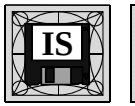
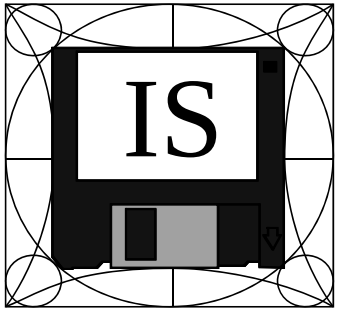


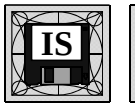
Progettazione *software*



Ingegneria del Software

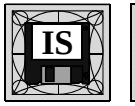
V. Ambriola, G.A. Cignoni
C. Montangero, L. Semini

Aggiornamenti di: T. Vardanega (UniPD)



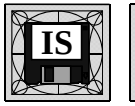
Contenuti

- La progettazione
- L'architettura
- Progettazione architetturale
- Progettazione di dettaglio



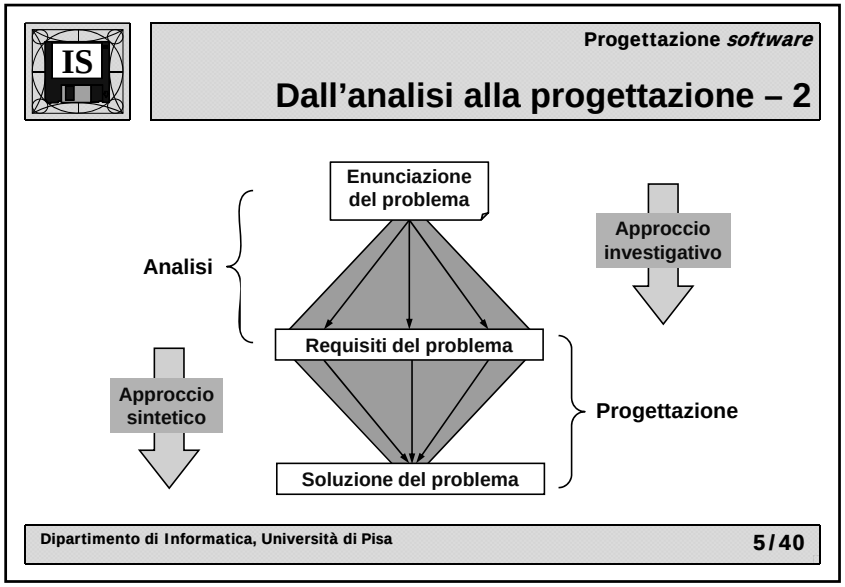
Progettare prima di produrre

- La progettazione <u>precede</u> la produzione
  - Approccio industriale e metodo ingegneristico
  - Costruzione *a priori* perseguendo la «correttezza per costruzione»
  - Prima analisi poi progettazione infine produzione
- 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)



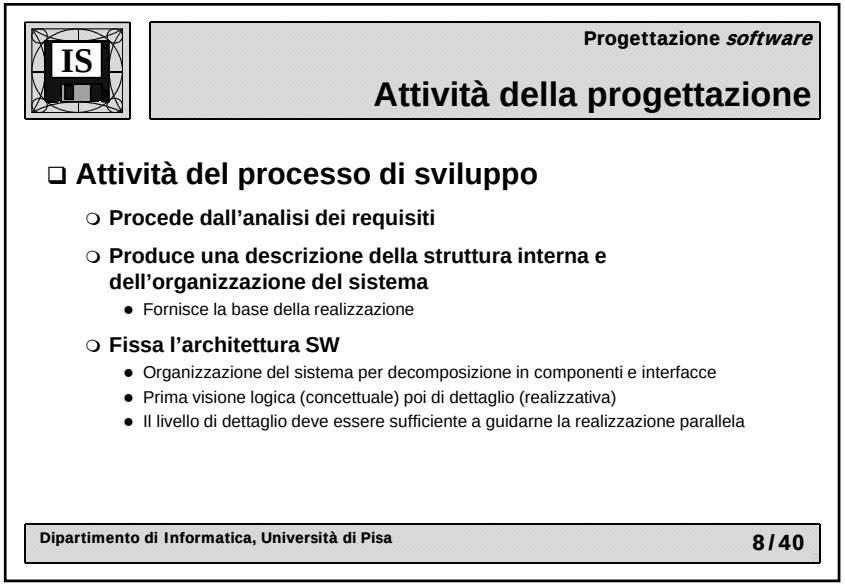
Dall'analisi alla progettazione – 1

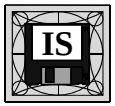
- Analisi: quale è il problema, quale la cosa giusta da fare?
  - Comprensione del dominio
  - Discernimento di obiettivi, vincoli e requisiti
  - Approccio investigativo
- Progettazione: come farla giusta?
  - Descrizione di una soluzione soddisfacente per tutti gli *stakeholder*
  - Il codice non esiste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


Progettazione *software*
Dall'analisi alla progettazione – 2
Enunciazione del problema
Analisi
Approccio investigativo
Requisiti del problema
Approccio sintetico
Progettazione
Soluzione del problema
Dipartimento di Informatica, Università di Pisa
5 / 40


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 Pisa
6 / 40

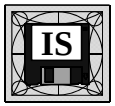

Progettazione *software*
Obiettivi della progettazione
Soddisfare i requisiti di qualità
Definire l'architettura del prodotto
Impiegando componenti con specifica chiara e coesa
Realizzabili con risorse date e costi fissati
Struttura che facilita eventuali cambiamenti futuri
L'architettura è essenziale al raggiungimento degli obiettivi di progetto
Diventa essa stessa obiettivo
Dipartimento di Informatica, Università di Pisa
7 / 40


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 realizzazione parallela
Dipartimento di Informatica, Università di Pisa
8 / 40



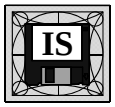
Punti di vista

- 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



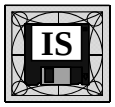
Arte e architettura

- Intorno al 1915, lo scrittore Herbert George 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



Definizioni di architettura – 1

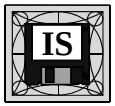
- La nozione di "architettura *software*" appare la prima volta nello stesso evento in cui venne riconosciuta la disciplina del *software engineering*
- Tuttavia fino alla fine degli anni '80 il termine "architettura" viene applicato prevalentemente al sistema fisico
- Nel 1992 D. Perry and A. Wolf propongono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



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*

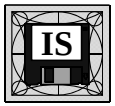
[B. Boehm, et al., '95]



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***

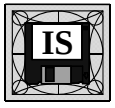
[P. Kruchten: The Rational Unified Process, '99]



Definizioni di architettura – 4

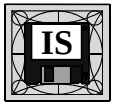
- La decomposizione del sistema in componenti
- L' organizzazione di tali componenti
  - Definizione di ruoli, responsabilità, interazioni
- Le interfacce necessarie all'interazione tra le componenti tra loro e con l'ambiente
- I paradigmi di composizione delle componenti
  - Regole, criteri, limiti,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

ISO/IEC/IEEE 42010:2011 Systems and software engineering Architecture description



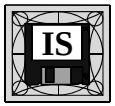
Qualità architetturali da ricercare – 1

- 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



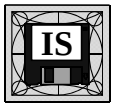
Qualità architetturali da ricercare – 2

- 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



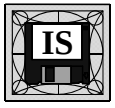
Qualità architetturali da ricercare – 3

- **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



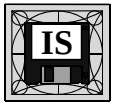
Qualità architetturali da ricercare – 4

- **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



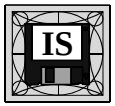
Semplicità

- **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*"



Incapsulazione

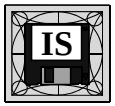
- **Le componenti sono "*black box*"**
  - I suoi clienti ne conoscono solo l'interfaccia
- **La loro specifica nasconde**
  - Gli algoritmi e le strutture dati usati all'interno
- **Benefici**
  - L'esterno non può fare assunzioni sull'interno
  - Cresce la manutenibilità
  - Diminuendo le dipendenze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



Progettazione *software*

Coesione

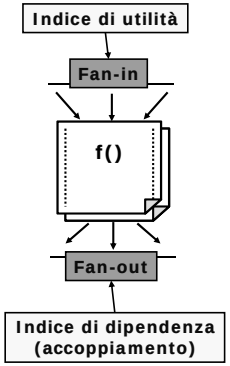
- **Proprietà interna di singole componenti**
  - **Funzionalità "vicine" devono stare nella stessa componente**
    - La modularità spinge a decomporre il grande in piccolo
    - La ricerca di coesione aiuta sia a decomporre che a porre un limite inferiore alla decomposizione
  - **Va massimizzata**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

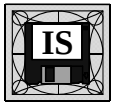


Progettazione *software*

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**

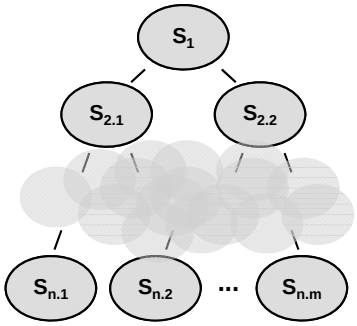


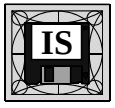


Progettazione *software*

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

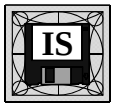




Progettazione *software*

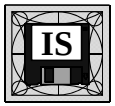
Riuso

- **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**



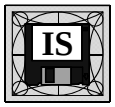
Framework

- **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**
- **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.



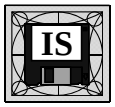
Ricetta generale

- **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**



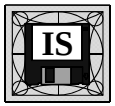
Schemi (*pattern*) architetturali

- **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)**



Design pattern

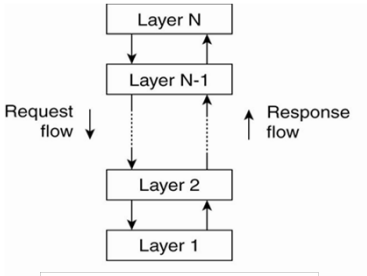
- **Soluzione progettuale a problema ricorrente**
  - **Definisce una funzionalità lasciando gradi di libertà d'uso**
    - Ha corrispondenza precisa nel codice sorgente
  - **Il corrispondente architetturale degli algoritmi**
    - Che invece specificano procedimenti di soluzione
  - **Concetto promosso 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**



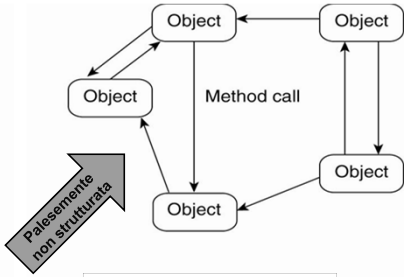
Pattern architetturali – 1

- **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***

Esempi – 1

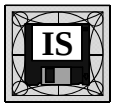


Architettura multilivello



Architettura a oggetti

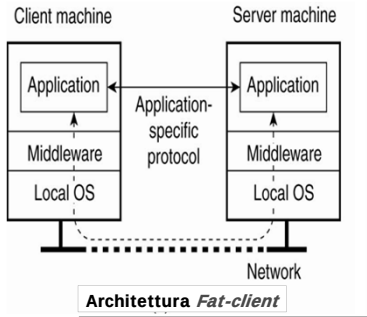
Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.



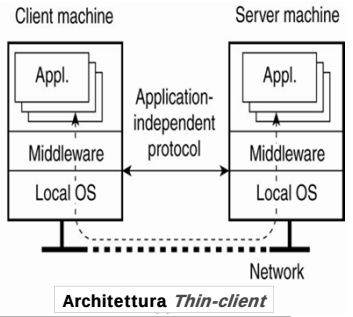
Pattern architetturali – 2

- **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**

Esempi – 2



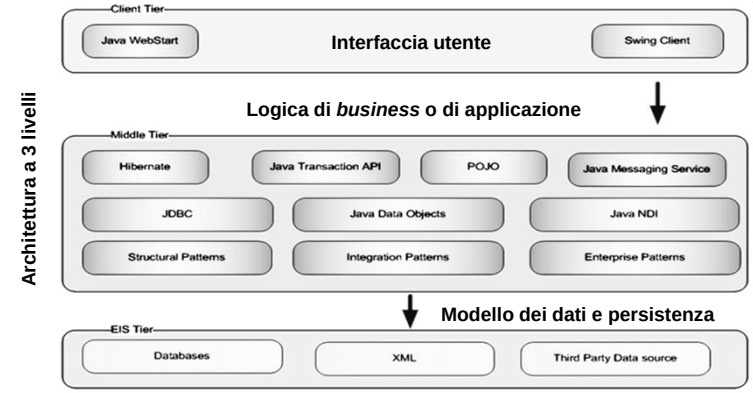
Architettura *Fat-client*



Architettura *Thin-client*

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

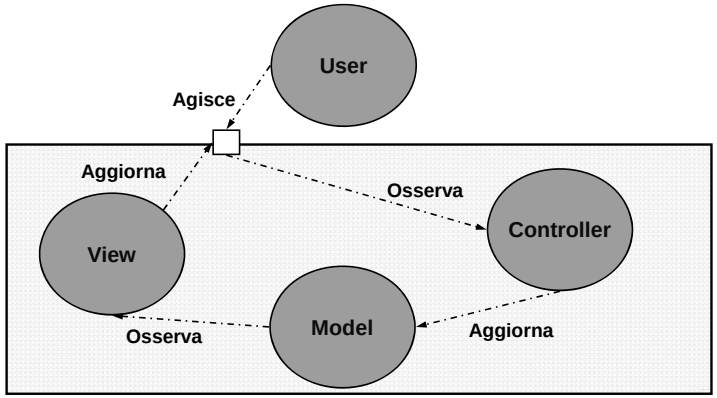
Esempi – 3



Architettura di relazione: MVC – 1

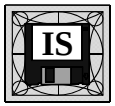
- ***Model***
  - Incorpora il modello dei dati e le operazioni su di essi
- ***View***
  - Incorpora la logica della presentazione
  - Riceve input dall'utente e mostra in output l'esito delle azioni conseguentemente svolte dal Controller sul Model
- ***Controller***
  - Incorpora la logica di controllo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

Architettura di relazione: MVC – 2



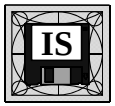
Linguaggi di descrizione architetturale

- **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



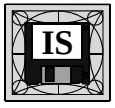
Progettazione di dettaglio: attività

- **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 *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**



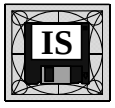
Progettazione di dettaglio: obiettivi

- **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 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**



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**



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**