomponibile) o un loro aggregato
  - Un insieme di entità (tipi, dati, funzionalità) strettamente correlate
    - Raccolti insieme in un modulo *package* (come un insieme di classi)
      - Nei sorgenti oppure nel codice oggetto (come in Java)
- **Specifica delle unità come insieme di moduli**
  - Definizione delle caratteristiche significative
    - Da fissare nella progettazione
  - Dal nulla o tramite specializzazione di componenti esistenti

Progettazione di dettaglio: obiettivi

- **Assegnare unità a componenti**
  - Per organizzare il lavoro di programmazione
  - Per assicurare congruenza con l'architettura di sistema
- **Produrre la documentazione necessaria**
  - Perché la programmazione possa procedere
    - Senza bisogno di ulteriori informazioni e senza spazi di creatività
  - Per attribuire requisiti alle unità
    - Tracciamento
  - Per definire le configurazioni ammissibili del sistema
- **Definire gli strumenti per le prove di unità**
  - Casi di prova e componenti ausiliarie
    - Per la verifica delle singole unità e della loro integrazione

Documentazione

- **IEEE 1016:1998 *Software Design Document***
  - **Introduzione**
    - Come nel documento AR (*software requirements specification*)
  - **Riferimenti normativi e informativi**
  - **Descrizione della decomposizione architetturale**
    - Moduli, processi, dati
  - **Descrizione delle dipendenze (tra moduli, processi, dati)**
  - **Descrizione delle interfacce (tra moduli, processi, dati)**
  - **Descrizione della progettazione di dettaglio**

Riepilogo

- **La progettazione**
- **Progettazione architetturale**
- **Progettazione di dettaglio**
- **Qualità della progettazione**
- **Approfondimento: viste multiple**

Riferimenti

- **D. Budgen, Software Design, Addison-Wesley**
- **C. Alexander, The origins of pattern theory, IEEE Software, settembre/ottobre 1999**
  - **http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/images/IBM_Rational/FINAL.SW.V16N5.71.pdf**
- **G. Booch, Object-oriented analysis and design, Addison-Wesley**
- **G. Booch, J. Rumbaugh, I. Jacobson, The UML user guide, Addison-Wesley**
- **C. Hofmeister, R. Nord, D. Soni, Applied Software Architecture, Addison-Wesley, 2000**
- **P. Krutchen, The Rational Unified Process, Addison-Wesley**