



Progettazione software



Ingegneria del Software
V. Ambriola, G.A. Cignoni
C. Montangero, L. Semini

Aggiornamenti di: T. Vardanega (UniPD)

Dipartimento di Informatica, Università di Pisa

1/41




Progettazione software

Dall'analisi alla progettazione – 1

- ❑ **Analisi: quale è il problema, quale 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 gli *stakeholder*
 - Il codice non esiste ancora
 - Prodotti: l'architettura scelta e i suoi modelli logici
 - Approccio sintetico ←

Dipartimento di Informatica, Università di Pisa

3/41




Progettazione software

Progettare prima di produrre

- ❑ **La progettazione precede la produzione**
 - Approccio industriale e metodo ingegneristico
 - Costruzione *a priori* perseguendo la «correttezza per costruzione»
 - Analisi → progettazione → codifica
- ❑ **Progettare per**
 - Governare la complessità del prodotto (divide-et-impera)
 - Organizzare e ripartire le responsabilità di realizzazione
 - Produrre in economia (efficienza)
 - Garantire controllo di qualità (efficacia)

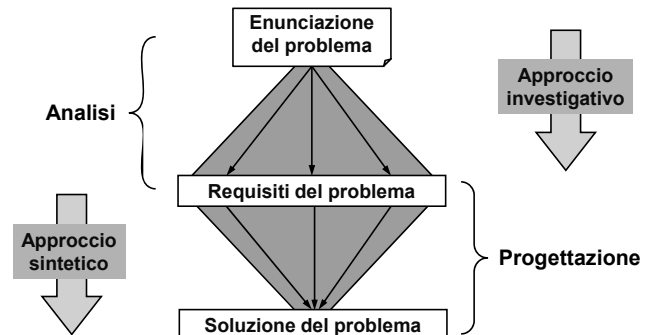
Dipartimento di Informatica, Università di Pisa

2/41



Progettazione software

Dall'analisi alla progettazione – 2



The diagram illustrates the process flow. At the top is 'Enunciazione del problema'. Below it is a diamond-shaped structure with 'Requisiti del problema' in the center. At the bottom is 'Soluzione del problema'. A bracket on the left labeled 'Analisi' encompasses the top and middle parts. A bracket on the right labeled 'Progettazione' encompasses the middle and bottom parts. A large downward arrow on the right is labeled 'Approccio investigativo'. A large downward arrow on the left is labeled 'Approccio sintetico'.

Dipartimento di Informatica, Università di Pisa

4/41

Progettazione software

Dall'analisi alla progettazione – 3

- Dice Edsger W. Dijkstra in «*On the role of scientific thought*» (1982)
- The task of “making a thing satisfying our needs” as a single responsibility is split into two parts*
 1. *Stating the properties of a thing, by virtue of which it would satisfy our needs, and*
 2. *Making a thing guaranteed to have the stated properties*
- La prima responsabilità è dell'analisi**
- La seconda è di progettazione e codifica**


Dipartimento di Informatica, Università di Pisa5/41

Progettazione software

Attività della progettazione

- 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
 - Fissa l'architettura SW**
 - Organizzazione del sistema per decomposizione in componenti e interfacce
 - Prima visione logica (concettuale) poi di dettaglio (realizzativa)
 - Il livello di dettaglio deve essere sufficiente a guidarne la codifica in parallelo


Dipartimento di Informatica, Università di Pisa7/41

Progettazione software

Obiettivi della progettazione

- Soddisfare i requisiti con un sistema di qualità**
- Definendo l'architettura del prodotto**
 - Impiegando componenti con specifica chiara e coesa
 - Realizzabili con risorse date e costi fissati
 - Struttura che facilita eventuali cambiamenti futuri
- La scelta di una buona architettura abilita al successo**


Dipartimento di Informatica, Università di Pisa6/41

Progettazione software

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
- Tutte vanno tenute in conto**

Dipartimento di Informatica, Università di Pisa8/41




Progettazione *software*

Arte e architettura

- ❑ Intorno al 1915, lo scrittore H.G. Wells (1866 – 1946), autore, tra altre opere, di “The War of the Worlds” (1898), scrive in una lettera al collega Henry James (1843 – 1916)
- ❑ *To you literature like painting is an end, to me literature like architecture is a means, it has a use*
- ❑ L’arte è un fine, l’architettura un mezzo

Dipartimento di Informatica, Università di Pisa9/41




Progettazione *software*

Definizioni di architettura – 2

- ❑ **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

[B. Boehm, et al., '95]

Dipartimento di Informatica, Università di Pisa11/41




Progettazione *software*

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*
- ❑ Tuttavia 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

Dipartimento di Informatica, Università di Pisa10/41




Progettazione *software*

Definizioni di architettura – 3

- ❑ **A software system architecture comprises**
 - 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

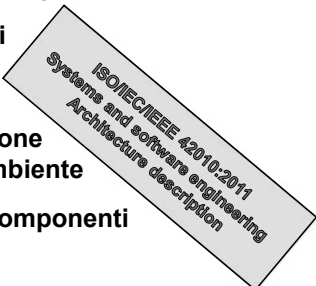
[P. Kruchten: The Rational Unified Process, '99]

Dipartimento di Informatica, Università di Pisa12/41

Progettazione *software*

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




Dipartimento di Informatica, Università di Pisa13/41

Progettazione *software*

Qualità architetture 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

Dipartimento di Informatica, Università di Pisa15/41

Progettazione *software*

Qualità architetture 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

Dipartimento di Informatica, Università di Pisa14/41

Progettazione *software*

Qualità architetture da ricercare – 3

- Disponibilità**
 - Necessita 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

Dipartimento di Informatica, Università di Pisa16/41




Progettazione *software*

Qualità architeturali 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

Dipartimento di Informatica, Università di Pisa

17/41




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

Dipartimento di Informatica, Università di Pisa

19/41



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

18/41




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**

Dipartimento di Informatica, Università di Pisa

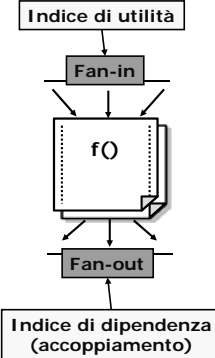
20/41



Progettazione *software*


Accoppiamento

- ❑ **Proprietà esterna 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 \Rightarrow massimizzare
 - SFOUT è indice di dipendenza \Rightarrow minimizzare
- ❑ **Buona progettazione**
 - Componenti con SFIN elevato



Dipartimento di Informatica, Università di Pisa

21/41




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 riusate per non perderne il valore
- ❑ **Puro costo nel breve periodo**
 - Risparmio (quindi investimento) nel medio termine

Dipartimento di Informatica, Università di Pisa

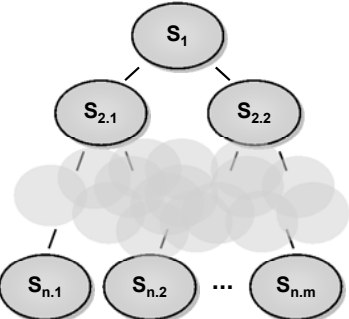
23/41



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



Dipartimento di Informatica, Università di Pisa

22/41



Progettazione *software*

Framework

- ❑ **Insieme integrato di componenti SW prefabbricate**
 - Nel mondo pre-OO erano chiamate librerie
 - Sono *bottom-up* perché fatti di codice già sviluppato
 - Sono *top-down* perché impongono uno stile architetturale
- ❑ **Utilissimi come base facilmente riusabile di diverse applicazioni entro un dato dominio**
 - Molti importanti esempi nel mondo J2EE
 - Spring (<http://www.springframework.org/about>) per architetture di *business* con MVC
 - Struts (<http://struts.apache.org/>) per Web Apps in stile MVC
 - Swing per GUI, ecc.

Dipartimento di Informatica, Università di Pisa

24/41



Progettazione *software*

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 otterremo è inferiore al costo
 - Eccessiva esposizione di dettagli causa accoppiamento
- ❑ **Cercare un buon bilanciamento**
 - **Più semplici le componenti più complessa la loro interazione**

Dipartimento di Informatica, Università di Pisa25/41




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
- ❑ **A livello sistema servono *pattern* architetturali**

Dipartimento di Informatica, Università di Pisa27/41




Progettazione *software*

Schemi (*pattern*) 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)**

Dipartimento di Informatica, Università di Pisa26/41



Progettazione *software*

Pattern architetturali – 1

- ❑ **Architettura "three-tier" (a livelli)**
 - **Strato della presentazione (GUI)**
 - **Strato della logica operativa (*business logic*)**
 - **Strato dell'organizzazione dei dati (*database*)**
- ❑ **Variante multilivello (pila OSI e TCP/IP)**
- ❑ **Architettura produttore-consumatore**
 - **Collaborazione a *pipeline***

Dipartimento di Informatica, Università di Pisa28/41

Progettazione software

Esempi – 1

Architettura multilivello

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

Architettura a oggetti

Dipartimento di Informatica, Università di Pisa

29/41

Progettazione software

Esempi – 2

Architettura Fat-client

Architettura Thin-client

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

Dipartimento di Informatica, Università di Pisa

31/41

Progettazione software

Pattern architetturali – 2

- **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 di scambio senza *server* intermedio

Dipartimento di Informatica, Università di Pisa

30/41

Progettazione software

Esempi – 3

Client Tier

Interfaccia utente

Swing Client

↓

Middle Tier

Logica di *business* o di applicazione

Hibernate

Java Transaction API

POJO

Java Messaging Service

JDBC

Java Data Objects

Java NDI

Structural Patterns

Integration Patterns

Enterprise Patterns

↓

EIS Tier

Modello dei dati e persistenza


Databases

XML

Third Party Data source

Dipartimento di Informatica, Università di Pisa

32/41




Progettazione *software*

Architettura di relazione: 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

Dipartimento di Informatica, Università di Pisa

33/41




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 architetture diverse

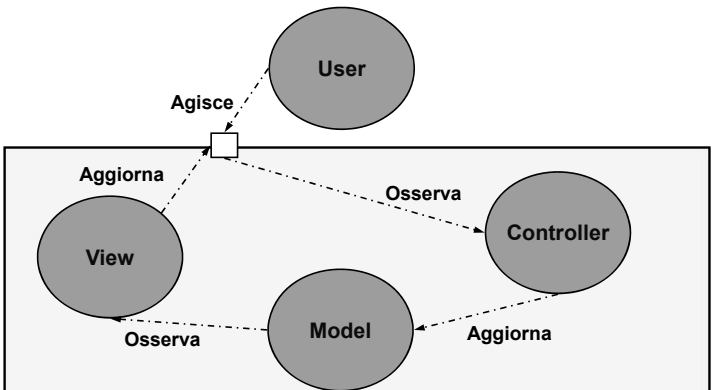
Dipartimento di Informatica, Università di Pisa

35/41



Progettazione *software*

Architettura di relazione: MVC – 2



The diagram illustrates the MVC-2 architecture. A User component (circle) is connected to a central square component. From this square, three dashed arrows labeled 'Agisce' point to the View, Controller, and Model components (circles). The View and Controller are connected to each other with a dashed arrow labeled 'Osserva'. The Controller and Model are connected to each other with a dashed arrow labeled 'Aggiorna'. The Model and View are connected to each other with a dashed arrow labeled 'Osserva'.

Dipartimento di Informatica, Università di Pisa

34/41




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 *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


36/41

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 in modo certo e disciplinato
 - Tracciamento per attribuire requisiti alle unità
 - Per definire le configurazioni ammissibili del sistema
- ❑ **Definire gli strumenti per le prove di unità**
 - Casi di prova e componenti ausiliarie per la verifica unitaria e di integrazione


Dipartimento di Informatica, Università di Pisa37/41

Progettazione *software*

Stati di progresso per SEMAT – 1

- ❑ **Architecture selected**
 - Selezione di una architettura tecnicamente adatta al problema: accordo sui criteri di selezione
 - Selezione delle tecnologie necessarie
 - Decisioni su *buy, build, make*
- ❑ **Demonstrable**
 - Dimostrazione delle principali caratteristiche dell'architettura: gli *stakeholder* concordano
 - Decisione sulle principali interfacce e configurazioni di sistema

Dipartimento di Informatica, Università di Pisa39/41

Progettazione *software*

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

Dipartimento di Informatica, Università di Pisa38/41

Progettazione *software*

Stati di progresso per SEMAT – 2

- ❑ **Usable**
 - Il sistema è utilizzabile e ha le caratteristiche desiderate
 - Il sistema può essere operato dagli utenti
 - Le funzionalità e le prestazioni richieste sono state verificate e validate
 - La quantità di difetti residui è accettabile
- ❑ **Ready**
 - La documentazione per l'utente è pronta
 - Gli *stakeholder* hanno accettato il prodotto e vogliono che diventi operativo

Dipartimento di Informatica, Università di Pisa40/41



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**