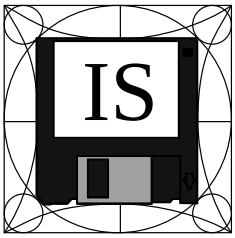
Progettazione *software*



IS 2001-4
Corso di Ingegneria del Software

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

Con aggiornamenti di: T. Vardanega

Contenuti

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

Progettare prima di produrre

- **Progettazione e produzione industriale**
  - Costruzione *a priori*
  - Analisi e progettazione → il metodo ingegneristico
- **Perché progettare**
  - Complessità del prodotto
  - Organizzazione e ripartizione delle responsabilità
  - Economia di produzione
  - Controllo di qualità
- **Progettare non è pianificare**

Dall'analisi alla progettazione

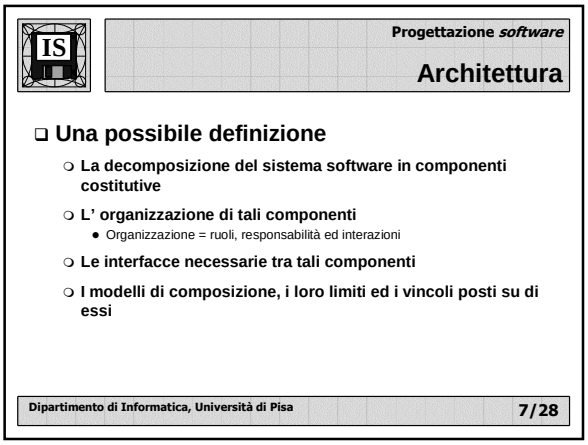
- **Analisi: qual'è la cosa giusta da fare?**
  - Comprensione del dominio
  - Discernimento dei vincoli e dei requisiti
    - Approccio **investigativo**
- **Progetto: come farla giusta?**
  - Descrizione di una soluzione che soddisfi tutti i portatori di interesse
  - Il codice non esiste ancora → i prodotti di questa fase sono l'architettura, il modello logico, ...
    - Approccio **sintetico** con valutazione delle possibili alternative

Dall'uso alla progettazione

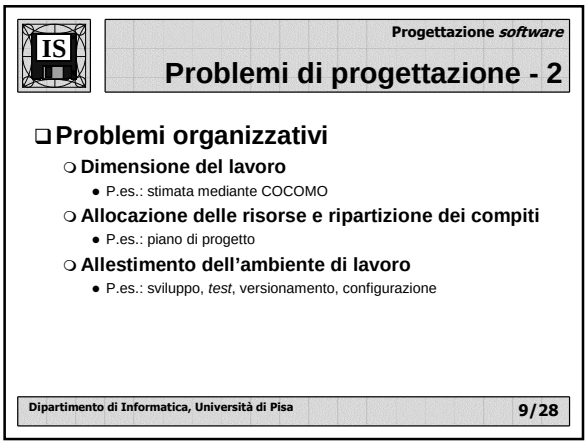
- **Ristrutturazione di prodotto**
  - Dopo molti interventi di manutenzione
  - Per un cambio di piattaforma o di tecnologia
- **Ripensamento di un sistema**
  - Sviluppo che parte direttamente dalla progettazione
  - Requisiti definiti e sostanzialmente stabili (non serve analisi)
  - Servono interventi, anche radicali, su architettura e codice
  - Re-ingegnerizzazione e riuso

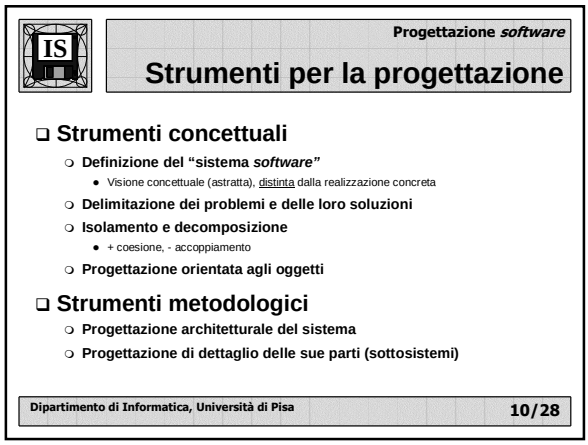
Obiettivi della progettazione

- **Soddisfare i requisiti di qualità fissati dall'utente e dal produttore**
- **Definire l'architettura del prodotto**
  - Impiegando componenti con specifica chiara e coesa
  - Realizzabili con risorse fissate
  - Con struttura che faciliti cambiamenti futuri dovuti a modifica od evoluzione dei requisiti
- **L'architettura non è il fine, ma lo strumento per il raggiungimento degli obiettivi di progetto**

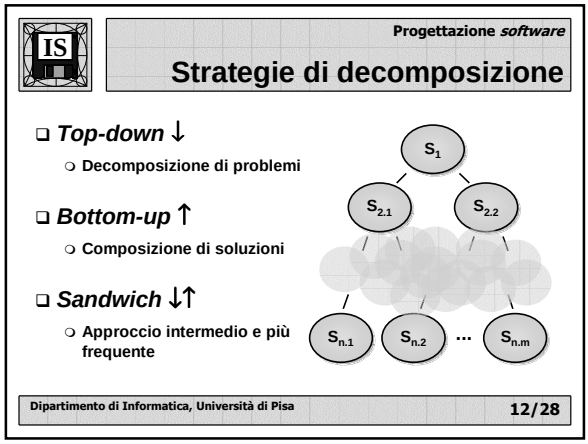

IS
Progettazione *software*
Architettura
Una possibile definizione
La decomposizione del sistema software in componenti costitutive
L' organizzazione di tali componenti
Organizzazione = ruoli, responsabilità ed interazioni
Le interfacce necessarie tra tali componenti
I modelli di composizione, i loro limiti ed i vincoli posti su di essi
Dipartimento di Informatica, Università di Pisa
7/28


IS
Progettazione *software*
Problemi di progettazione - 1
Problemi tecnologici
Strutturali
P.es.: sistema distribuito vs. sistema centralizzato
Infrastrutturali
P.es.: piattaforma di sistema operativo; supporto DBMS; sistema di comunicazione e trasporto dati; sistema di interfaccia utente
Tecnologici
P.es.: linguaggi di programmazione e strumenti di sviluppo associati
Realizzativi
P.es.: componenti acquistabili, riusabili o da sviluppare
Tecnici
P.es.: scelte algoritmiche
Dipartimento di Informatica, Università di Pisa
8/28


IS
Progettazione *software*
Problemi di progettazione - 2
Problemi organizzativi
Dimensione del lavoro
P.es.: stimata mediante COCOMO
Allocazione delle risorse e ripartizione dei compiti
P.es.: piano di progetto
Allestimento dell'ambiente di lavoro
P.es.: sviluppo, *test*, versionamento, configurazione
Dipartimento di Informatica, Università di Pisa
9/28


IS
Progettazione *software*
Strumenti per la progettazione
Strumenti concettuali
Definizione del "sistema *software*"
Visione concettuale (astratta), distinta dalla realizzazione concreta
Delimitazione dei problemi e delle loro soluzioni
Isolamento e decomposizione
+ coesione, - accoppiamento
Progettazione orientata agli oggetti
Strumenti metodologici
Progettazione architetturale del sistema
Progettazione di dettaglio delle sue parti (sottosistemi)
Dipartimento di Informatica, Università di Pisa
10/28


IS
Progettazione *software*
Progettazione architetturale
Dominare la complessità del sistema
Organizzare il sistema in componenti di bassa complessità
Secondo la logica del "*divide et impera*"
Per ridurre le difficoltà di comprensione e realizzazione
Identificare schemi realizzativi e componenti riusabili
Riconoscere le componenti terminali
Che non necessitano di ulteriori trasformazioni
Il beneficio ottenibile non vale il costo di decomposizione
Eccessiva esposizione di dettagli → troppe assunzioni sul funzionamento
Cercare un buon bilanciamento
Più semplici le componenti più complessa la loro interazione
Dipartimento di Informatica, Università di Pisa
11/28


IS
Progettazione *software*
Strategie di decomposizione
Top-down ↓
Decomposizione di problemi
Bottom-up ↑
Composizione di soluzioni
Sandwich ↓↑
Approccio intermedio e più frequente
S_1
$S_{2.1}$
$S_{2.2}$
$S_{n.1}$
$S_{n.2}$
...
$S_{n.m}$
Dipartimento di Informatica, Università di Pisa
12/28

Schemi architetturali

- Soluzioni fattorizzate per problemi ricorrenti
  - Riprendere un metodo tipico dell'ingegneria classica
    - Sviluppandolo ai fini software. P.es.: ***design patterns***
  - La soluzione deve riflettere il contesto
    - La soluzione si deve piegare al bisogno e non il bisogno alla soluzione!
  - La soluzione deve essere credibile (dunque provata altrove)
- Esempi
  - Modello di cooperazione di tipo cliente-servente
  - Comunicazione via memoria condivisa o scambio di messaggi
  - Comunicazioni sincrone (interrogazione ed attesa) oppure asincrone (per eventi)

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

- Definizione delle unità realizzative (moduli)
  - Un carico di lavoro realizzabile dal singolo programmatore
  - Un "sottosistema" definito (una componente terminale od un aggregato di esse)
  - Un insieme di funzionalità affini (un modulo *package*)
    - Non necessariamente nel senso Java!
- Specifica delle unità
  - Definizione delle caratteristiche significative
    - Quelle che occorre fissare in fase di progettazione
  - Dal nulla o tramite specializzazione di componenti esistenti

Progettazione di dettaglio: obiettivi

- Attribuire unità a componenti fisiche
  - Anche per organizzare il lavoro di codifica
- Produrre la documentazione necessaria
  - Per far procedere la codifica senza bisogno di ulteriori informazioni
  - Per attribuire i requisiti alle unità (tracciamento)
  - Per definire le configurazioni del sistema
- Definire gli strumenti per le prove delle unità
  - Casi di prova e componenti ausiliarie per le prove e l'integrazione

Progettazione orientata agli oggetti

- Identificazione degli oggetti
  - "Fotografie" statiche del sistema a tempo di esecuzione
  - Ciascuno istanza delle classi individuate durante l'analisi
- Definizione dinamica delle attività del sistema
  - Come attività del sistema: visione d'insieme dall'esterno
  - Come interazioni fra gli oggetti: visione d'insieme dall'interno
  - Come stati degli oggetti: visione di dettaglio interno

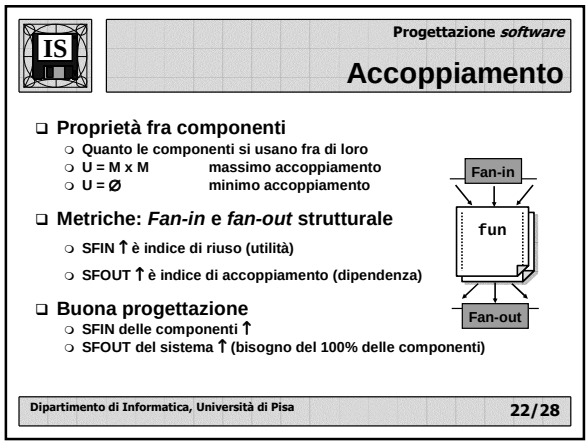
Riuso

- Capitalizzare i sottosistemi già realizzati
  - Impiegandoli più volte per la realizzazione di altri 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% e modificare è difficile e rischioso
- Investimento → risparmio a lungo termine

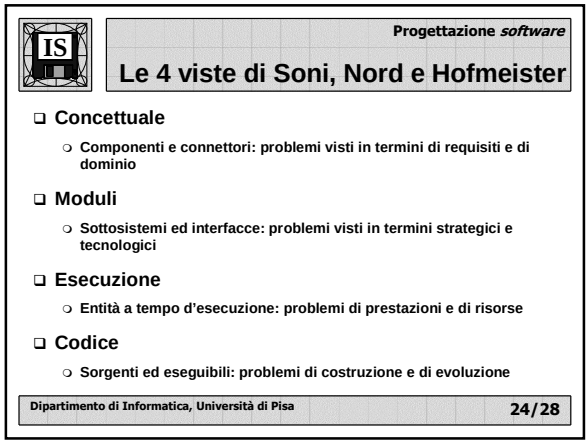

IS
Progettazione *software*
Qualità della progettazione
Progettazione architetturale
Astrazione ↔ dettaglio
Ricerca del "miglior equilibrio"
Progettazione di dettaglio
Incapsulazione (*Information hiding*) ↑
Coesione ↑
Accoppiamento ↓
Dipartimento di Informatica, Università di Pisa
19/28

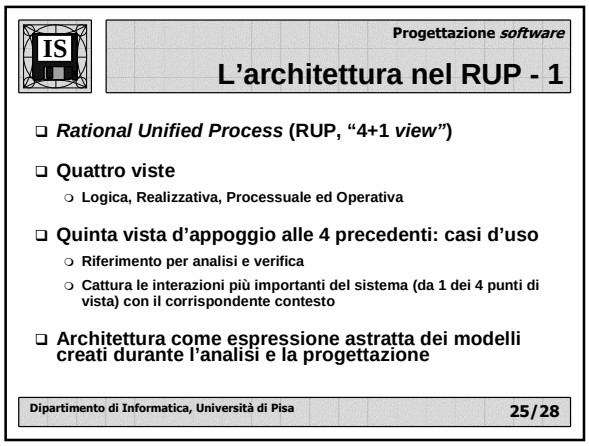

IS
Progettazione *software*
Incapsulazione
Componenti "*black box*"
Fornitori e clienti di funzionalità (relazione d'uso)
È nota solo la loro interfaccia (dichiarazione dei servizi)
Sono mantenuti nascosti
Algoritmi usati
Strutture dati interne
Vantaggi di un alto grado di incapsulazione
Nessuna assunzione sul funzionamento della componente
Maggiore manutenibilità
Maggiori opportunità di riuso
Dipartimento di Informatica, Università di Pisa
20/28

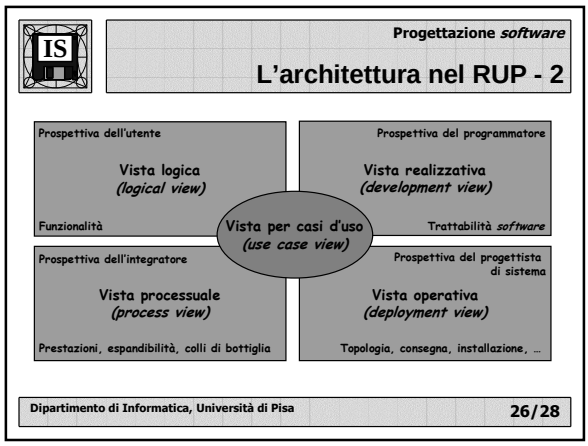

IS
Progettazione *software*
Coesione
Proprietà interna alla componente
Funzionalità "vicine" devono stare nello stesso componente
Vicinanza per tipologia, algoritmi, dati in ingresso ed in uscita
Vantaggi di un alto grado di coesione
Facilita il riuso e migliora la manutenibilità
Riduce il grado di dipendenza fra componenti
Facilita la comprensione dell'architettura del sistema
Dipartimento di Informatica, Università di Pisa
21/28


IS
Progettazione *software*
Accoppiamento
Proprietà fra componenti
Quanto le componenti si usano fra di loro
U = M x M massimo accoppiamento
U = Ø minimo accoppiamento
Metriche: *Fan-in* e *fan-out* strutturale
SFIN ↑ è indice di riuso (utilità)
SFOUT ↑ è indice di accoppiamento (dipendenza)
Buona progettazione
SFIN delle componenti ↑
SFOUT del sistema ↑ (bisogno del 100% delle componenti)
Fan-in
fun
Fan-out
Dipartimento di Informatica, Università di Pisa
22/28

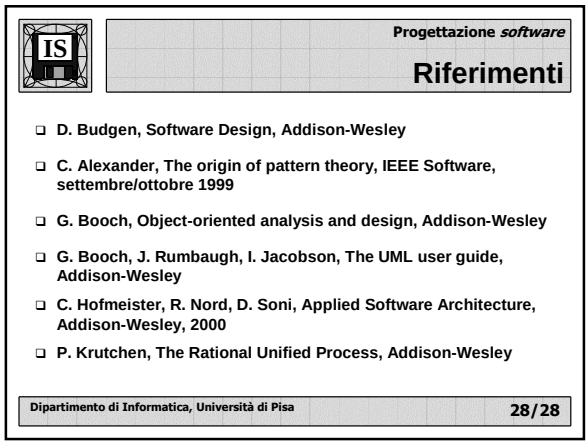

IS
Progettazione *software*
Architetture *software*: viste multiple
Viste diverse per scopi diversi
Utente: per descrizione sintetica del prodotto usato/acquistato
Fornitore: per capire i confini del proprio lavoro
Analista: per individuare i vincoli ed i rischi tecnologici
Progettista: per localizzare i confini della propria attività
Architetto: per ragionare circa evoluzione e riuso
Dipartimento di Informatica, Università di Pisa
23/28


IS
Progettazione *software*
Le 4 viste di Soni, Nord e Hofmeister
Concettuale
Componenti e connettori: problemi visti in termini di requisiti e di dominio
Moduli
Sottosistemi ed interfacce: problemi visti in termini strategici e tecnologici
Esecuzione
Entità a tempo d'esecuzione: problemi di prestazioni e di risorse
Codice
Sorgenti ed eseguibili: problemi di costruzione e di evoluzione
Dipartimento di Informatica, Università di Pisa
24/28


IS
Progettazione *software*
L'architettura nel RUP - 1
Rational Unified Process (RUP, "4+1 *view*")
Quattro viste
Logica, Realizzativa, Processuale ed Operativa
Quinta vista d'appoggio alle 4 precedenti: casi d'uso
Riferimento per analisi e verifica
Cattura le interazioni più importanti del sistema (da 1 dei 4 punti di vista) con il corrispondente contesto
Architettura come espressione astratta dei modelli creati durante l'analisi e la progettazione
Dipartimento di Informatica, Università di Pisa
25/28


IS
Progettazione *software*
L'architettura nel RUP - 2
Prospettiva dell'utente
Vista logica
(logical view)
Funzionalità
Prospettiva del programmatore
Vista realizzativa
(development view)
Trattabilità *software*
Vista per casi d'uso
(use case view)
Prospettiva dell'integratore
Vista processuale
(process view)
Prestazioni, espandibilità, colli di bottiglia
Prospettiva del progettista di sistema
Vista operativa
(deployment view)
Topologia, consegna, installazione, …
Dipartimento di Informatica, Università di Pisa
26/28


IS
Progettazione *software*
Riepilogo
La progettazione
Progettazione architetturale
Progettazione di dettaglio
Qualità della progettazione
Approfondimento: viste multiple
Dipartimento di Informatica, Università di Pisa
27/28


IS
Progettazione *software*
Riferimenti
D. Budgen, Software Design, Addison-Wesley
C. Alexander, The origin of pattern theory, IEEE Software, settembre/ottobre 1999
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
28/28