Introduzione alla ingegneria del software

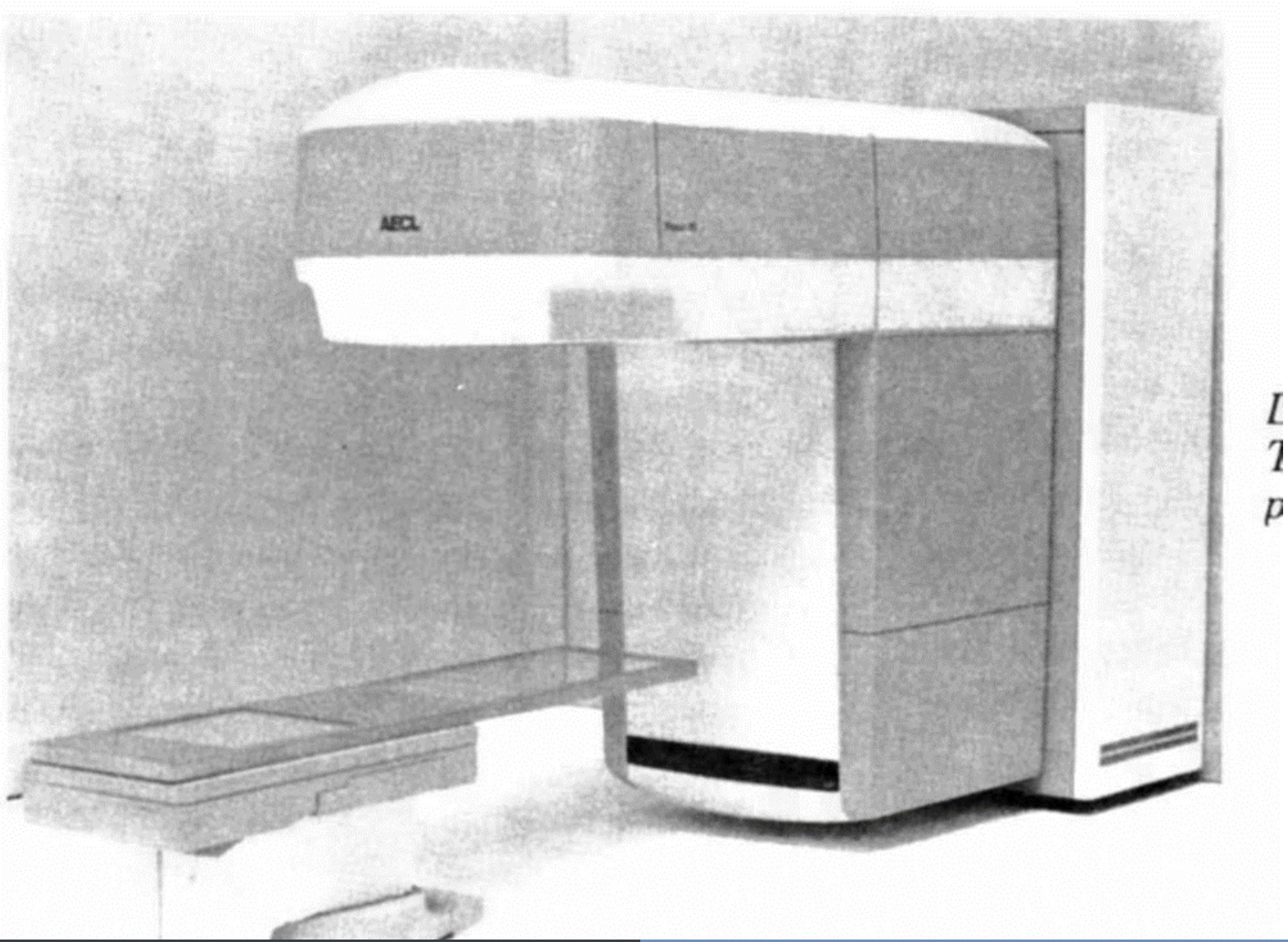
Ingegneria del software

Vincenzo Bonnici
Corso di Laurea in Informatica
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Università degli Studi di Parma

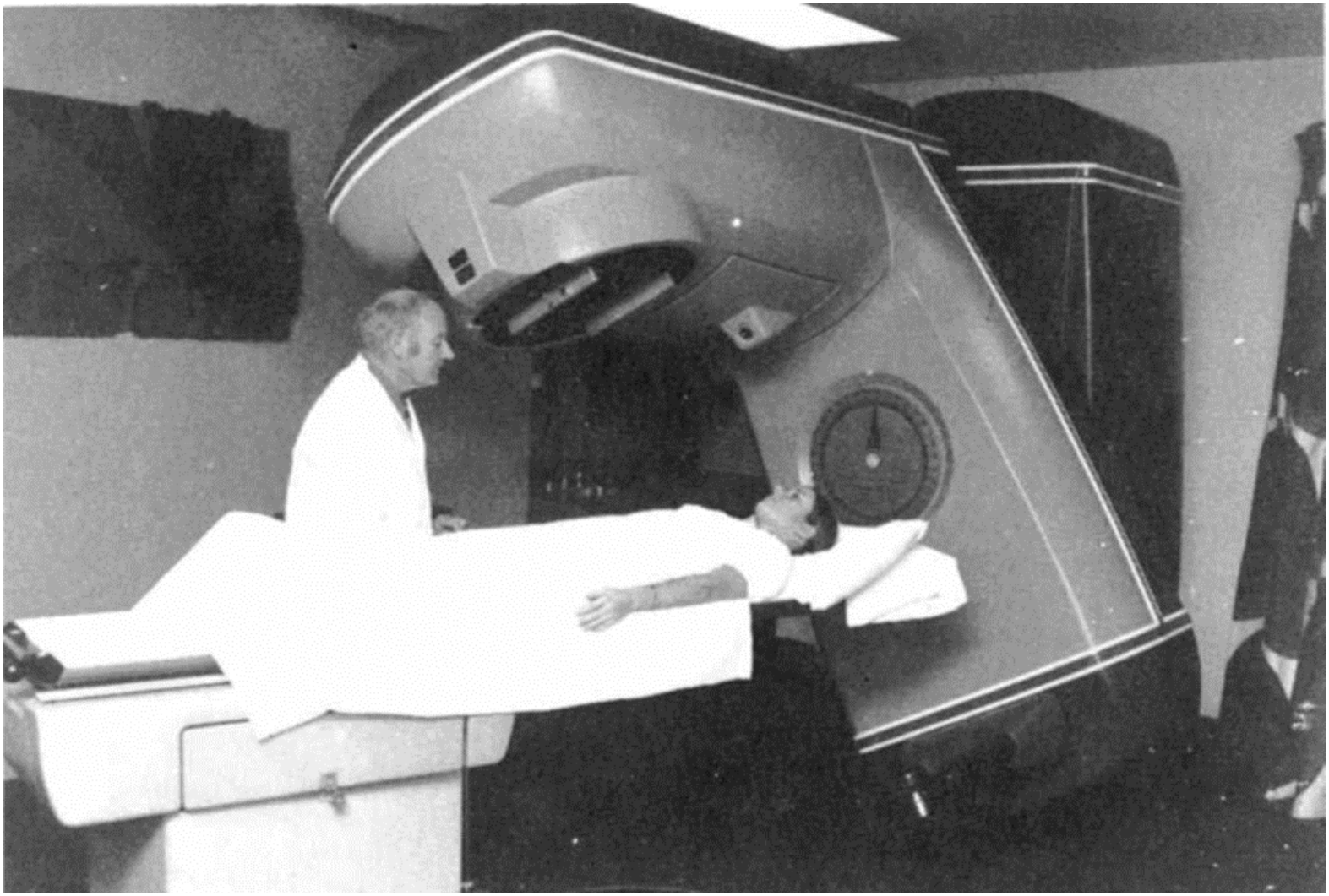
2025-2026

Preludio: il caso Therac-25

Il Therac-25 era una macchina per radioterapia che provocò almeno sei gravi incidenti di sovradosaggio, di cui tre mortali, fra il 1985 e il 1987.



Preludio: il caso Therac-25



Il predecessore Therac-6, con blocchi di sicurezza (interlock) elettromeccanici ed operazione manuale e software

Preludio: il caso Therac-25

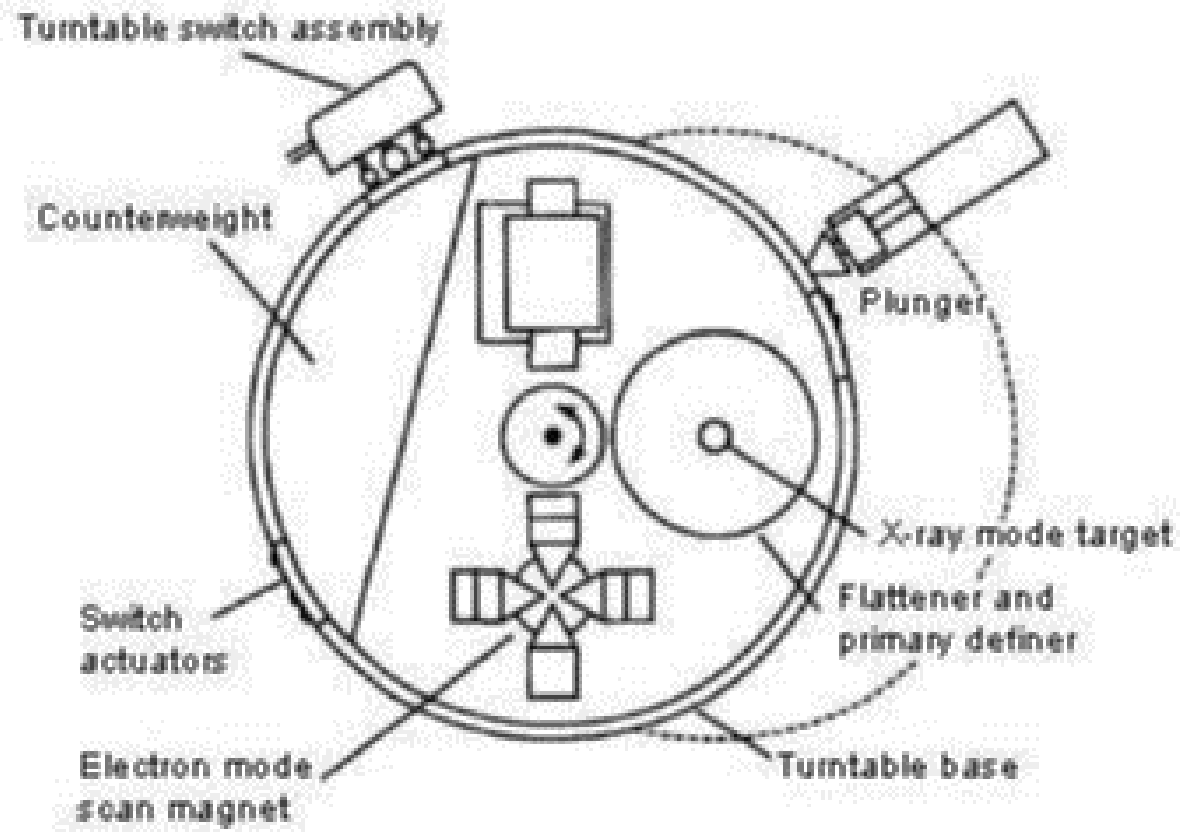


Figure B. Upper turntable assembly

Il vassoio portaaccessori girevole

Preludio: il caso Therac-25

Il vassoio girevole aveva tre posizioni, corrispondenti a tre modi di operazione:

- **specchio**: per il puntamento con un raggio di luce, a fascio spento.
- **elettromagneti e camera di ionizzazione**: per diffondere e misurare il fascio di **elettroni, ad energia bassa o alta ed a bassa intensità di corrente**.
- **bersaglio, diffusore e camera di ionizzazione**: per generare, diffondere e misurare il **fascio di raggi X, ad alta energia ed alta intensità di corrente**.

Il sovradosaggio avveniva quando il fascio di elettroni per raggi X (25 MeV ed alta intensità di corrente) era attivo senza che gli appositi accessori fossero posizionati.

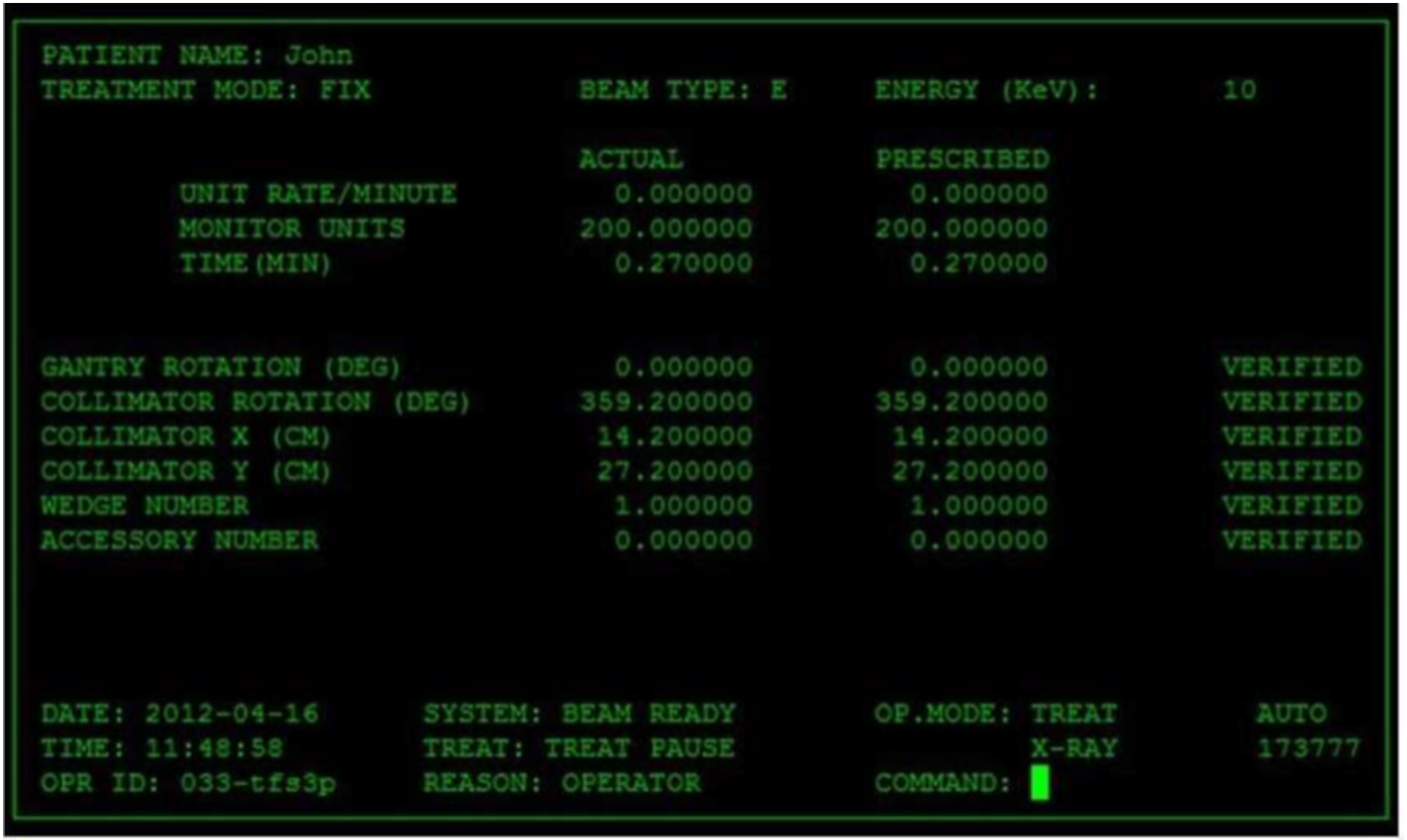
Preludio: il caso Therac-25

Genealogia

- Therac-6 (raggi X): blocchi di sicurezza elettromeccanici, comando manuale (interruttori, potenziometri. . .) e software (terminale VT100).
- Therac-20 (raggi X ed elettroni): blocchi di sicurezza elettromeccanici, comando manuale e software.
- Therac-25 (raggi X ed elettroni): blocchi di sicurezza software, comando prevalentemente software, stato dei sensori non controllato.

Terminale Digital Equipment Corporation VT100.

Preludio: il caso Therac-25



Interfaccia utente del Therac-25.

Preludio: il caso Therac-25

L'operatore può inserire dei valori di default con un ritorno carrello.

Portando il cursore sulla linea `COMMAND`, l'operatore segnala il completamento dell'inserimento o modifica dei dati.

Il monitor mostra i parametri di funzionamento in due colonne:

- ACTUAL: valori rilevati dai sensori;
- PRESCRIBED: valori impostati dall'operatore.

Se i valori non corrispondono, sono possibili due condizioni di arresto:

- SUSPEND: richiede un reset completo (**reimpostando i parametri**) per riprendere il funzionamento;
- PAUSE: basta il comando **P** per riprendere, dopo aver corretto qualche parametro e **lasciando gli altri invariati**.

Preludio: il caso Therac-25

Gestione degli errori:

- molti messaggi consistevano solo in un numero, p.es., `MALFUNCTION 54`;
- nessuna spiegazione nella documentazione;
- messaggi di significato ambiguo (e comunque non documentato):
- `MALFUNCTION 54` $\mapsto$ delivered dose **either too high or too low**;
- nessuna indicazione di livello di gravità;
- condizioni irrilevanti segnalate **frequentemente** come malfunzionamenti (*al lupo, al lupo!*).

Preludio: il caso Therac-25

L'incidente dell'East Texas Cancer Center, Tyler, Texas, marzo 1986

- L'operatrice imposta velocemente i dati e porta il cursore su `COMMAND`;
- si accorge di aver scritto 'X' (raggi X) invece di 'E' (elettroni);
- corregge l'errore e usa il ritorno carrello per confermare gli altri dati;
- il display mostra `VERIFIED` e `BEAM READY`;
- l'operatrice accende il fascio (tasto **B**);
- la macchina va in `PAUSE` col messaggio `MALFUNCTION 54` e mostra una dose somministrata di 6 unità invece delle 202 richieste;
- l'operatrice preme **P**;
- `PAUSE`, `MALFUNCTION 54`, 6 unità;
- il paziente si alza dal tavolo e cerca di uscire dalla stanza.

Il paziente morí cinque mesi dopo.

Preludio: il caso Therac-25

L'incidente dell'East Texas Cancer Center, Tyler, Texas, aprile 1986

- simile al precedente, ma l'operatrice non cercò di ripetere il trattamento dopo il primo arresto della macchina.

Il paziente comunque morí tre settimane dopo.

Il problema SW (a grandi linee):

- ci sono diversi processi concorrenti, fra cui uno per gestire l'interfaccia e impostare i parametri ed uno per attivare i magneti di deflessione del fascio;
- i processi comunicano per mezzo di **variabili condivise**, ma **senza meccanismi di sincronizzazione** (semafori etc.).
- quando la sequenza *impostazione-correzione-accensione* viene completata prima che finisca l'attivazione dei magneti, il sistema mantiene i valori non corretti,
- risultando in un fascio alla massima energia ed intensità con la macchina predisposta per il trattamento a elettroni.

Preludio: il caso Therac-25

Il problema HW (a grandi linee):

- La camera di ionizzazione, esposta ad un fascio per raggi X senza l'attenuazione del bersaglio e del diffusore, si satura e dà valori inattendibili (6 dosi);
- il sistema non ha meccanismi per controllare che i sensori funzionino correttamente.

Preludio: il caso Therac-25

L'incidente dello Yakima Valley Memorial Hospital, gennaio 1987

- L'operatore esegue due trattamenti a raggi X per un totale di 7 rad;
- imposta i dati per un trattamento a raggi X da 79 rad;
- usa lo specchio di puntamento e dà il comando **set** per posizionare correttamente il vassoio;
- il monitor mostra `BEAM READY` e l'operatore preme **B**;
- la macchina va in pausa e non mostra alcuna dose somministrata;
- l'operatore preme **P** e la macchina torna in pausa, mostrando una dose di 7 rad;
- il paziente si lamenta.

Il paziente morí in aprile.

Preludio: il caso Therac-25

Il problema SW (a grandi linee):

- due processi concorrenti comunicano attraverso variabili condivise;
- la variabile `Class3` uguale a zero significa che i parametri sono corretti per il tipo di trattamento;
- la variabile `F\$mal` uguale a zero significa che la macchina è pronta;
- il processo `SetUpTest` incrementa `Class3` finché i parametri non sono corretti;
- poi legge `F\$mal` e passa alla fase successiva se `F\$mal` è uguale a zero.
- il processo `Lmtchk` legge `Class3` e se questa è uguale a zero **non** controlla la posizione del vassoio; **ma**
- `Class3` va a zero per **overflow** ogni 127 volte che viene incrementata; **quindi**
- può accadere che il fascio venga abilitato anche se il vassoio non è posizionato correttamente.

Errori di programmazione

Sono stati individuati con certezza due particolari errori:

- **overflow**:
  - si usa un intero per rappresentare valori booleani, con la convenzione "diverso da zero" $\mapsto$ FALSO;
  - per assegnare "FALSO" si incrementa la variabile invece di assegnare un valore costante.
- **sincronizzazione**:
  - non si usano meccanismi di protezione per l'accesso alle variabili condivise.

Problemi di progetto e di sistema

- sistema operativo real-time (RTOS) sviluppato ad hoc
  - quando era disponibile il DEC RT-11;
- sistema operativo scritto interamente in assembler
  - quando era disponibile il linguaggio C;
  - probabilmente monolitico;
- mancanza di *programmazione difensiva*
  - eccezioni, asserzioni, "trucchi" vari:
- interfaccia utente piú amichevole che sicuro;
- gestione di errori e malfunzionamenti inadeguata
  - vedi lucidi precedenti.

Errori di processo

- Documentazione di specifica e di progetto:
- controllo di qualità;
- valutazione dei rischi
  - malfunzionamenti SW non considerati;
- collaudo del SW
  - test di unità;
  - test di integrazione;
  - SW troppo complesso;
- risposta inadeguata alle segnalazioni fatte dagli utenti.

Errori di metodologia

- considerare solo errori HW
- valutazione pseudoquantitativa dei rischi
  - espressi come (bassi) valori di probabilità senza giustificazione sperimentale o matematica:
- riuso acritico del SW
  - in parte ripreso da Therac-6 e Therac-20
  - che avevano blocchi di sicurezza HW;
- confusione fra affidabilità e sicurezza;
  - *affidabilità*: capacità di funzionare a lungo senza guasti.
  - *sicurezza*: incapacità di fare danni.
- confusione fra uso e collaudo;
  - *uso*: operazione del sistema nel suo ambiente di applicazione, allo scopo di fornire i servizi richiesti.
  - *collaudo*: esercizio del sistema in ambiente controllato, allo scopo di individuare malfunzionamenti.

Preludio: l'incidente Ariane 5 Flight 501

Il razzo Ariane 5 si distrusse 37 secondi dopo il lancio nel giugno del 1997.

Malfunzionamento

- overflow in una conversione da 64 a 16 bit;
- arresto del sistema di riferimento inerziale (SRI);
- il "messaggio di errore" dell'SRI interpretato dal processore principale come un dato valido, **quindi**
- il processore principale comanda un brusco cambiamento di rotta.

Il colpevole?

... codice Ada

```
declare
     horizontal_veloc_sensor: float;
     horizontal_veloc_bias: integer;
     ...
 begin
     declare -- "suppress" ELIMINA IL CONTROLLO A RUNTIME
         pragma suppress(numeric_error, horizontal_veloc_bias);
     begin
         sensor_get(horizontal_veloc_sensor);
         horizontal_veloc_bias := integer(horizontal_veloc_sensor);
         ...
     exception -- L'ECCEZIONE "numeric_error" NON VIENE GESTITA
         when numeric_error => calculate_vertical_veloc();
         when others => use_irs1();
 end;
```

Problemi di progetto e di sistema

- mancanza di defensive programming
  - nella versione precedente (Ariane 4) alcune operazioni non erano protette delle eccezioni
  - **per motivi di efficienza**
  - perché le protezioni erano considerate inutili;
- gestione di errori e malfunzionamenti inadeguata
  - arresto totale dell'SRI per eccezione HW;
- interfacciamento fra sottosistemi
  - flag di errore scambiata per dato normale;

Errori di processo

- specifiche incomplete
  - fra i requisiti dell'SRI manca la traiettoria della missione;
  - mancano i vincoli fisici del sistema;
- collaudi incompleti;

Errori di metodologia

- considerare solo errori HW
  - il sistema aveva doppia ridondanza (2 SRI, 2 processori principali), **ma**
  - una sola versione di SW;
- riuso acritico del SW
  - overflow causato dalle diverse caratteristiche della traiettoria fra Ariane 4 e Ariane 5.
  - la funzione che causò l'overflow era inutile nell'Ariane 5!

Introduzione: prime definizioni

L'ingegneria del software non mi piace. Mi piacciono le cose pratiche, mi piace programmare (in assembler).
(confessione tra studenti)

Ein Architekt ist ein Maurer, der Latein gelernt hat.
(un architetto è un muratore che ha studiato il latino.)
Adolf Loos

- L'ingegneria del SW è una cosa pratica?
- Che differenza c'è fra programmare e sviluppare SW?
- Un ingegnere del SW è un programmatore che ha studiato il latino?

L'ingegneria del software non piaceva nemmeno al Prof. Dijkstra (EWD1036.pdf)

L'ingegneria del software è il settore dell'informatica che si occupa della creazione di sistemi software talmente grandi o complessi da dover essere realizzati da piú squadre di ingegneri. Di solito questi sistemi esistono in varie versioni e rimangono in servizio per parecchi anni. Durante la loro vita subiscono numerose modifiche.

C. Ghezzi et al., Ingegneria del software: fondamenti e principi

- In realtà, l'ISW è molto utile anche in progetti piccoli, fatti da un solo sviluppatore.

Introduzione: le parti in causa

- **Sviluppatore**: chi partecipa direttamente allo sviluppo del SW, come analisti, progettisti, programmatori, o collaudatori. In alcuni casi il termine "sviluppatore" verrà contrapposto a "collaudatore".
- **Produttore**: per "produttore" del SW si intende un'organizzazione che produce SW, o una persona che la rappresenta. Uno sviluppatore generalmente è un dipendente del produttore.
- **Committente**: organizzazione o persona che chiede del SW al produttore.
- **Utente**: una persona che usa il SW. Generalmente il committente è distinto dagli utenti, ma spesso si userà il termine "utenti" per riferirsi sia agli utenti propriamente detti che al committente.

Spesso il SW non viene sviluppato per un committente particolare (applicazioni dedicate, o custom, bespoke), ma viene messo in vendita (o anche distribuito liberamente) come un prodotto di consumo (applicazioni generiche, o shrink-wrapped).

Introduzione: specifica e implementazione

- **Specifica**: una descrizione precisa dei **requisiti** (proprietà o comportamenti richiesti) di un sistema o di una sua parte.
  - descrive una certa entità "dall'esterno", cioè dice quali servizi devono essere forniti o quali proprietà devono essere esibite da tale entità.

- **Implementazione**: un sistema che **soddisfa** (implementa) la specifica.
  - generalmente, una specifica può avere piú implementazioni;
  - nei rami tradizionali dell'ingegneria (civile, meccanica. . .), l'implementazione è un sistema fisico;
  - nell'ingegneria informatica, l'implementazione è un codice eseguibile, **cioè**
  - **un modello a basso livello** di un processo di calcolo.

Introduzione: modelli e linguaggi

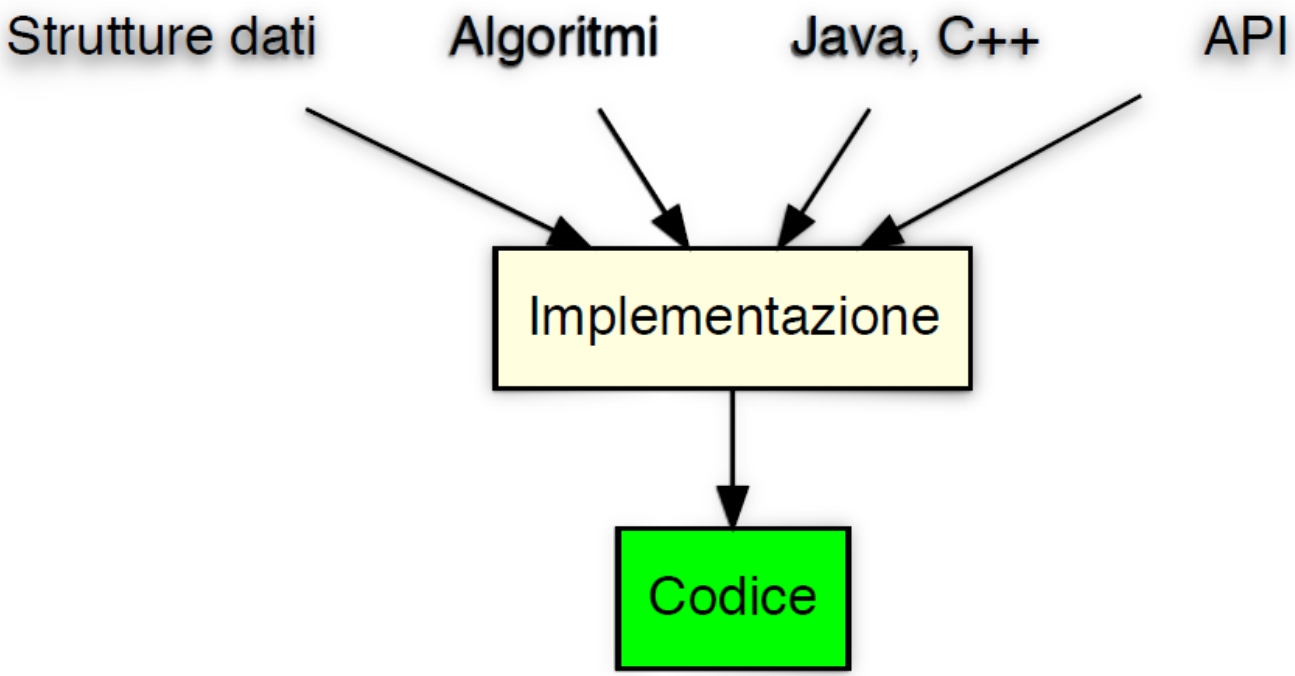
Il processo di sviluppo del SW si può vedere come la costruzione di una serie di **modelli**

- Un modello è una descrizione astratta di un sistema, che serve a studiarlo prendendone in considerazione soltanto quelle caratteristiche che sono necessarie al conseguimento di un certo scopo.
- Ogni sistema, quindi, dovrà essere rappresentato per mezzo di piú modelli, ciascuno dei quali ne mostra solo alcuni aspetti, spesso a diversi livelli di dettaglio.

Un **linguaggio** offre

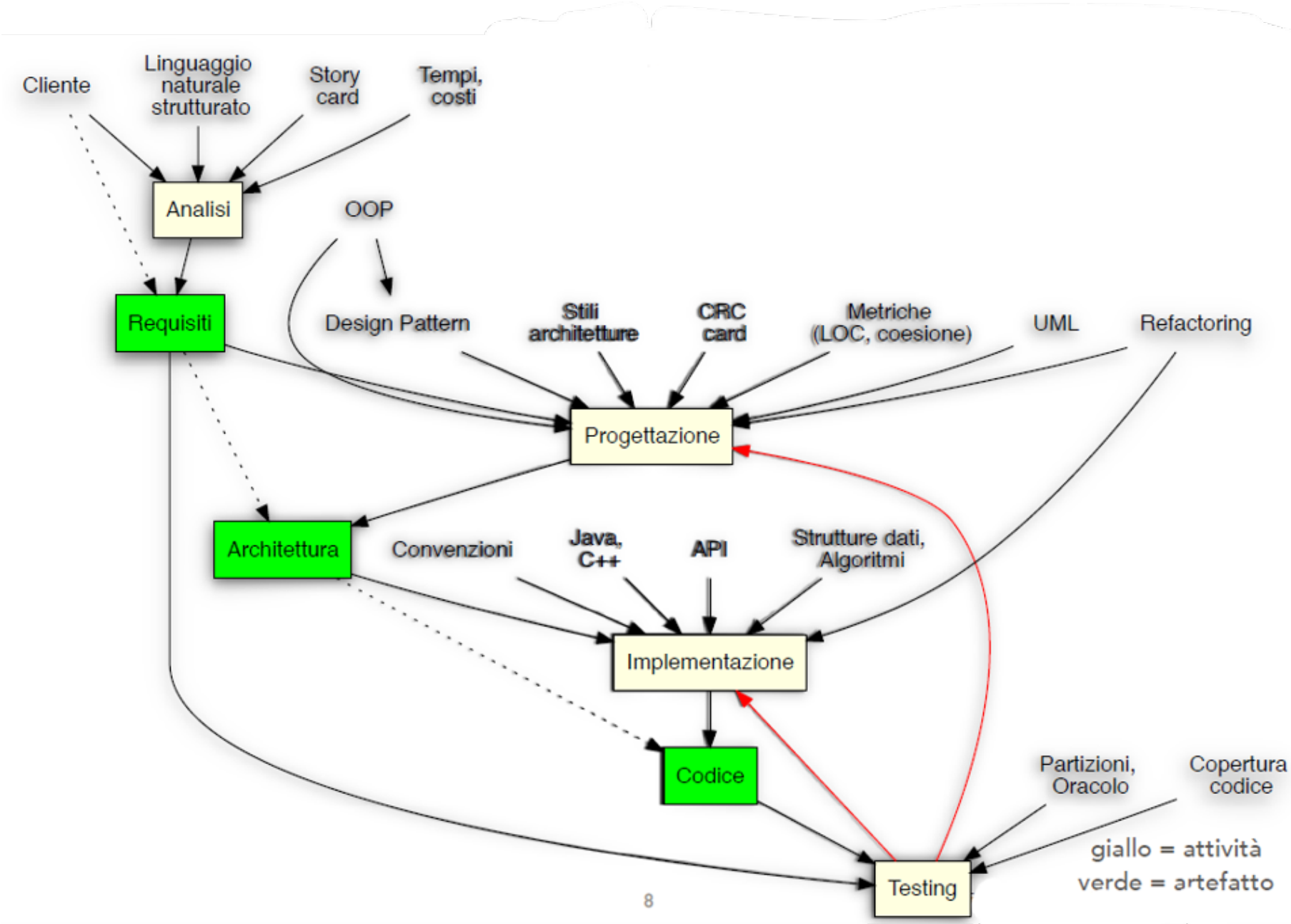
- un vocabolario di simboli (parole o segni grafici) che rappresentano i concetti necessari a descrivere certi aspetti di un sistema,
- una sintassi che stabilisce in quali modi si possono costruire delle espressioni (anche espressioni grafiche, cioè diagrammi), e
- una semantica che definisce il significato delle espressioni.

Senza l'ingegneria del software



giallo = attività
verde = artefatto

Attraverso l'ingegneria del software



Contesto

Sir Tony Hoare: There are two ways of constructing a software design

- One way is to make it so simple that there are obviously no deficiencies
- The other way is to make it so complicated that there are no obvious deficiencies

Progetti piccoli	Progetti grandi
Uno sviluppatore	Team di sviluppatori
Lo sviluppatore decide cosa fare	I clienti decidono cosa fare
Un prodotto	Famiglie di prodotti
Pochi cambiamenti	Tanti cambiamenti contemporanei
Vita breve	Lunga vita
Poco costoso	Molto costoso
Poche conseguenze	Grandi conseguenze

Programmazione — Ingegneria

Cosa è l'ingegneria del software?

- L'ingegneria del software si occupa della creazione di soluzioni economiche ed efficienti
  - per problemi pratici
  - applicando conoscenze scientifiche
  - per costruire prodotti software
  - con benefici per i clienti ed anche per la società
- La differenza tra ingegneria del software e computer science/informatics (informatica)
  - L'informatica si orienta sulla teoria ed i fondamenti
  - L'ingegneria del software si orienta ai problemi pratici di sviluppo e consegna di prodotti software utili

Cosa è l'ingegneria del software e di sistema

- L'ingegneria di sistema ha come oggetto tutti gli aspetti dello sviluppo di un sistema basato su computers, inclusi gli aspetti hardware, software e di processo.
- L'ingegneria del software può essere vista come una parte dell'ingegneria di sistema.
- Gli ingegneri del software collaborano
  - alla specifica del sistema,
  - alla progettazione architetturale
  - all'integrazione con le altre componenti.

Attività richieste nel processo di sviluppo software

- Specifica
- Progettazione
- Implementazione
- Validazione
- Installazione
- Manutenzione
- Smaltimento

Problemi nel processo di sviluppo del software

- Specifiche incomplete/incoerenti
- Mancanza di distinzione tra specifica, progettazione e implementazione
- Assenza di un sistema di validazione
- Il software non si consuma: la manutenzione non significa riparare alcune componenti "rotte", ma modificare il prodotto rispetto a nuove esigenze

Costi

- Processo software
  - Un insieme di attività ed i risultati ad esse associati che produce un sistema software
- Modello software
  - Una descrizione di un processo software sotto una particolare prospettiva
- Costi orientativi
  - Dipendono dal modello adottato
  - 15% specifiche, 20% design, 18% codice, 47% integrazione e test
  - 25% sviluppo, 75% evoluzione

Crisi del software e ingegneria del software

- Contesto degli anni ‘60
  - software
    - da programmi (sviluppati informalmente) – ad es., per risolvere sistemi di equazioni
    - a grandi sistemi commerciali – ad es., OS 360 per IBM 360
  - gli avanzamenti nelle tecniche di programmazione (ad es., programmazione strutturata) non aiutavano a costruire sistemi software (complessi) migliori
  - necessario un nuovo approccio, ingegneristico, con strumenti e tecniche opportuni

Crisi del software e ingegneria del software

- 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
  - ...
- Si ipotizzò che la crisi del software era temporanea
  - poteva essere risolta con strumenti e tecniche migliori – codificati nell'ambito di una nuova disciplina – l'ingegneria del software

Crisi del sw: temporanea o permanente?

- Dall'introduzione di Software Systems Architecture (2005)
  - i grandi sistemi software di oggi sono tra le strutture costruite dagli uomini più complesse
    - contengono milioni di linee di codice, migliaia di tabelle nelle basi di dati, e sono eseguiti da dozzine di calcolatori
    - ciò presenta ai team di sviluppo del software delle sfide formidabili
  - se queste sfide non vengono affrontate presto, i sistemi sono consegnati in ritardo, costano più del previsto, e con un livello di qualità inaccettabilmente povero

Crisi del sw: temporanea o permanente?

- Dunque, gli stessi problemi esistono tuttora – perché?
- mentre migliora l'abilità generale nella produzione del software, aumenta anche la complessità dei sistemi software
- nuove tecnologie e maggiori capacità di calcolo motivano nuove esigenze e nuovi obiettivi
  - le prestazioni dell'hardware crescono secondo la legge di Moore
  - in che modo crescono le aspettative sui sistemi che comprendono hardware e software?
  - in che modo cresce la capacità di produrre software?

Crisi del sw: No silver bullet

- Ecco la previsione di "No silver bullet" [Brooks, 1986]
  - of all the monsters who fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors – for these, we seek bullets of silver that can magically lay them to rest
  - there is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity

Che cos'è il software?

- Il software è/comprende
  - istruzioni (programmi) che quando eseguite svolgono una funzione desiderata con prestazioni desiderate
  - strutture di dati mediante le quali i programmi possono manipolare le informazioni in modo adeguato, e
  - documenti che descrivono le operazioni e l'uso dei programmi
  - ma anche la documentazione associata – come requisiti, modelli di progetto, ulteriori scelte di progetto, suite di test, ...

Tipi di prodotti software

- Alcune grandi categorie di software
  - software di sistema – ad es., un SO
  - software applicativo
  - software scientifico o per l'ingegneria
  - software embedded
  - linee di prodotti software
  - applicazioni web
  - software per l'intelligenza artificiale

Tipi di prodotti software

- Un'altra classificazione – due tipi fondamentali di prodotti software
  - prodotti generici – sviluppati per il mercato generale, per essere venduto a tutti i clienti interessati; ad es., MS Excel o Mozilla Firefox
  - prodotti personalizzati (a richiesta) – sviluppati per un cliente particolare, sulla base delle sue specifiche

- Nuovo software può essere creato sviluppando nuovi programmi, configurando dei sistemi software generici oppure riusando software esistente

Che cos'è l'ingegneria del software?

- L'ingegneria del software è l'applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, l'esercizio e la manutenzione del software

- Dunque
  - una disciplina ingegneristica
  - interessata allo sviluppo del software
    - interessata a tutti gli aspetti della produzione del software – dalla specifica alla manutenzione, dagli aspetti tecnici a quelli economici e di gestione del progetto e delle persone
    - interessata allo sviluppo efficiente di software di alta qualità
  - diversa da altre discipline ingegneristiche
    - a causa delle caratteristiche distintive del software

10 aree di conoscenza dell'ingegneria del sw

- **Requisiti del software**
  - un requisito è una proprietà che deve essere esibita nella risoluzione di un problema del mondo reale
- **Progettazione del software**
  - la progettazione (design) è il processo di definizione dell'architettura, dei componenti, delle interfacce e di altre caratteristiche di un sistema o componente
  - il progetto (design) è il risultato del processo di progettazione
- **Costruzione del software**
  - la costruzione del software si riferisce alla creazione dettaglia di software funzionante e significativo, attraverso una combinazione di codifica, verifica, test unitari, test di integrazione e debugging
- **Verifica (testing) del software**
  - il test del software consiste nella verifica dinamica del comportamento di un programma sulla base di un insieme finito di test case, scelto opportunamente da un dominio di esecuzione solitamente infinito
- **Manutenzione del software**
  - manutenzione del software in "operazione", per gestire anomalie, cambiamenti nell'ambiente operativo e nuovi requisiti utente

10 aree di conoscenza dell'ingegneria del sw

- **Gestione delle configurazioni del software**
  - disciplina che riguarda l'identificazione delle configurazioni del software in momenti temporali diversi, al fine di controllare in modo sistematico cambiamenti della configurazione e di mantenere l'integrità e la tracciabilità della configurazione durante tutto il ciclo di vita del sistema
- **Gestione dell'ingegneria del software**
  - riguarda la gestione e la misurazione dell'ingegneria del software
- **Processi dell'ingegneria del software**
  - riguarda definizione, implementazione, valutazione, misurazione, gestione, cambiamento e miglioramento del processo di ingegneria del software stesso
- **Strumenti e metodi dell'ingegneria del software**
  - riguarda metodi (informali, formali e basati su prototipazione) dell'ingegneria del software e strumenti di supporto alle diverse aree dell'ingegneria del software
- **Qualità del software**
  - riguarda considerazioni sulla qualità del software che trascendono i processi per la gestione del ciclo di vita del software

Quali sono i miti del software?

- Miti del software
  - opinioni diffuse – ma in realtà atteggiamenti fuorvianti
  - ovvero – perché l'ingegneria del software non è inutile?

- Miti del management
  - abbiamo già interi volumi di standard e procedure da seguire nello sviluppo del software – non c'è forse tutto l'indispensabile?
  - se siamo in ritardo, possiamo sempre recuperare aumentando il numero di programmatori
  - se decido di far realizzare un progetto software a una terza parte, posso restare tranquillo perché tutto il lavoro verrà svolto esternamente

Quali sono i miti del software?

- Miti della clientela
  - aun'affermazione generica degli scopi è sufficiente per iniziare a scrivere i programmi – i dettagli si possono trattare in seguito
  - i requisiti di un progetto mutuano di continuo, ma i mutamenti si gestiscono agevolmente grazie alla flessibilità del software
- Miti del programmatore
  - una volta scritto e fatto funzionare il programma, il nostro lavoro è finito
  - fino a quando il programma non può essere eseguito, non c'è modo di valutarne la qualità
  - il solo prodotto di un progetto concluso è il programma funzionante
  - l'ingegneria del software ci farà scrivere un'inutile e voluminosa documentazione che inevitabilmente rallenterà le cose

Risposta ai miti del software

- L'ingegneria del software
  - non si occupa di creare di documenti
  - piuttosto, si occupa di creare qualità
    - una migliore qualità porta a minor lavoro
    - un minor lavoro porta a tempi di consegna più rapidi

Abilità degli ingegneri del software

- Conoscenza di
  - Algoritmi e strutture dati
  - Linguaggi di programmazione
- Capacità di modellare
  - Operare a vari livelli di astrazione
  - Capire i requisiti, scrivere specifiche
  - Costruire modelli e ragionare con essi
- Capacità sociali
  - Lavorare in team grandi
  - Comunicare con le persone del team e con i clienti
  - Gestire il tempo e le risorse

Caratteristiche del software

- Caratteristiche che rendono i sistemi software diversi dagli altri prodotti
  - Complessità, Conformità, Modificabilità, Invisibilità
- Il software è un prodotto complesso
  - Componenti tutte differenti
  - Numero di stati cresce in modo combinatorio
  - Grandi dimensioni
  - Astratto ed immateriale
  - Non esistono leggi naturali che lo regolano
- Difficile da comprendere

Conformità

- Il software si deve conformare (adattare) all'ambiente esterno
  - Molte interfacce hardware
  - Vari utenti con profili differenti
  - Processi lavorativi predefiniti
- La conformità aggiunge complessità al software

Modificabilità

- Se un sistema software è di successo esiste sempre la necessità di cambiarlo
  - Per adattarlo ad una realtà che cambia (mutate esigenze)
  - Le richieste di estensione aumentano all'aumentare del successo
  - Poiché di successo, il sistema software sopravvive all'hardware per cui era stato sviluppato, generando una nuova esigenza di adattamento

Invisibilità

- Il software è invisibile e immateriale
  - Non può essere catturato completamente da un'unica rappresentazione geometrica
    - Flusso di controllo
    - Flusso di dati
    - Dipendenze di componenti e variabili
    - Sequenze temporali

Qualità dei sistemi software

- Le tecniche dell'ingegneria cercano di produrre istemi software entro i costi e i tempi preventivati e con qualità accettabile

- Come si valuta la qualità del software?
  - Aderenza allo scopo
  - Conformità alle specifiche
  - Definizione: Totalità di caratteristiche di un prodotto che si basano sulla abilità a soddisfare i bisogni espliciti ed impliciti [ISO]

Qualità dei sistemi software

- Correttezza
  - Un sistema software è corretto se soddisfa le specifiche funzionali, assumendo che tali specifiche esistono
  - Se le specifiche sono formali, la correttezza può essere definita formalmente
    - Può essere provata come un teorema
    - Può essere rifiutata trovando contro-esempi
- Cosa succede se le specifiche sono errate?

Qualità dei sistemi software

- Efficienza
  - L'uso di risorse da parte del software dovrebbe evitare sprechi (memoria, processore, comunicazione)
- Manutenibilità
  - La facilità di cambiare il software per soddisfare esigenze che cambiano
- Dependability
  - Il software dovrebbe essere affidabile (reliability), sicuro (security, safety)
- Usabilità
  - Il software dovrebbe essere usato facilmente dagli utenti per cui è stato progettato

Obiettivi (sfide) dell'ingegneria del software

- Affrontare sistemi legacy, eterogenei e ridurre i tempi di consegna
- Sistemi legacy
  - Sistemi software antichi ma di valore (indispensabili) che devono essere mantenuti ed aggiornati
- Eterogeneità
  - I sistemi sono distribuiti e consistono di un mix di hardware e software
- Consegna
  - C'è una pressione crescente per avere software più velocemente

Responsabilità

- Un ingegnere del software deve comportarsi in modo onesto ed eticamente responsabile
  - La confidenzialità di collaboratori e clienti dovrebbe essere rispettata
  - Il livello di competenza non dovrebbe essere falsato
  - Le leggi sulla proprietà intellettuale (copyright) dovrebbero essere conosciute e rispettate
  - Le capacità tecniche non dovrebbero essere impiegate in modo non appropriato (es. per diffondere virus)