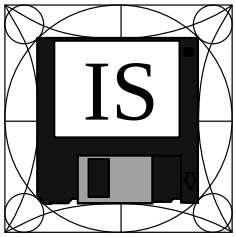
Progettazione *software*



IS 2001-7
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

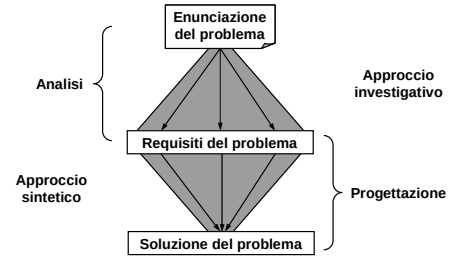
Progettare prima di produrre

- Progettazione e produzione industriale
  - Costruzione *a priori*
  - Analisi e progettazione
    - Il 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

Dall'analisi alla progettazione – 1

- Analisi: quale è il problema e 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'analisi alla progettazione – 2

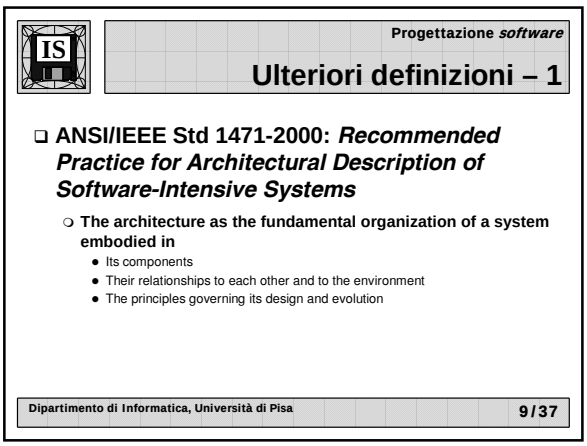


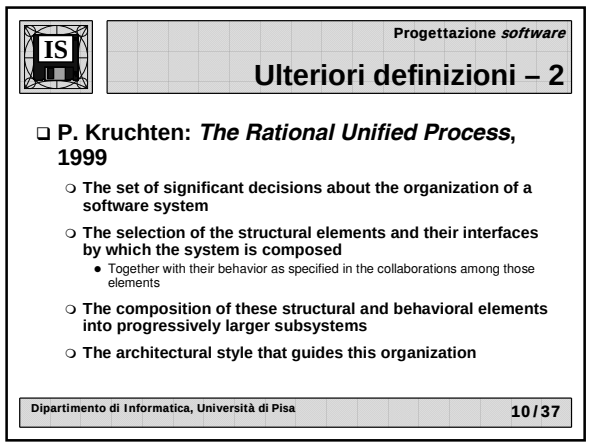
Dall'uso alla progettazione

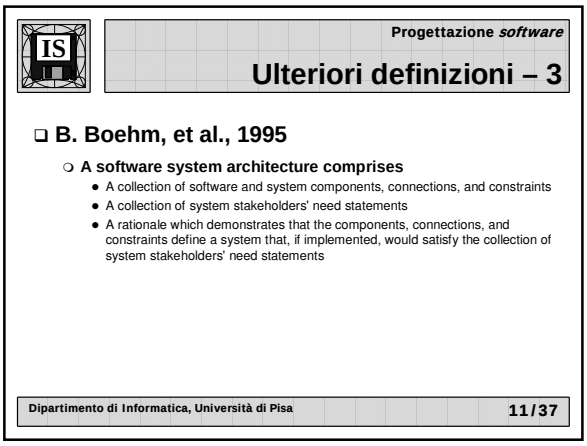
- Ristrutturazione di prodotto
  - Dopo molti interventi di manutenzione
  - Per un cambio di piattaforma o di tecnologia
- Ripensamento di un sistema
  - Sviluppo che parte direttamente dalla progettazione
    - Analisi a posteriori invece che a priori
    - Requisiti definiti e sostanzialmente stabili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


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 modifica od evoluzione dei requisiti
L'architettura non è un fine ma uno strumento importante per il raggiungimento degli obiettivi di progetto
Dipartimento di Informatica, Università di Pisa
7 / 37

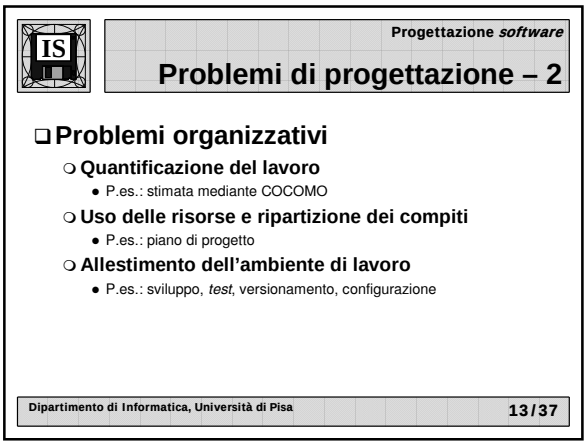

IS
Progettazione *software*
Definizione di architettura – 1
La decomposizione del sistema in componenti costitutive
L' organizzazione di tali componenti
Definizione di ruoli, responsabilità, interazioni
Le interfacce necessarie alla interazione tra i componenti
I modelli di composizione delle componenti
Criteri, limiti, vincoli
Dipartimento di Informatica, Università di Pisa
8 / 37

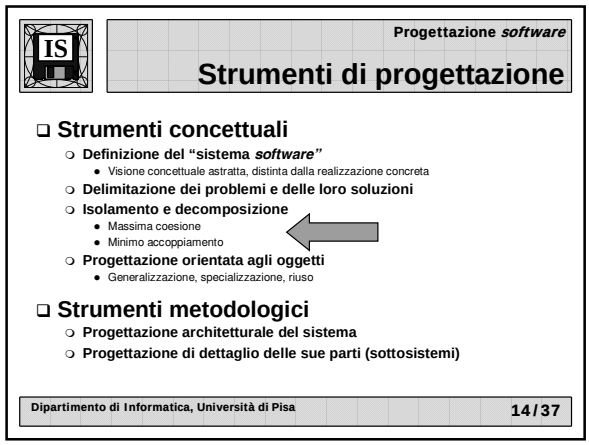

IS
Progettazione *software*
Ulteriori definizioni – 1
ANSI/IEEE Std 1471-2000: *Recommended Practice for Architectural Description of Software-Intensive Systems*
The architecture as the fundamental organization of a system embodied in
Its components
Their relationships to each other and to the environment
The principles governing its design and evolution
Dipartimento di Informatica, Università di Pisa
9 / 37


IS
Progettazione *software*
Ulteriori definizioni – 2
P. Kruchten: *The Rational Unified Process*, 1999
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 / 37

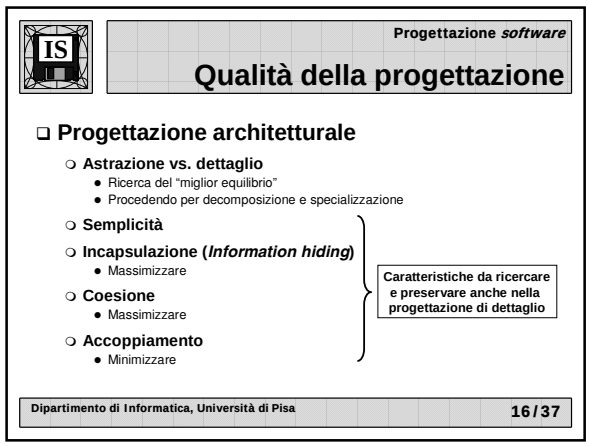

IS
Progettazione *software*
Ulteriori definizioni – 3
B. Boehm, et al., 1995
A software system architecture comprises
A collection of software and system components, connections, and constraints
A collection of system stakeholders' need statements
A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements
Dipartimento di Informatica, Università di Pisa
11 / 37


IS
Progettazione *software*
Problemi di progettazione – 1
Problemi strutturali → architettura interna
P.es.: sistema distribuito vs. sistema centralizzato
Problemi infrastrutturali → architettura esterna
P.es.: piattaforma di sistema operativo; supporto DBMS; sistema di comunicazione e trasporto dati; sistema di interfaccia utente
Problemi tecnologici
P.es.: linguaggi di programmazione e strumenti di sviluppo associati
Problemi realizzativi
P.es.: componenti acquistabili, riusabili o da sviluppare ex-novo
Problemi tecnici
P.es.: scelte algoritmiche
Dipartimento di Informatica, Università di Pisa
12 / 37

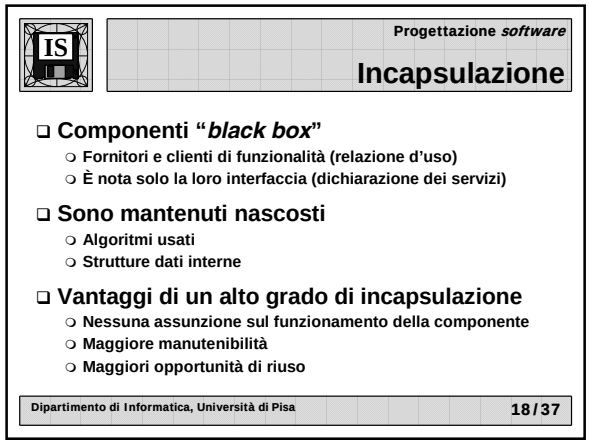

IS
Progettazione *software*
Problemi di progettazione – 2
Problemi organizzativi
Quantificazione del lavoro
P.es.: stimata mediante COCOMO
Uso delle risorse e ripartizione dei compiti
P.es.: piano di progetto
Allestimento dell'ambiente di lavoro
P.es.: sviluppo, *test*, versionamento, configurazione
Dipartimento di Informatica, Università di Pisa
13 / 37


IS
Progettazione *software*
Strumenti di progettazione
Strumenti concettuali
Definizione del "sistema *software*"
Visione concettuale astratta, distinta dalla realizzazione concreta
Delimitazione dei problemi e delle loro soluzioni
Isolamento e decomposizione
Massima coesione
Minimo accoppiamento
Progettazione orientata agli oggetti
Generalizzazione, specializzazione, riuso
Strumenti metodologici
Progettazione architetturale del sistema
Progettazione di dettaglio delle sue parti (sottosistemi)
Dipartimento di Informatica, Università di Pisa
14 / 37

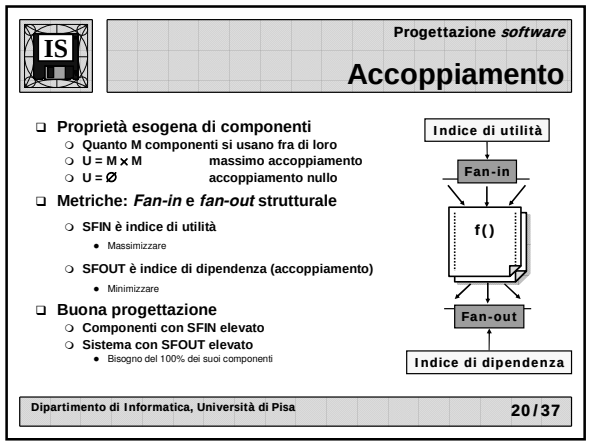

IS
Progettazione *software*
Progettazione architetturale
Dominare la complessità del sistema
Organizzare il sistema in componenti di complessità trattabile
Secondo la logica del "*divide et impera*"
Per ridurre le difficoltà di comprensione e realizzazione
Identificare schemi realizzativi e componenti 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 interno
Cercare un buon bilanciamento
Più semplici le componenti più complessa la loro interazione
Dipartimento di Informatica, Università di Pisa
15 / 37

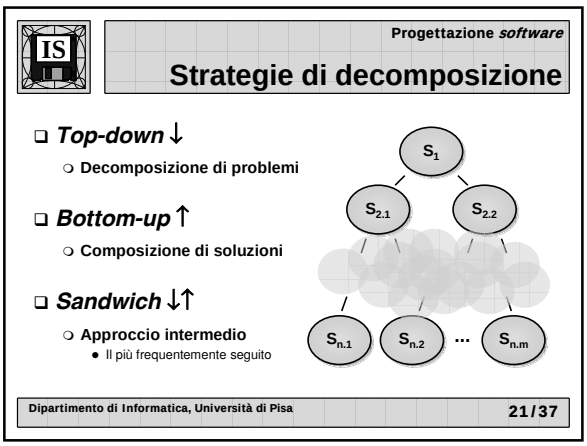

IS
Progettazione *software*
Qualità della progettazione
Progettazione architetturale
Astrazione vs. dettaglio
Ricerca del "miglior equilibrio"
Procedendo per decomposizione e specializzazione
Semplicità
Incapsulazione (*Information hiding*)
Massimizzare
Coesione
Massimizzare
Accoppiamento
Minimizzare
Caratteristiche da ricercare e preservare anche nella progettazione di dettaglio
Dipartimento di Informatica, Università di Pisa
16 / 37

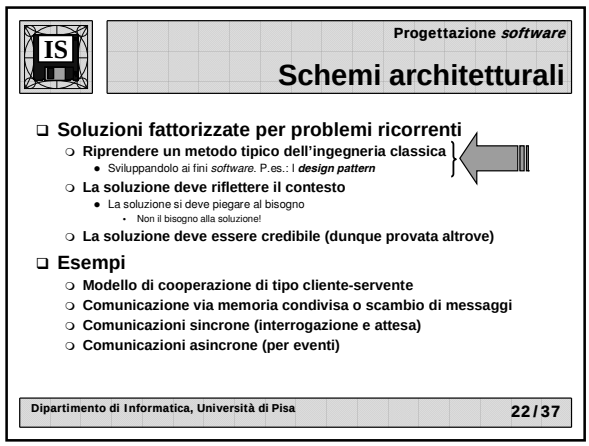

IS
Progettazione *software*
Semplicità
William Ockham (1285 – 1347/49)
"Pluralitas non est ponenda sine necessitate"
Le entità usate per spiegare un fenomeno non devono essere moltiplicate senza necessità
Principio noto come "il rasoio di Occam"
Adottato da Isaac Newton (1643 – 1727) nella fisica
"*We 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*"
Come applicarlo alla progettazione *software*?
Dipartimento di Informatica, Università di Pisa
17 / 37

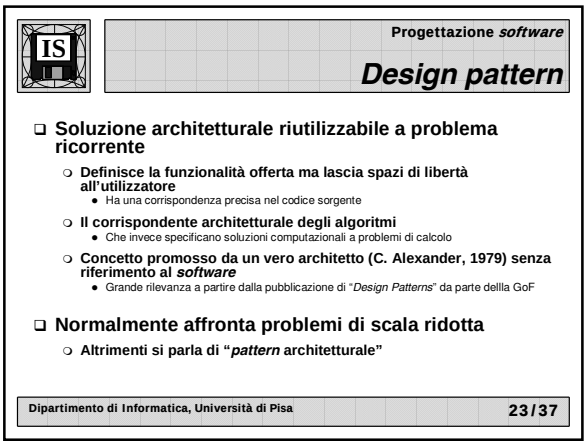

IS
Progettazione *software*
Incapsulazione
Componenti "*black box*"
Fornitori e clienti di funzionalità (relazione d'uso)
È nota solo la loro interfaccia (dichiarazione dei servizi)
Sono mantenuti nascosti
Algoritmi usati
Strutture dati interne
Vantaggi di un alto grado di incapsulazione
Nessuna assunzione sul funzionamento della componente
Maggiore manutenibilità
Maggiori opportunità di riuso
Dipartimento di Informatica, Università di Pisa
18 / 37

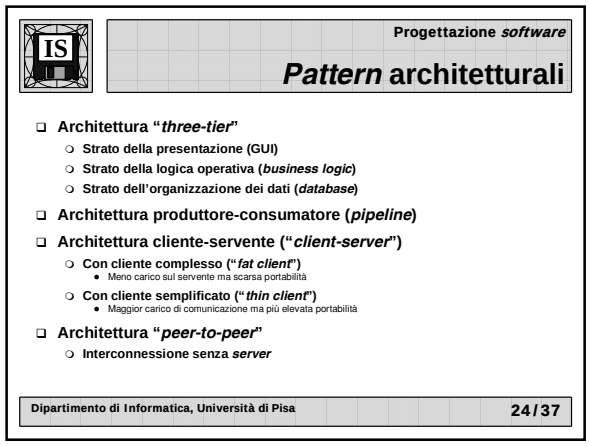

IS
Progettazione *software*
Coesione
Proprietà endogena di singole componenti
Funzionalità "vicine" devono stare nello stesso componente
Vicinanza per tipologia, algoritmi, dati in ingresso e in uscita
Vantaggi di un elevato grado di coesione
Migliora la manutenibilità e facilita il riuso
Riduce il grado di dipendenza fra componenti
Facilita la comprensione dell'architettura del sistema
Chi fa cosa
Dipartimento di Informatica, Università di Pisa
19/37


IS
Progettazione *software*
Accoppiamento
Proprietà esogena di componenti
Quanto M componenti si usano fra di loro
U = M x M massimo accoppiamento
U = Ø accoppiamento nullo
Metriche: *Fan-in* e *fan-out* strutturale
SFIN è indice di utilità
Massimizzare
SFOUT è indice di dipendenza (accoppiamento)
Minimizzare
Buona progettazione
Componenti con SFIN elevato
Sistema con SFOUT elevato
Bisogno del 100% dei suoi componenti
Indice di utilità
Fan-in
f()
Fan-out
Indice di dipendenza
Dipartimento di Informatica, Università di Pisa
20/37


IS
Progettazione *software*
Strategie di decomposizione
Top-down ↓
Decomposizione di problemi
Bottom-up ↑
Composizione di soluzioni
Sandwich ↓↑
Approccio intermedio
Il più frequentemente seguito
S_1
$S_{2.1}$
$S_{2.2}$
$S_{n.1}$
$S_{n.2}$
...
$S_{n.m}$
Dipartimento di Informatica, Università di Pisa
21/37


IS
Progettazione *software*
Schemi architetturali
Soluzioni fattorizzate per problemi ricorrenti
Riprendere un metodo tipico dell'ingegneria classica
Sviluppandolo ai fini *software*. P.es.: I ***design pattern***
La soluzione deve riflettere il contesto
La soluzione si deve piegare al bisogno
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 e attesa)
Comunicazioni asincrone (per eventi)
Dipartimento di Informatica, Università di Pisa
22/37

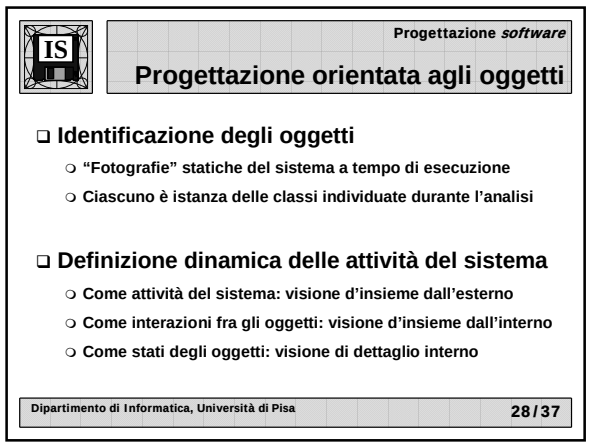

IS
Progettazione *software*
Design pattern
Soluzione architetturale riutilizzabile a problema ricorrente
Definisce la funzionalità offerta ma lascia spazi di libertà all'utilizzatore
Ha una corrispondenza precisa nel codice sorgente
Il corrispondente architetturale degli algoritmi
Che invece specificano soluzioni computazionali a problemi di calcolo
Concetto promosso da un vero architetto (C. Alexander, 1979) senza riferimento al *software*
Grande rilevanza a partire dalla pubblicazione di "*Design Patterns*" da parte dellla GoF
Normalmente affronta problemi di scala ridotta
Altrimenti si parla di "*pattern* architetturale"
Dipartimento di Informatica, Università di Pisa
23/37


IS
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 servente ma scarsa portabilità
Con cliente semplificato ("*thin client*")
Maggior carico di comunicazione ma più elevata portabilità
Architettura "*peer-to-peer*"
Interconnessione senza *server*
Dipartimento di Informatica, Università di Pisa
24/37


IS
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
25 / 37


IS
Progettazione *software*
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
Quelle che occorre fissare in fase di progettazione
Dal nulla o tramite specializzazione di componenti esistenti
Dipartimento di Informatica, Università di Pisa
26 / 37


IS
Progettazione *software*
Progettazione di dettaglio: obiettivi
Assegnare unità logiche a componenti fisiche
Anche per organizzare il lavoro di programmazione
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
27 / 37


IS
Progettazione *software*
Progettazione orientata agli oggetti
Identificazione degli oggetti
"Fotografie" statiche del sistema a tempo di esecuzione
Ciascuno è 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
Dipartimento di Informatica, Università di Pisa
28 / 37


IS
Progettazione *software*
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 nel breve periodo
Ma risparmio a lungo termine
Dipartimento di Informatica, Università di Pisa
29 / 37


IS
Progettazione *software*
Attributi di architettura
Capacità prestazionale
Maggiore località delle operazioni riduce l'onere di comunicazione tra sottosistemi
Sicurezza da intrusioni
Architettura a livelli gerarchici (*information hiding*)
Le parti più critiche nei livelli più interni
Sicurezza da malfunzionamenti gravi
Componenti critiche isolate e protette da possibili interferenze esterne
Continuità di servizio (*availability*)
Presenza di componenti ridondanti
Manutenibilità
Componenti a grana fine, con *fan-out* basso
Dipartimento di Informatica, Università di Pisa
30 / 37

Modelli e metamodelli

- Un modello è una semplificazione della realtà
  - Usata per ragionare più facilmente a proposito del sistema che si intende realizzare, utilizzare, ampliare
- In ambito *software* usiamo modelli per esprimere la semantica statica e dinamica di un sistema
  - Nell'ottica di *stakeholder* diversi, ciascuno con il suo punto di vista
- Il linguaggio che serve per esprimere modelli viene detto metamodello
  - Elementi lessicali e regole grammaticali

Architetture *software*: viste multiple

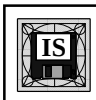
- Viste diverse per scopi diversi
  - Committente: per avere una visione d'insieme del prodotto
  - Fornitore: per capire i confini del proprio lavoro
  - Analista: per individuare i vincoli e i rischi tecnologici
  - Progettista: per localizzare i confini della propria attività
  - Architetto: per ragionare circa evoluzione e riuso
    - Specializzazione "orizzontale" del progettista

Le 4 viste di Soni, Nord e Hofmeister

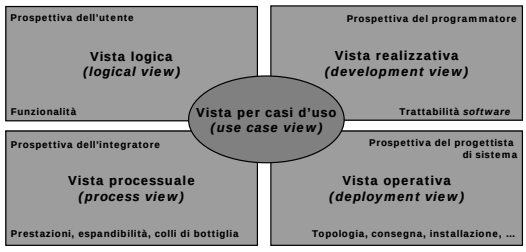
- Concettuale
  - Componenti e connettori: problemi visti in termini di requisiti e di dominio
- Moduli
  - Sottosistemi e interfacce: problemi visti in termini strategici e tecnologici
- Esecuzione
  - Entità a tempo d'esecuzione: problemi di prestazioni e di risorse
- Codice
  - Sorgenti ed eseguibili: problemi di costruzione e di evoluzione

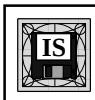
L'architettura nel RUP - 1

- *Rational Unified Process* (RUP, "4+1 *view*")
- Quattro viste
  - Logica, Realizzativa, Processuale e 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



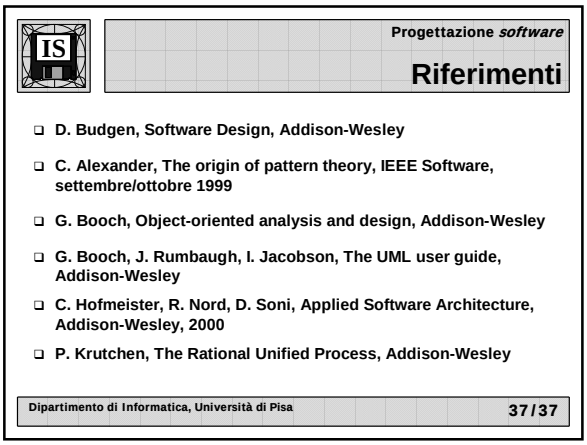
L'architettura nel RUP - 2





Riepilogo

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


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
37 / 37