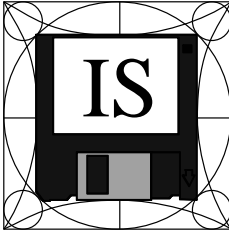


 **Progettazione software**

IS 2001-8  
Corso di Ingegneria del Software  
V. Ambriola, G.A. Cignoni  
C. Montanero, L. Semini  
Con aggiornamenti di: T. Vardanega (UniPD)



Dipartimento di Informatica, Università di Pisa 1/36

 **Progettazione software**  
**Contenuti**

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

Dipartimento di Informatica, Università di Pisa 2/36

 **Progettazione software**  
**Progettare prima di produrre**


- **Progettazione e produzione industriale**
  - *Costruzione a priori*
  - **Analisi e progettazione**
    - Metodo ingegneristico
- **Perché progettare**
  - Per governare la complessità del prodotto
  - Per organizzare e ripartire le responsabilità
  - Per produrre in economia
  - Per garantire controllo di qualità
- **Progettare non è pianificare**

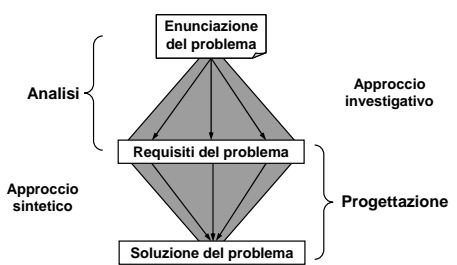
Dipartimento di Informatica, Università di Pisa 3/36

 **Progettazione software**  
**Dall'analisi alla progettazione – 1**

- **Analisi: quale è il problema e la cosa giusta da fare?**
  - Comprensione del dominio
  - **Discernimento di vincoli e requisiti**
    - Approccio investigativo ←
- **Progetto: come farla giusta?**
  - Descrizione di una soluzione che soddisfi tutti i portatori di interesse
  - Il codice non esiste ancora
  - I prodotti della progettazione sono architettura, modello logico, ...
    - Approccio sintetico ←

Dipartimento di Informatica, Università di Pisa 4/36

 **Progettazione software**  
**Dall'analisi alla progettazione – 2**



Dipartimento di Informatica, Università di Pisa 5/36

 **Progettazione software**  
**Dall'uso alla progettazione**

- **Ristrutturazione di prodotto**
  - Dopo molti interventi di manutenzione
  - Per un cambio di piattaforma o di tecnologia
- **Ripensamento di un sistema**
  - Sviluppo che parte direttamente dalla progettazione
    - Analisi a posteriori invece che a priori
    - Requisiti definiti e sostanzialmente stabili
  - Servono interventi, anche radicali, su architettura e codice
  - Re-ingegnerizzazione e riuso

Dipartimento di Informatica, Università di Pisa 6/36

IS Progettazione software

## Obiettivi della progettazione

- Soddisfare i requisiti di qualità
  - Fissati esternamente dal committente e internamente dal fornitore
- Definire l'architettura del prodotto
  - Impiegando componenti con specifica chiara e coesa
  - Realizzabili con risorse date e costi fissati
  - Con struttura che faciliti cambiamenti futuri dovuti a modifiche nei requisiti
- L'architettura è così essenziale al raggiungimento degli obiettivi di prodotto che diventa essa stessa obiettivo

Dipartimento di Informatica, Università di Pisa 7/36

IS Progettazione software

## Punti di vista multipli sul sistema

- Viste diverse per problemi diversi
  - 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
    - Specializzazione "orizzontale" del progettista

Dipartimento di Informatica, Università di Pisa 8/36

IS Progettazione software

## Definizioni di architettura – 1

- 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

Dipartimento di Informatica, Università di Pisa 9/36

IS Progettazione software

## Definizioni di architettura – 2

- 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 as specified in the collaborations among those elements
  - The composition of these structural and behavioral elements into progressively larger subsystems
  - The architectural style that guides this organization

Dipartimento di Informatica, Università di Pisa 10/36

IS Progettazione software

## Definizioni di architettura – 3

- 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

Dipartimento di Informatica, Università di Pisa 11/36

IS Progettazione software

## L'architettura nel RUP - 1

- Quattro viste distinte
  - Logica, realizzativa, processuale, operativa
- Una vista di supporto: casi d'uso
  - Riferimento per analisi e verifica
  - Cattura le interazioni più importanti del sistema con il suo ambiente secondo un particolare punto di vista
- Architettura come sintesi astratta dei modelli prodotti da analisi (problema) e progettazione (soluzione)

Dipartimento di Informatica, Università di Pisa 12/36

Progettazione software

## L'architettura nel RUP - 2

The diagram illustrates four perspectives of software architecture surrounding a central 'Vista per casi d'uso (use case view)'. The perspectives are:

- Prospettiva dell'utente (Vista logica / logical view):** Focuses on 'Funzionalità' (Functionality).
- Prospettiva del programmatore (Vista realizzativa / development view):** Focuses on 'Trattabilità software' (Software manageability).
- Prospettiva dell'integratore (Vista processuale / process view):** Focuses on 'Prestazioni, espandibilità, colli di bottiglia' (Performance, scalability, bottlenecks).
- Prospettiva del progettista di sistema (Vista operativa / deployment view):** Focuses on 'Topologia, consegna, installazione, ...' (Topology, delivery, installation, ...).

Dipartimento di Informatica, Università di Pisa 13/36

Progettazione software

## Modelli e metamodelli

- ❑ **Un modello è una semplificazione della realtà**
  - Per ragionare più facilmente sul sistema
  - Secondo il punto di vista di ciascuno *stakeholder*
- ❑ **Nel software usiamo modelli per esprimere la semantica statica e dinamica di un sistema**
  - Secondo il punto di vista di ciascuno *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!

Dipartimento di Informatica, Università di Pisa 14/36

Progettazione software

## Attributi di architettura

- ❑ **Capacità prestazionale**
  - Maggiore località delle operazioni riduce l'onere di comunicazione
- ❑ **Sicurezza da intrusioni**
  - Architettura a livelli gerarchici con *information hiding*
  - Le parti più critiche nei livelli più interni
- ❑ **Sicurezza da malfunzionamenti gravi**
  - Componenti critiche isolate e protette da interferenze esterne
- ❑ **Continuità di servizio (availability)**
  - Presenza di componenti ridondanti
- ❑ **Manutenibilità**
  - Componenti a grana fine, con poche dipendenze dall'esterno

Dipartimento di Informatica, Università di Pisa 15/36

Progettazione software

## Decomposizione architetturale

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

Dipartimento di Informatica, Università di Pisa 16/36

Progettazione software

## Riuso

- ❑ **Capitalizzare i sottosistemi già realizzati**
  - Impiegandoli più volte per più prodotti
  - Ottenendo minor costo realizzativo
  - Ottenendo minor costo di verifica
- ❑ **Problemi**
  - Progettare diventa un problema aperto
    - Occorre anticipare bisogni futuri
  - È raro riusare al 100%
    - Modificare è difficile e rischioso
- ❑ **Costo (investimento) nel breve periodo**
  - Risparmio a lungo termine

Dipartimento di Informatica, Università di Pisa 17/36

Progettazione software

## Problemi di progettazione - 1

- ❑ **Architettura interna**
  - Esempio: sistema distribuito o sistema centralizzato
- ❑ **Architettura esterna**
  - Esempi: piattaforma di S/O; supporto DBMS; sistema di comunicazione e trasporto dati; sistema di interfaccia utente

Dipartimento di Informatica, Università di Pisa 18/36

Progettazione software

**Problemi di progettazione – 2**

- **Tecnologie**
  - Esempi: linguaggi di programmazione e strumenti di sviluppo associati
- **Realizzazione**
  - Esempio: componenti acquistabili, riusabili, da sviluppare ex-novo
- **Tecniche**
  - Esempio: scelte algoritmiche

Dipartimento di Informatica, Università di Pisa
19/36

Progettazione software

**Problemi di progettazione – 3**

- **Organizzazione e controllo**
  - **Quantificazione del lavoro**
    - Esempio: coerente con uso di COCOMO fatto in pianificazione
  - **Uso delle risorse e ripartizione dei compiti**
    - Esempio: coerente con il piano di progetto in vigore
  - **Allestimento dell'ambiente di lavoro**
    - Esempi: decisione su sviluppo, test, congruente con versionamento, configurazione, tracciamento (di azioni, requisiti, problemi aperti)

Dipartimento di Informatica, Università di Pisa
20/36

Progettazione software

**Strumenti di progettazione – 1**

- **Strumenti concettuali**
  - **Definizione del "sistema software"**
    - Visione astratta distinta dalla realizzazione concreta
  - **Delimitazione dei problemi e delle soluzioni**
  - **Decomposizione e incapsulazione**
    - Massima coesione
    - Minimo accoppiamento
  - **Progettazione orientata agli oggetti**
    - Generalizzazione, specializzazione, riuso

Dipartimento di Informatica, Università di Pisa
21/36

Progettazione software

**Strumenti di progettazione – 2**

- **Strumenti metodologici**
  - **Prima progettazione architeturale del sistema**
    - Da sistema a componenti architeturali
    - Può essere parallelizzata solo se il sistema stesso è stato modularizzato (*pattern* architeturali)
  - **Poi progettazione di dettaglio delle componenti**
    - Da componenti a moduli
    - Può essere parallelizzata sotto la regola che la progettazione interna di un componente non ne modifichi la specifica esterna

Dipartimento di Informatica, Università di Pisa
22/36

Progettazione software

**Progettazione architeturale – 1**

- **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
  - **Identificare schemi realizzativi e componenti riusabili (*pattern*)**
- **Riconoscere le componenti terminali**
  - **Che non necessitano di ulteriore decomposizione**
    - Il beneficio ottenibile non ne vale il costo
    - Eccessiva esposizione di dettagli
      - Troppe assunzioni sul funzionamento interno
- **Cercare un buon bilanciamento**
  - Più semplici le componenti più complessa la loro interazione

Dipartimento di Informatica, Università di Pisa
23/36

Progettazione software

**Progettazione architeturale – 2**

- **Astrazione vs. dettaglio**
  - Ricerca del "miglior equilibrio"
  - Procedendo per decomposizione e specializzazione
- **Semplicità**
  - Massimizzare
- **Incapsulazione (*Information hiding*)**
  - Massimizzare
- **Coesione**
  - Massimizzare
- **Accoppiamento**
  - Minimizzare

Caratteristiche da ricercare e preservare anche nella progettazione di dettaglio

Dipartimento di Informatica, Università di Pisa
24/36

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"

Dipartimento di Informatica, Università di Pisa

25/36

Progettazione software

## Incapsulazione

- **Componenti "black box"**
  - Fornitori e clienti di funzionalità (relazione d'uso)
  - È nota solo la loro interfaccia (dichiarazione dei servizi)
- **Sono mantenuti nascosti**
  - Algoritmi usati
  - Strutture dati interne
- **Vantaggi di un alto grado di incapsulazione**
  - Nessuna assunzione sul funzionamento della componente
  - Maggiore manutenibilità
  - Maggiori opportunità di riuso

Dipartimento di Informatica, Università di Pisa

26/36

Progettazione software

## Coesione

- **Proprietà endogena di singole componenti**
  - Funzionalità "vicine" devono stare nello stesso componente
  - Vicinanza per tipologia, algoritmi, dati in ingresso e in uscita
- **Vantaggi di un elevato grado di coesione**
  - Maggiore manutenibilità e riusabilità
  - Minore interdipendenza fra componenti
  - Maggiore comprensione dell'architettura del sistema
    - Chi fa cosa

Dipartimento di Informatica, Università di Pisa

27/36

Progettazione software

## Accoppiamento

- **Proprietà esogena di componenti**
  - Quanto M componenti si usano fra di loro
  - $U = M \times M$  massimo accoppiamento
  - $U = \emptyset$  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
  - Sistema con SFOUT > 0
    - Necessario accoppiamento con l'ambiente

Dipartimento di Informatica, Università di Pisa

28/36

Progettazione software

## Schemi architetturali

- **Soluzioni fattorizzate per problemi ricorrenti**
  - Riprendere un metodo tipico dell'ingegneria classica
    - Sviluppandolo ai fini software ⇒ Design pattern
  - 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)

Dipartimento di Informatica, Università di Pisa

29/36

Progettazione software

## 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
- **Per il livello sistema servono "pattern architetturali"**

Dipartimento di Informatica, Università di Pisa

30/36

Progettazione software

## 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 (“*client-server*”)
  - Con cliente complesso (“*fat client*”)
    - Meno carico sul server ma scarsa portabilità
  - Con cliente semplificato (“*thin client*”)
    - Maggior carico di comunicazione ma buona portabilità
- ❑ Architettura “peer-to-peer”
  - Interconnessione senza *server* intermedio

Dipartimento di Informatica, Università di Pisa 31/36

Progettazione software

## 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

Dipartimento di Informatica, Università di Pisa 32/36

Progettazione software

## Progettazione di dettaglio: attività

- ❑ Definizione delle unità realizzative (i 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

Dipartimento di Informatica, Università di Pisa 33/36

Progettazione software

## 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

Dipartimento di Informatica, Università di Pisa 34/36

Progettazione software

## Riepilogo

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

Dipartimento di Informatica, Università di Pisa 35/36

Progettazione software

## 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\\_ieeeecs/ieeecs/images/IBM\\_Rational/FINAL\\_SW.V16NS.71.pdf](http://www.computer.org/portal/cms_docs_ieeeecs/ieeecs/images/IBM_Rational/FINAL_SW.V16NS.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

Dipartimento di Informatica, Università di Pisa 36/36