INGEGNERIA DEL SOFTWARE
AA. 2016/2017

Laura Semini
Dipartimento di Informatica
Università di Pisa

Outline della lezione

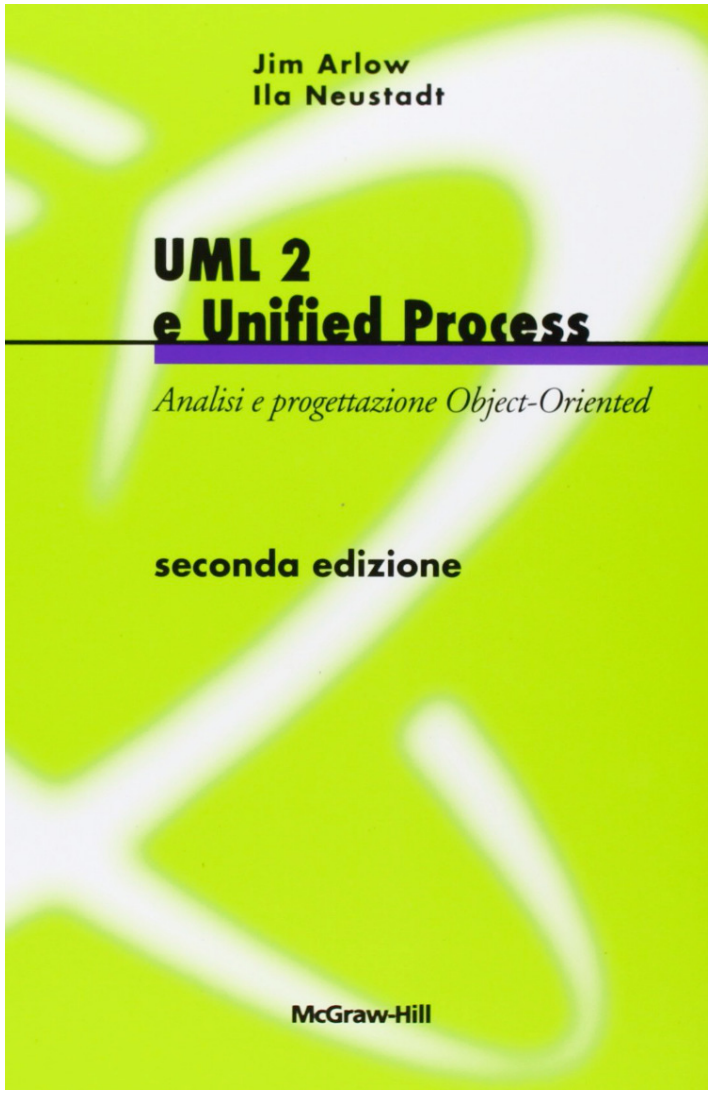
- Dettagli tecnici: libri, lezioni, esami, ecc.
- Ingegneria del Software:
  - Cos'è
  - Perché e quando è nata questa disciplina

Dettagli tecnici: libri, lezioni, esami…

- **Pagine Web**
  - Pagina comune a Ing Sw A e B
    - Per il materiale didattico
  - Pagina dei singoli corsi A/B
    - Per risultati prove, etc
- Accessibili
  - Da didawiki
  - Dalla mia pagina web www.di.unipi.it/~semini

Dettagli tecnici: libri, lezioni, esami…

- **Libro 1**
  - J. Arlow, I. Neustadt, *UML 2 e Unified Process, Seconda Edizione italiana*, McGraw-Hill, 2006.



Dettagli tecnici: libri, lezioni, esami…

- **Libro 2**
  - A. Binato, A. Fuggetta, L. Sfardini, *Ingegneria del Software - Creatività e metodo*, Addison Wesley, 2006.
    - **(alcuni capitoli)**

Dettagli tecnici: libri, lezioni, esami…

- **Dispense (scaricabili da didawiki)**
- C. Montangero, L. Semini, *Dispensa di architettura e progettazione di dettaglio*.
  - Utile quando si comincerà a parlare di progettazione (circa metà corso).
- C. Montangero, L. Semini (a cura di), *Il controllo del software - verifica e validazione*.
  - Utile nelle ultime lezioni del corso
- Esercizi:
  - Compiti degli anni passati.
  - Esercizio più datati: V. Ambriola, C. Montangero, L. Semini, *Esercizi di Ingegneria del Software*.
- + … altro materiale che verrà reso disponibile quando necessario.

Modalità d'esame

- Scritto: appello o prove in itinere
  - Il voto delle scritto vale solo per l'appello, le prove in itinere esonerano dallo scritto per tutto l'anno accademico
- Ammissione all'orale
  - Prove itinere: voto minimo 16, media minima 16
  - Appello: voto minimo 16
- Problemi disponibili in anticipo

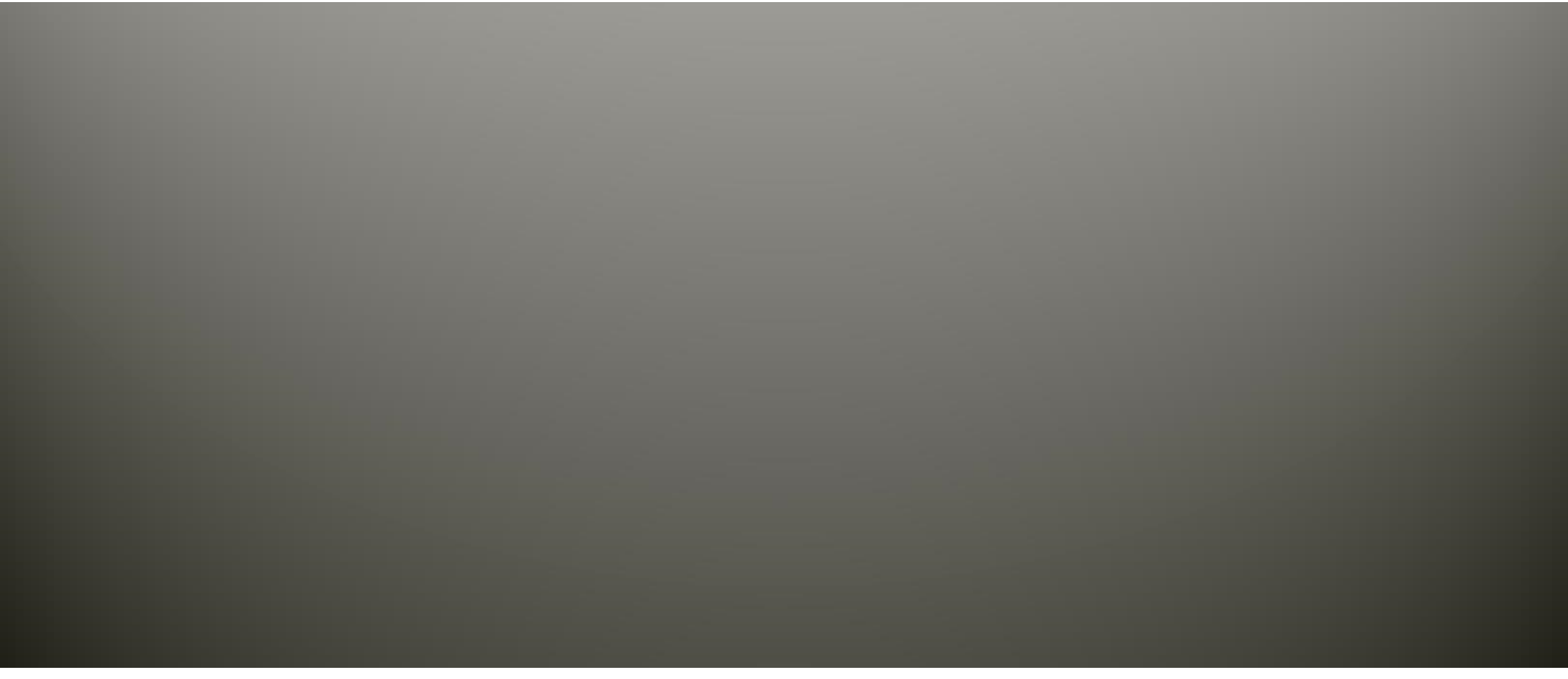
Obiettivi di apprendimento

- Introduzione alle tecniche di modellazione in ingegneria del software.
- ***Conoscenze***. Lo studente conoscerà i principali modelli di processi di sviluppo software, e le tecniche di modellazione proprie delle varie fasi.
- ***Capacità***. Lo studente saprà utilizzare notazioni di modellazione come UML2 per l'analisi dei requisiti e la progettazione sia architettonica sia di dettaglio di un sistema software.

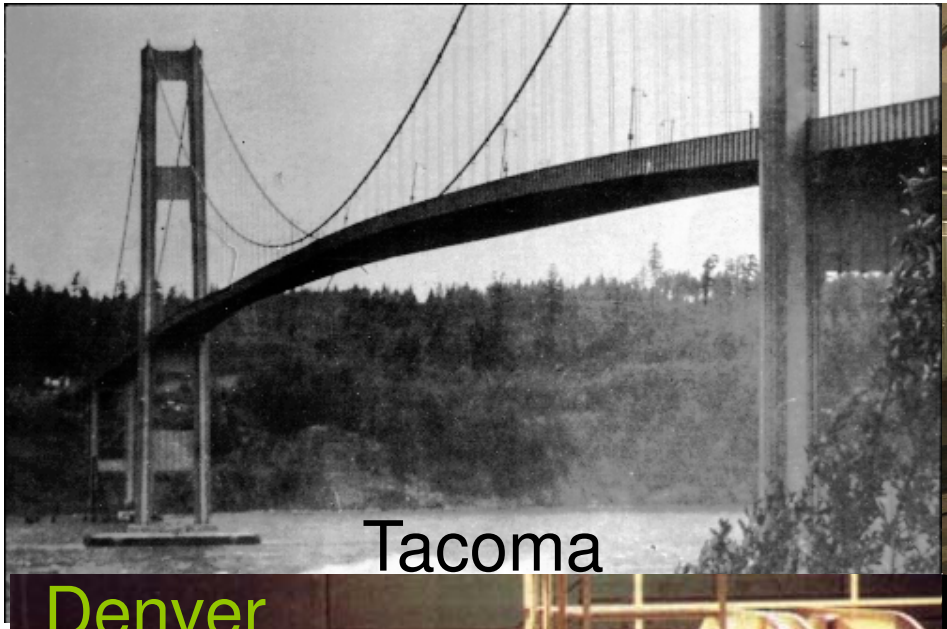
Programma

- *Processo di sviluppo software*
  - Problemi della produzione del software. Modelli di ciclo di vita.
- *Analisi del domino e dei requisiti*
  - modelli statici (classi, casi d'uso) e dinamici (attività, macchine a stati, narrazioni). Problem Frames.
- *Architetture software*:
  - viste strutturali, comportamentali, logistiche. Stili architettonici.
- *Progettazione di dettaglio*:
  - Progettazione di componenti orientata agli oggetti. Design Patterns
- *Verifica*:
  - Progettazione delle prove - criteri funzionali e strutturali.
- *Standard per la modellazione:*
  - Unified Modelling Language (UML 2).

Qual è il problema da risolvere?



Chi è l'intruso?

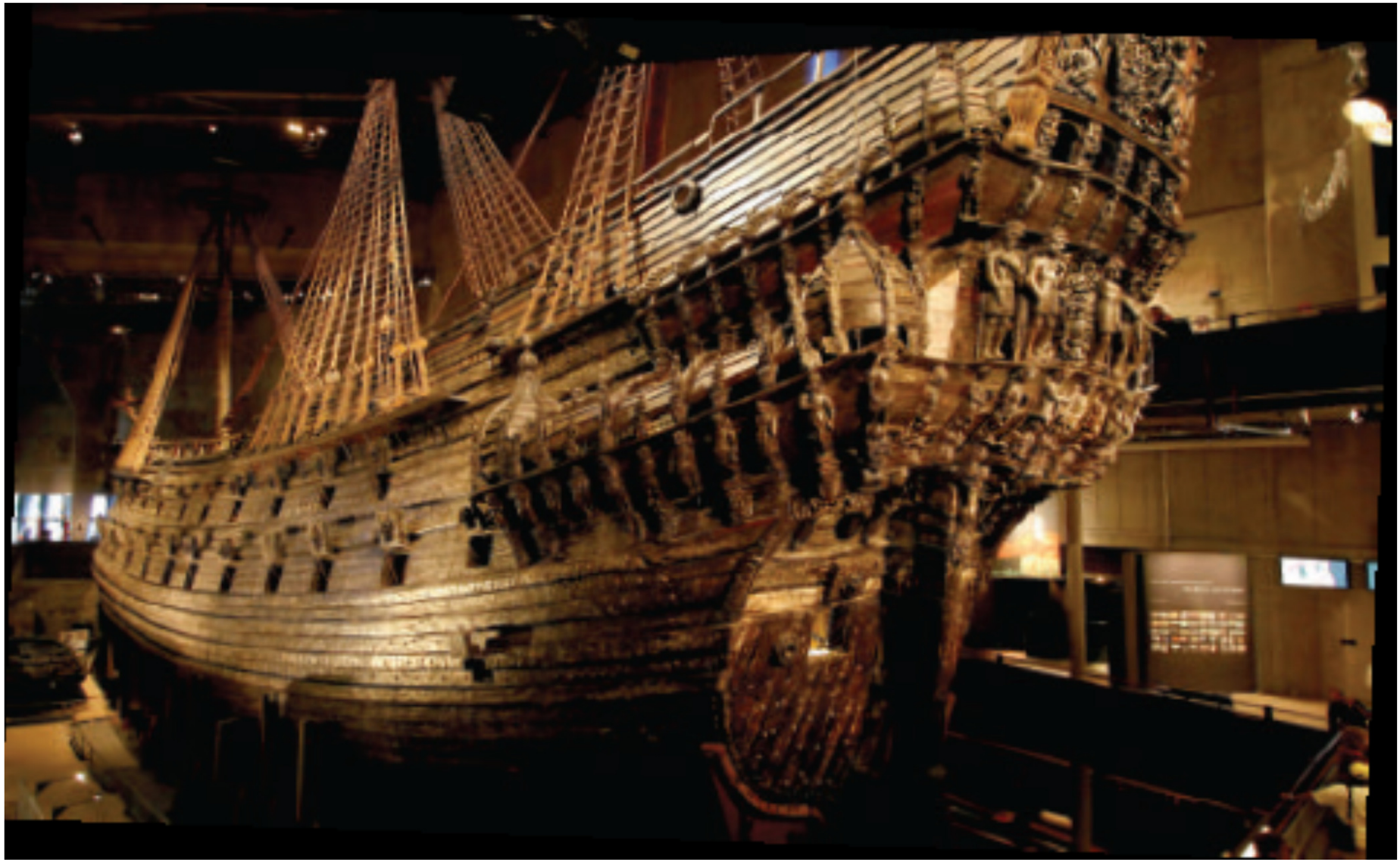

Tacoma


Paris


Denver


Stockholm

Vasa: orgoglio e pregiudizio (1625)



Tacoma Bridge, 1940

- https://www.youtube.com/watch?v=XggxeuFDaDU

Denver Airport (1995 spento nel 2005)

- Sistema di smistamento dei bagagli
- 35 Km di rete, 4000 carrelli, 5000 "occhi", 56 lettori
- $ 193 000 000 di investimento
- Risultati
  - Inaugurazione dell'aereoporto ritardata 16 mesi (a 1 milione $ al giorno)
  - sforamento di 3,2 miliardi di dollari rispetto ai preventivi
- Progettazione difettosa
  - jams (mancanza di sincronizzazione)
  - No fault tolerance
  - si perdeva traccia di quali carrelli fossero pieni e quali vuoti dopo un riavvio dovuto a un jam
- Dopo anni di tentativi di "aggiustarlo"
  - Staccata la spina nel 2005

Il caso Therac- 25 (1985-1987)

- *Canada- USA:* 3 persone uccise per sovradosaggi di radiazioni
- Problema causato da editing troppo veloce dell'operatore e mancanza di controlli sui valori immessi.
- Le cause:
  - errori nel sistema SW, e di interfacciamento SW/ HW (erronea integrazione di componenti SW preesistenti nel Therac- 20).
- Poca robustezza
- Difetto latente

Il sistema antimissile Patriot (1991)

- Una caserma a Dhahran (Arabia Saudita) colpita per un difetto nel sistema di guida: 28 soldati americani morti.
- Concepito per funzionare ininterrottamente per un massimo di 14 h.
  - Fu usato per 100 h: errori nell'orologio interno del sistema accumulati al punto da rendere inservibile il sistema di tracciamento dei missili da abbattere.
- Scarsa robustezza.

London ambulance service (1992)

- Sistema per gestire il servizio ambulanze
- Ottimizzazione dei percorsi, guida vocale degli autisti
- Risultati
  - 3 versioni, costo totale: 11 000 000 Euro
  - L'ultima versione abbandonata dopo soli 3 giorni d'uso
- Analisi errata del problema:
  - interfaccia utente inadeguata
  - poco addestramento utenti
  - sovraccarico non considerato
  - nessuna procedura di backup
  - scarsa verifica del sistema

Ariane 5 (1996)

- http://it.youtube.com/watch?v=kYUrqdUyEpI
- Il sistema, progettato per l'Ariane 4, tenta di convertire la velocità laterale del missile dal formato a 64 bit al formato a 16 bit. Ma l'Ariane 5 vola molto più velocemente dell'Ariane 4, per cui il valore della velocità laterale è più elevato di quanto possa essere gestito dalla routine di conversione.
- *Overflow*, spegnimento del sistema di guida, e trasferimento del controllo al 2° sistema di guida, che però essendo progettato allo stesso modo era andato in tilt nella medesima maniera pochi millisecondi prima.
- Test con dati vecchi.
  - Fu necessario quasi un anno e mezzo per capire quale fosse stato il malfunzionamento che aveva portato alla distruzione del razzo.

Un successo!!!! Linea 14 Metro Parigi (1998 con estensione nel 2003)

- La linea 14 della metropolitana di Parigi
  - Prima linea integralmente automatizzata .
- Nome di progetto, Météor: Metro Est-Ovest Rapide
  - 8 km. 7 stazioni. 19 treni. Intervallo tra 2 treni: 85 secondi.
  - Siemens Transportation Systems
  - B-method di Abrial.
  - Abstract achines
  - generazione di codice
    - ADA, C, C++.

Quindi ???

- Anche gli ingegneri con decenni, se non secoli, di tradizione alle spalle possono sbagliare
- Anche i produttori di software possono aver successo
- Anche i produttori di software devono imparare dagli errori
- Anche i produttori di software devono diventare ingegneri (del software)

Perché ingegneria del software

- Per realizzare sistemi complessi
- Problemi incontrati nello sviluppo di sistemi software complessi:
  - progetti in ritardo rispetto ai termini prefissati
  - sforamento del budget
  - scarsa affidabilità
  - scarse prestazioni
  - manutenzione ed evoluzione difficile
  - alta percentuale di progetti software cancellati

Quando nasce

- Contesto degli anni '60
  - DA: software sviluppato informalmente
    - ad es., per risolvere sistemi di equazioni
  - A: grandi sistemi commerciali
    - OS 360 per IBM 360 (milioni di righe di codice)
    - sistemi informativi aziendali, per gestire tutte le informazioni delle funzioni aziendali
  - Dalla programmazione individuale alla programmazione di squadra
  - Necessario un nuovo approccio, ingegneristico, con strumenti e tecniche opportuni
- Due termini coniati verso la fine degli anni '60
  - ***crisi del software*** – problemi incontrati nello sviluppo di sistemi sw complessi
  - ***ingegneria del software –*** soluzione alla crisi del software
- 1968, Conferenza di Garmish

Standish Group report 1994

- Progetti software completati in tempo 16,2%
- in ritardo (il doppio del tempo): 52,7%
  - Difficoltà nelle fasi iniziali dei progetti
  - Cambi di piattaforma e tecnologia
  - Difetti nel prodotto finale
- abbandonati: 31,1%
  - Per obsolescenza prematura
  - Per incapacità di raggiungere gli obiettivi
  - Per esaurimento dei fondi

Le cause di abbandono

1. Requisiti incompleti
2. Scarso coinvolgimento degli utenti
3. Mancanza di risorse
4. Attese irrealistiche
5. Mancanza di supporto esecutivo
6. Modifiche a specifiche e requisiti
7. [...]
8. [...]
9. [...]
10. Ignoranza tecnologica

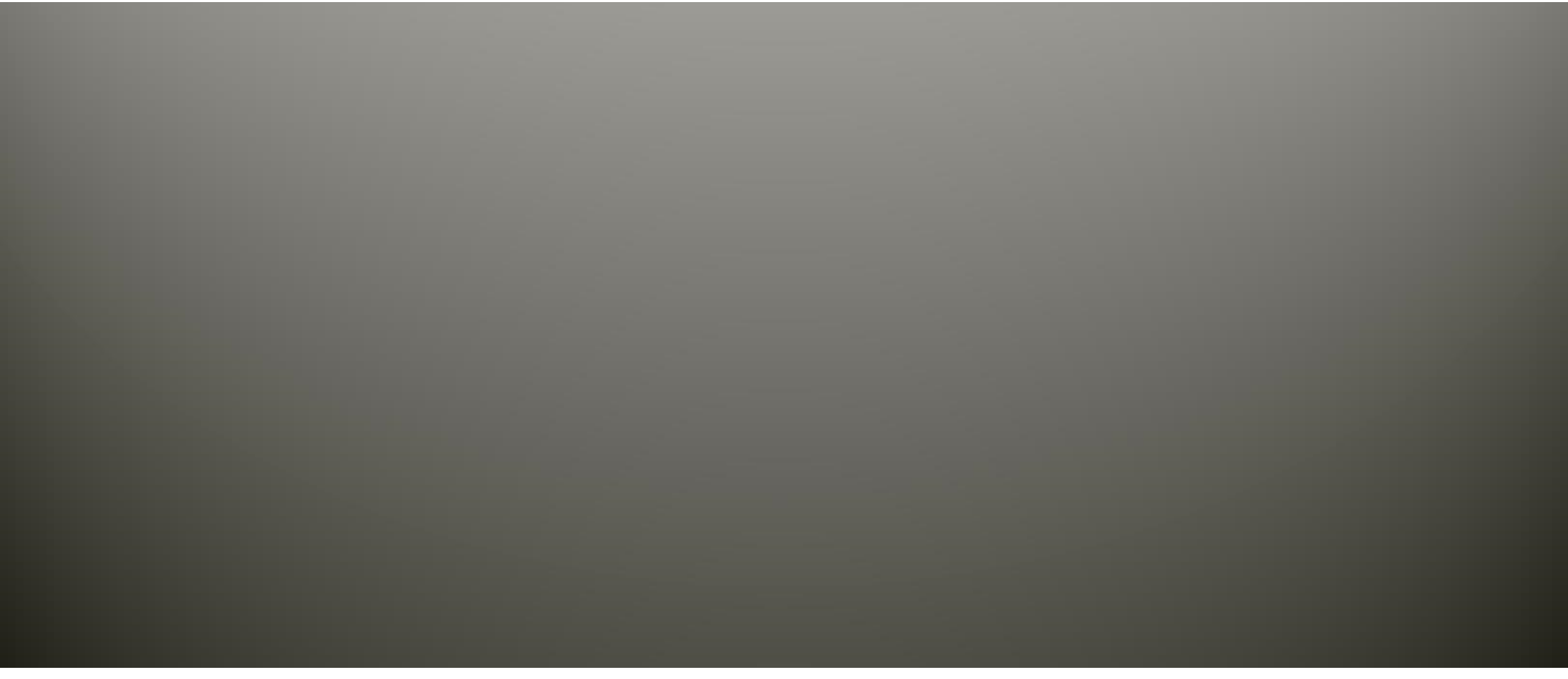
Standish Group: reasons for success

Top Ten Reasons for Success

- ☑ 1. User Involvement
- ☑ 2. Executive Management Support
- ☑ 3. Clear Business Objectives
- ☑ 4. Optimizing Scope
- ☑ 5. Agile Process
- ☑ 6. Project Manager Expertise
- ☑ 7. Financial Management
- ☑ 8. Skilled Resources
- ☑ 9. Formal Methodology
- ☑ 10. Standard Tools and Infrastructure

Software engineering

Definizioni, temi.



Software Engineering: definizione IEEE

- **L'approccio sistematico allo sviluppo, all'operatività, alla manutenzione e al ritiro del software**
  (Glossario IEEE)

- Il software è un prodotto con il suo ciclo vitale
- Approccio sistematico

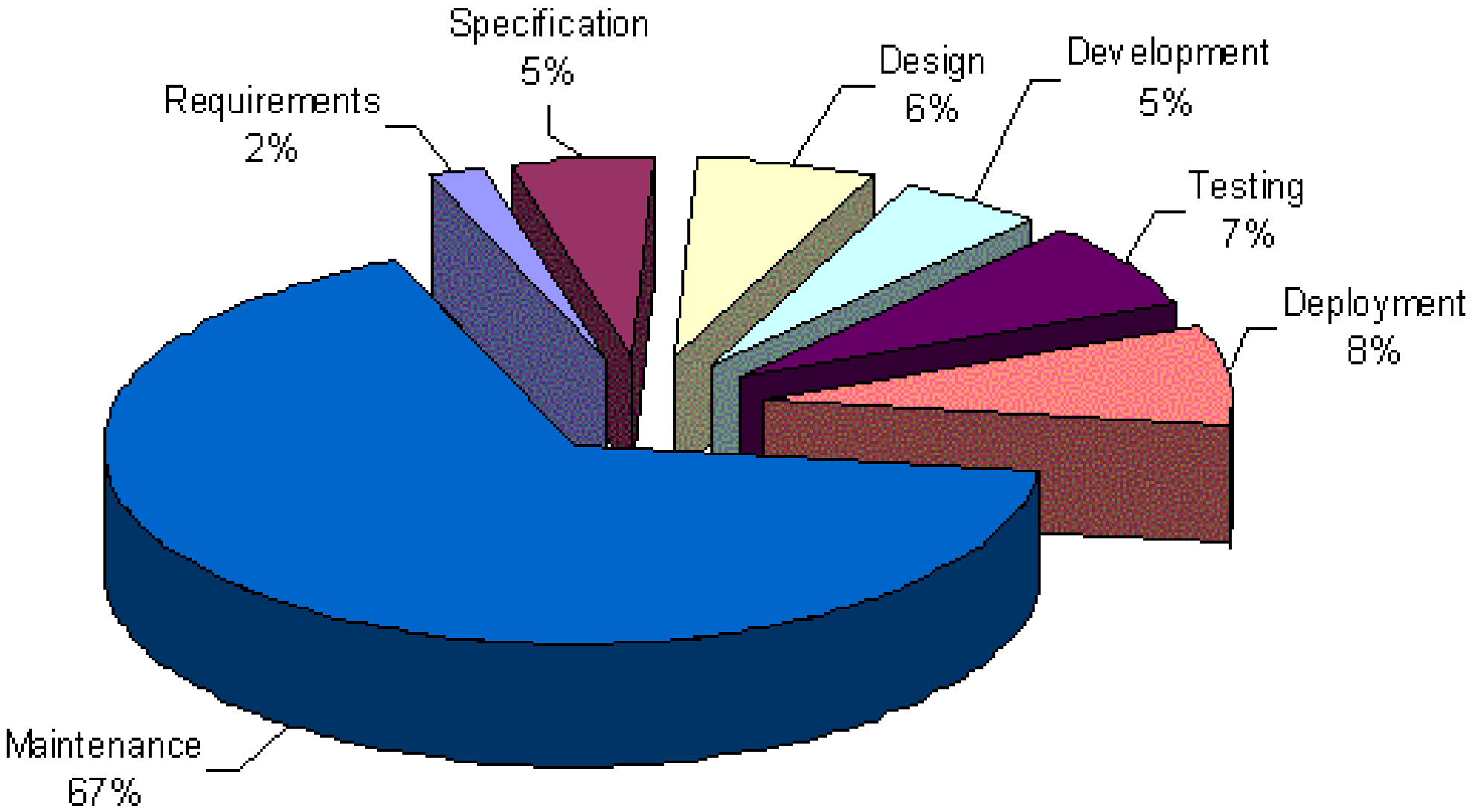
Software Engineering: definizione di Fairley

- **La disciplina tecnologica e gestionale per la produzione sistematica e la manutenzione di prodotti software sviluppati e modificati con tempi e costi preventivati**

  (R. Fairley, 1985)

- Disciplina gestionale: costi, tempi, risorse
- Controllo della qualità: costi e risultati definiti

Costi relativi nello sviluppo di un sistema sw



Temi di ingegneria del software

- Processo software
- Realizzazione di sistemi software
- Qualità del software

Processo software

- Organizzazione e gestione dei progetti
- Metodi di composizione dei gruppi di lavoro
- Strumenti di pianificazione, analisi, controllo
- Cicli di vita del software
- Definizione e correlazione delle attività
- Modelli ideali di processo di sviluppo

Realizzazione di sistemi sw

- Strategie di analisi e progettazione
  - Tecniche per la comprensione e la soluzione di un problema
  - Top-down, bottom-up, progettazione modulare, OO
- Linguaggi di specifica e progettazione
  - Strumenti per la definizione di sistemi software
  - Reti di Petri, Z, OMT, UML
- Ambienti di sviluppo
  - Strumenti per analisi, progettazione e realizzazione
  - Strumenti tradizionali, CASE, CAST, RAD

Qualità del software

- Modelli di qualità
  - Definizione di caratteristiche della qualità
- Metriche software
  - Unità di misura, scale di riferimento, strumenti
  - Indicatori di qualità
- Metodi di verifica e controllo
  - Metodi di verifica, criteri di progettazione delle prove
  - Controllo della qualità, valutazione del processo di sviluppo

Metodi ingegneristici e non mito

Metodi ingegneristici e non mito (Pressman)

- La malattia del software si può in gran parte far risalire a una mitologia nata agli inizi dell'informatica.
- I miti del software hanno diffuso disinformazione e confusione:
  - In genere hanno la forma di affermazioni ragionevoli (talvolta non prive di elementi di verità), sono intuitivamente condivisibili e sono spesso stati diffusi da esperti del settore.

Miti del management (1/2)

I manager con responsabilità legate al software, come in tutti gli altri campi, sono spesso sotto pressione per il budget, le scadenze e la qualità.

- **Mito** Abbiamo già interi volumi di standard e procedure da seguire nello sviluppo. Non c'è forse tutto l'indispensabile?
- **Realtà** Gli standard sono applicati? I programmatori li conoscono? Quegli standard riflettono i metodi moderni di sviluppo del software? Sono completi? In molti casi reali, la risposta a tutte queste domande è "No".
- **Mito** Abbiamo i più moderni strumenti di sviluppo. Dopo tutto, acquistiamo sempre i computer più recenti.
- **Realtà** Lo sviluppo di software di alta qualità richiede ben più dell'ultimo modello di computer. Gli strumenti di CASE (ingegneria del software assistita dal computer) sono più importanti dell'hardware per ottenere alta qualità e produttività; tuttavia, la più parte degli sviluppatori non se ne serve realmente.

Miti del management (2/2)

- **Mito** Se siamo in ritardo, possiamo sempre recuperare aumentando il numero di programmatori.
- **Realtà** Lo sviluppo di software non è un processo meccanico come la produzione manifatturiera. Per citare Brooks (1975), "aggiungere persone a un progetto software in ritardo lo rallenta ulteriormente". A prima vista, questa affermazione può suonare controintuitiva, ma si deve considerare che i nuovi arrivati devono essere addestrati da chi lavorava già al progetto, togliendo così risorse allo sviluppo effettivo. È possibile inserire nuovo personale in un progetto, ma solo in modo pianificato e coordinato.
- **Mito** Se decido di far realizzare un progetto software a una terza parte, posso restare tranquillo perché tutto il lavoro verrà svolto esternamente.
- **Realtà** Se un'organizzazione non sa come gestire e controllare internamente i progetti software, inevitabilmente incontrerà problemi se deciderà di fornire all'esterno lo sviluppo.

Miti della clientela (1/2)

Il cliente che chiede un prodotto software può essere il collega della stanza accanto, un reparto dell'azienda, il settore vendite o un'azienda esterna che lo ha commissionato.

- **Mito** Un'affermazione generica degli scopi è sufficiente per cominciare a scrivere i programmi; i dettagli si possono trattare in seguito.

- **Realtà** Una insufficiente descrizione preliminare è la causa principale di fallimento di un progetto software. La descrizione formale e dettagliata del dominio dei dati, delle funzioni, delle prestazioni, dell'interfaccia, dei vincoli progettuali e dei criteri di validazione è essenziale. Tutti questi aspetti si possono stabilire solo dopo una approfondita comunicazione fra cliente e sviluppatore.

Miti della clientela (2/2)

- **Mito** I requisiti di un progetto mutano di continuo, ma i mutamenti si gestiscono agevolmente grazie alla flessibilità del software.

- **Realtà** È vero che i requisiti cambiano, ma l'effetto delle modifiche varia secondo la fase in cui vengono introdotte.
  - Le richieste di modifiche nelle prime fasi si soddisfano facilmente: il cliente ha modo di esaminare i requisiti e di richiedere modifiche, senza effetti pesanti sui costi.
  - L'effetto di modifiche richieste durante la fase di progettazione cresce rapidamente. Le risorse sono state già allocate e si è stabilita la cornice del progetto. Una modifica può comportare rivolgimenti che richiedono nuove risorse e correzioni importanti al progetto, cioè costi supplementari.
  - Le modifiche alla funzionalità, alle prestazioni, all'interfaccia o ad altre caratteristiche durante l'implementazione (stesura del codice e collaudo) hanno un effetto pesante sui costi.
  - Le modifiche richieste dopo l'entrata in esercizio del software possono rivelarsi più costose di un ordine di grandezza rispetto alle stesse modifiche richieste nelle fasi precedenti.

Miti del programmatore (1/2)

I miti ancora popolari fra i programmatori sono stati alimentati in 50 anni di cultura della programmazione.

- **Mito** Una volta scritto e messo in opera il programma, il nostro lavoro è finito.
- **Realtà** Alcuni dati relativi all'industria indicano che tra il 60 e l'80% del lavoro speso su un programma avviene dopo la consegna della prima versione al cliente.

- **Mito** Fino a quando il programma non è in condizione di essere eseguito, non c'è modo di valutarne la qualità.
- **Realtà** Uno dei metodi più efficaci di esame della qualità del software, la *revisione tecnica formale*, si applica già dall'inizio di un progetto. Le revisioni sono un "filtro della qualità", rivelatosi più efficace del collaudo nel rilevare alcuni tipi di errori del software.

Miti del programmatore (2/2)

- **Mito** Il solo prodotto di un progetto concluso è il programma funzionante.
- **Realtà** Un programma funzionante è solo una parte di una configurazione software che comprende più elementi. La documentazione è il fondamento di uno sviluppo riuscito e, cosa più importante, è la guida dell'attività di manutenzione.

- **Mito** L'ingegneria del software ci farà scrivere un'inutile e voluminosa documentazione che inevitabilmente rallenterà le cose.
- **Realtà** L'ingegneria del software non prevede la creazione di documenti. Si occupa di creare qualità. Una migliore qualità porta a minor lavoro. E un minor lavoro porta a tempi di consegna più rapidi. È stato detto: "Prima cominci a stendere il codice, più tempo impiegherai a terminare il programma".