CORSO I.F.T.S.

"TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE"

Matricola 2014LA0033

DISPENSE DIDATTICHE
MODULO DI "PROGETTAZIONE SOFTWARE"

Dott. Imad Zaza

Lezione del 08/07/2014

Elementi di Ingegneria del software

Principi

Principi dell'ingegneria del software

- Principi fondamentali che si applicano sia al processo che al prodotto
  - Si applicano attraverso metodi e tecniche
  - Metodi e tecniche si possono comporre in metodologie
  - L'applicazione delle metodologie può avvalersi di (o essere imposta da) strumenti

Principi, metodi e tecniche, metodologie, strumenti

- Ogni strato si basa su quello immediatamente più interno
- La variabilità nel tempo cresce verso l'esterno

I principi

- Rigore e formalità
- Separazione degli interessi
- Modularità
- Astrazione
- Anticipazione del cambiamento
- Generalità
- Incrementalità

Rigore e formalità

- L'ingegneria del software è un'attività creativa
- Deve però essere praticata in modo sistematico
- Il rigore aumenta la nostra fiducia nei prodotti
- La formalità (definizione matematica dei concetti usati) è il rigore al massimo grado

Rigore e formalità

- L'ingegnere deve scegliere il grado di rigore opportuno (il rigore costa)
- In generale, a una maggior necessità di fiducia (componenti critici) deve corrispondere un maggior rigore
- La formalità facilita le procedure automatiche di generazione e verifica

Rigore e formalità

- Esempi nel prodotto:
  - prova formale (matematica) di correttezza di un programma
  - generazione sistematica dei dati di test
- Esempio nel processo:
  - una documentazione rigorosa (anche se informale) del processo facilita la gestione dei progetti e il controllo dei tempi di produzione

Rigore

- Il rigore influenza positivamente anche:
  - manutenibilità
  - riusabilità
  - portabilità
  - comprensibilità
  - interoperabilità

Separazione degli interessi

- Affrontare la complessità di un problema considerandone separatamente gli aspetti.
  - Quasi sempre gli aspetti di un problema come la produzione di un software sono correlati e a volte in contrasto.
  - Tuttavia, nella pratica, la separazione degli interessi è necessaria (anche per suddividere il lavoro fra persone)

Separazione degli interessi

- Separazione delle attività in diversi periodi temporali
- Separazione della trattazione delle diverse qualità (es.: correttezza, prestazioni, usabilità)
- Separazione del sistema in diverse parti (modularità)
- Separazione degli aspetti relativi al dominio del problema da quelli relativi all'implementazione

Separazione degli interessi

- A volte si perde la possibilità di fare ottimizzazioni globali
- Perciò occorre decidere in anticipo cosa trattare separatamente e cosa no

Modularità

- Istanza del principio della separazione degli interessi, applicato alle parti del sistema.
- Un sistema può essere suddiviso in parti dette moduli.
- Si opera (specifica, progettazione, implementazione, verifica):
  - separatamente sui singoli moduli
  - sulla loro interazione

Modularità

- Top-down (divide et impera)
  - Prima si suddivide il sistema nei moduli che lo compongono
  - Poi si opera sui singoli moduli
- Bottom-up:
  - Prima si opera su singoli moduli
  - Poi li si compone per formare il sistema

Modularità

- Benefici: possibilità di
  - scomporre un sistema in parti più semplici
  - comporre un sistema a partire da parti piu semplici
  - comprendere un sistema come insieme di parti interagenti
  - modificare un sistema operando localmente sui moduli
  - favorisce la comprensibilità e la manutenibilità

Modularità

- Alta coesione dei moduli
  - gli elementi di un modulo sono raggruppati per un motivo logico
    - es.: funzioni raggruppate come metodi di una classe se condividono lo stato
- Debole accoppiamento fra moduli
  - due moduli hanno poche dipendenze reciproche
  - è più facile considerarli separatamente
  - sostituirne uno è meno costoso

Astrazione

- Consiste nell'ignorare i dettagli non significativi e considerare gli aspetti importanti di un problema
- Necessaria per affrontare la complessità dei problemi
- Che cosa sia significativo dipende dallo scopo dell'astrazione

Astrazione

- L’astrazione produce modelli:
  - La specifica (formale o informale) di un software è un modello della sua implementazione
  - Ragionando sulla specifica si possono ignorare i dettagli implementativi
    - Es.: la semantica dei linguaggi di programmazione astrae dall’hardware sottostante

Astrazione:

- I commenti nel codice sono astrazioni di ciò che fa una funzione, procedura o frammento di codice
  - L'astrazione viene applicata nei processi per stimare i costi di un nuovo progetto:
    - si utilizzano alcune caratteristiche salienti del nuovo progetto per confrontarlo con progetti già svolti

Anticipazione del cambiamento

- Nel tempo, il software può richiedere cambiamenti per adattarsi ai cambiamenti dell'ambiente e dei requisiti
  - La possibilità di cambiare un software (evolvibilità) richiede che esso sia progettato in vista del cambiamento
  - Cercare di localizzare i probabili cambiamenti in singoli moduli

Anticipazione del cambiamento

- La riusabilità è fortemente influenzata dall'anticipazione del cambiamento
  - Può essere vista come evolvibilità a livello dei singoli componenti
  - Necessario poter gestire le versioni del software (configuration management)
  - Applicabile anche ai processi:
  - prevedere possibili turnover
  - prevedere la manutenzione

Generalità

- Il problema considerato potrebbe essere un'istanza di un problema più generale
- Il problema generale potrebbe essere già stato risolto, o la sua soluzione potrebbe essere riusata in seguito
- E' però necessario pesare i costi e la perdita di prestazioni portati da una maggiore generalità

Generalità

- Prodotti di uso generale (off-the-shelf):
- pacchetti generali e componenti per problemi di uso comuni. Es.
  - elaborazione testi
  - fogli elettronici
- Passo successivo: server applicativi (application servers), forniscono le funzionalità generalizzate in maniera remota

Incrementalità

- In molti casi può essere utile portare avanti un processo come sequenza di passi successivi verso il prodotto finale.
- Permette di ricevere feedback sui prodotti intermedi
- L'incrementalità è strettamente correlata all'evolvibilità

Incrementalità

- Esempi:
- consegnare una versione del sistema con funzionalità ridotte, per ricevere un primo feedback
- implementare prima le funzionalità, poi prendere in considerazione le prestazioni
- produrre una serie di prototipi mentre aumenta la comprensione dei requisiti

Incrementalità

- La prototipazione rapida richiede un ciclo di vita differente
- Ciclo di vita flessibile e iterativo
  - Richiede accurata documentazione dei passi intermedi e delle loro relazioni per evitare di perdere il controllo

Compilatore: rigore e formalità

- Un compilatore è un prodotto critico: gli errori portano alla produzione di codice non corretto. Necessità di alta fiducia sulla sua correttezza
- Le specifiche dei compilatori sono generalmente fornite in modo formale (grammatiche dei linguaggi, modelli di esecuzione)

Compilatore: rigore e formalità

- Notazione per la sintassi: forma di Backus-Naur
- Teoria degli automi e dei linguaggi formali

Compilatore: separazione degli interessi

- Correttezza, efficienza (della compilazione e del codice oggetto) e usabilità (messaggi diagnostici) trattate separatamente
  - Es.: non occorre pensare ai messaggi di errore mentre si implementano gli algoritmi di allocazione dei registri
  - Aspetti correlati e contrastanti: diagnostica a run-time (es.: controllo degli indici di array) e prestazioni.

Compilatore: modularità

- La compilazione si può suddividere in alcune fasi:
  - Analisi lessicale
  - Analisi sintattica (parsing)
  - Generazione del codice
  - La suddivisione in fasi si può riflettere nella suddivisione del sistema in moduli

Compilatore: anticipazione del cambiamento

- Sono possibili
  - cambiamenti della sintassi o semantica del linguaggio di programmazione
  - introduzione di istruzioni macchina più potenti
  - introduzione di nuovi dispositivi di input/output che richiedono istruzioni specifiche
  - le istruzioni di I/O in C e Ada sono parte di una libreria, invece che del linguaggio come in Pascal, per una migliore flessibilità

Compilatore: generalità

- Uso di codice intermedio per generare codice per più macchine fisiche
  - generare codice per più linguaggi
  - Compilatore parametrico rispetto al linguaggio sorgente (compiler compiler).
  - lex e yacc in Unix
  - Definite Claue Grammars in Prolog

Compilatore: incrementalità

- Sviluppo di un compilatore per un sottoinsieme del linguaggio, poi aggiunta progressiva di tutte le caratteristiche
- Concentrarsi successivamente su
  - correttezza
  - diagnostica
  - ottimizzazione
- Fornire via via un insieme sempre più ampio di librerie

Specifica dei requisiti

Più in dettaglio

Come si fa ?

- Metodo classico
  - Standart IEEE-830-1998
  - Altri standart

Termini

- Committente: colui che paga per il prodotto. Generalmente è colui che decide i requisiti
- Utente: La persone che usa il sistema non sempre il committente
- Fornitore: Chi produce il prodotto
  - La specifica dei requisiti è parte del contratto

Requisiti Funzionali

- Ciò che il sistema deve fare
- Costituiscono la ragione stessa per il quale il sistema viene impiegato

Specifica dei requisiti

- Analisi: studio di un problema prima di intraprendere qualsiasi azione
  - Studio del dominio del problema, che porta alla specifica di un comportamento esternamente osservabile
  - Identificazione di oggetti, funzioni, stati del problema in esame, relazione di entità
- Tutto ciò si concretizza in
  - SRS -Software Requirment Specification
  - Casi d'uso UML

Cosa sono i requisiti

- Con la parola requisito si intende una caretteristica o una proprietà che un adata persona o cosa è tenuta a possedere.
  - Requesiti del prodotto software
  - Un requsito è:
    - qualcosa che il prodotto deve fare
    - una caretteristica che esso deve provvedere
    - Un requisito esiste sia perchè il prodtto stesso lo richiede, sia perchè il committente vuole che esso sia parte del prodotto

Cosa sono i requisiti

- Distinguiamo tra scopo del prodotto e i suoi requisiti
  - Scopo: Obbiettivo fininalità che il prodotto si prefigge
  - Requisiti: Insieme delle caratteristiche che il prodotto deve soddisfare al proprio scopo ed uso

SRS

- Lo scopo primario è dare una descrizine completa di che cosa fa il sw senza descrivere come lo fa
  - Si parla di specifica: il sistam è visto come una scatola nera che interagisce con l'ambiente
  - Serve a:
    - Costruire la base per la fae successiva
    - Costuire la base di collaudo finale del sistema
    - Costituire la base contrattuale

SRS

- Ne il committente ne il fornitore sono in grado di produrre un buon SRS
  - Il committente non ha sufficiente conoscenza del processo software per poter scrivere un srs utilizzabile
  - il fornitore non conosce il dominio del problema come il committente

  - Il processo di stesura del SRS è un processo di confronto tra necessità e possibilità

IEEE - Std 830-1998

- E' la revisione di due precedenti versioni: dell'84 e del 93
- E' quindi uno standart vecchio (non prevede i casi d'uso)
- Ma è ancora usato

Struttura SRS

- Introduzione
  - Scopo
  - Applicazione
  - Glossario dei termini:
    - Definzioni, acronimi, abbreviazioni
    - Riferimenti
    - Visione generale
- Descrizione Generale
  - Prospettiva del prodotto
  - Funzioni del prodotto
  - Caratteristiche utente
  - Vincoli
  - Assunzione e dipendenze
- Requisiti Specifici
- Appendici
- Indice

Requisiti Specifici

- Requisiti Specifici
  - Requisiti interfaccia esterna
    - Interfaccia utente
    - Interfaccia HW
    - Interfaccia SW
    - Interfaccia di comunicazione
    - Requisiti Funzionali
      - REQUISITI uno per uno
    - Requisiti di performance
      - Vincoli di progettazione
      - Attributi di sistema software
        - » Affidabilità
        - » Manuntentabilità
        - » ...
    - Altri requisiti

Requisiti Funzionali

- Schema consigliato
- <id> il <SW> deve <funzione>

- R1 il sistema deve gestire tutti i bancomat della filiale
- R2 il sistema deve stampare la sintesi del giacenza per bancomat

.

SRS (ieee std 830-1998)

- Un SRS è buono se:
  - Corretto
  - Non ambiguo
  - Completo
  - Consistente
  - Ordinati i requisiti per importanza
  - Verificabile
  - Modificabile
  - Tracciabile

Non ambiguità

- Ogni requisito deve avere una sola interpretazione
  - Per ogni caratteristica descritta usa sempre e solo lo stesso termine
  - Se un termine può assumere differenti significati a seconda del contesto occorre provvedere un glossario dei termini che lo precisi

Glossario dei termini

- Il glossario dei termini serve a dare interpretazione univoca ai termini usati;

Esempio Cattivo

- R1 : tutti i sistemi sono controllati da un blocco di controllo

- Interpretazioni:
  - Un unico blocco di controllo controlla tutti i sistemi
  - Ogni sistema ha il suo controllore
  - Ogni file ha il suo controllore ma ogni controllore potrebbe controllare più file
  - Occorre usare un linguaggio di specifica FORMALE

Tracciabilità

- Una SRS è tracciabile se
  - E' chiara l'origine di ogni requisito (tracciatura all'indeitro , ai dcumenti precedenti)
  - ogni requisito ha un nome o un numero (tracciatura in avanti, per i requisiti futuri)
  - Quando un requisito è una conseguenza di un altro i due devono essere tracciabili in avanti ed in indietro
    - Il requisito sulla risposta della base di dati deriva dal requisito sulla risposta totale del sistema
  - Quando un requisito deriva da ragioni legilstative , normative ... dovrebbe riportare i loro estremi
    - Il requisito sul passaggio all'anno successivo di corso deve fare riferimento all'ordinamento della Scuola
  - Esistono molti standart

Elementi di Ingegneria del software

Qualità

E' un pur sempre un prodotto

- Caratteristiche peculiari rispetto ad altre tipologie di prodotto
  - intangibile
  - malleabile
- estremamente facile da modificare “fisicamente”
- human-intensive
- il processo di fabbricazione è banale e rappresenta una frazione piccolissima del costo
- la produzione coincide sostanzialmente con progettazione e implementazione

Come si misura la qualità!?

- Definizioni
  - Prodotto: il sistema consegnato al cliente (sono significativi anche i prodotti intermedi, o artefatti)
  - Processo: la composizione delle attività che portano al prodotto
- Le qualità del processo influenzano le qualità del prodotto (es.: un processo con test accuratamente pianificati genera un prodotto più affidabile)

Qualità esterne ed interne del prodotto

- Qualità esterne: visibili agli utenti (es.: affidabilità, prestazioni, etc.)
- Qualità interne: accessibili agli ingegneri (es.: verificabilità, rispetto delle specifiche di progetto, etc.)
- Le qualità interne influenzano le esterne (es.: la verificabilità migliora l'affidabilità)
- Iso-9126

Correttezza, affidabilità, robustezza

- Qualità esterne del prodotto :
  - Strettamente correlate
  - Intuitivamente: un software possiede queste qualità se fa ciò che l'utente si aspetta, realizza le funzionalità attese

Correttezza

- Un programma è corretto se soddisfa i suoi requisiti funzionali (cioè se svolge i suoi compiti come previsto)

- Se i requisisti funzionali sono specificati formalmente, è possibile dimostrare (matematicamente) o confutare (matematicamente o empiricamente) la correttezza di un software

Limiti del concetto di correttezza

- I requisiti funzionali possono essere
  - ambigui: impossibile stabilire la correttezza del software
  - sbagliati: il software può essere corretto ma non soddisfare l'utente
- E' una proprietà assoluta difficile definire un grado di correttezza
- non tutte le "scorrettezze" sono ugualmente importanti

Metodi per verificare la correttezza

- Sperimentali:
  - testing
- Analitici:
  - ispezioni
  - prove di correttezza
  - esecuzione simbolica

Affidabilità(Reliability)

- Definita in termini statistici: probabilità che il software operi come atteso in un determinato intervallo di tempo
- Un software si può considerare affidabile se la sua affidabilità supera una determinata soglia(ad esempio: uptime di un server web > 99.9%)

Affidabilità

- I prodotti ingegneristici in genere hanno una garanzia
- Il software invece viene venduto con una clausola di non responsabilità

Affidabilità e correttezza

- Se i requisiti funzionali sono esatti (cioè rappresentano fedelmente le aspettative dell'utente) (quasi mai ...) allora:
  - la correttezza implica l'affidabilità
- non vale il viceversa:
  - Ma i requisiti funzionali possono essere sbagliati: un software corretto può non essere affidabile

Robustezza

- Capacità del software di operare in modo accettabile in condizioni non previste (input inatteso, malfunzionamento hardware)
- La robustezza è il rispetto di requisiti non specificati
- Affidabilità e robustezza anche per i processi

Prestazioni

- Qualità esterna del prodotto :
  - Quantità di compiti che un software è in grado di svolgere in un tempo determinato

    Esempio: servire N richieste al secondo
  - Influenzate dall'hardware disponibile
  - Influenzate dall' efficienza (uso delle risorse senza sprechi di tempo e spazio)

Influiscono sull'usabilità

Prestazioni

- Influenzate dalla complessità degli algoritmi usati
  - Per molti problemi sono stati dimostrati limiti inferiori della complessità degli algoritmi
- Influenzano la scalabilità, cioè la capacità di gestire quantità maggiori di input

Prestazioni: Valutazione

- Studiando la complessità degli algoritmi usati
  - i risultati teorici forniscono ordini di grandezza per tempo e spazio: O(n), O(n^2) , …
  - le prestazioni di un'implementazione dipendono anche dalle costanti: an+b, an2+bn+c, …
- studiando l'implementazione:
  - misura: sistemi hardware e software di monitoraggio
  - analisi: studio analitico di un modello del prodotto (spesso basato sulla teoria delle code)
- simulazione: misura di un modello del prodotto (più costosa e accurata)

Prestazioni: Migliorie

- migliorando l'efficienza dei singoli moduli
  - può essere necessario riprogettare l'architettura del sistema
  - delegando alcune operazioni critiche ad hardware special-purpose

Usabilità

- Qualità esterna del prodotto
  - Misura quanto un utente reputa il software facile da usare
- Qualità soggettiva
  - un utente principiante può gradire messaggi frequenti e prolissi ma un utente esperto può essere infastidito
  - un utente principiante può preferire un'interfaccia grafica ma un utente esperto può preferire una riga di comando

Usabilità

- Per sistemi interattivi dipende dalla coerenza e dalla prevedibilità dell'interfaccia utente
  - Standardizzazione delle interfacce
  - Per sistemi embedded dalla facilità di installazione e configurazione
  - Dipende anche da correttezza e prestazioni

Verificabilità

- Misura quanto è facile verificare le proprietà del software come correttezza e prestazioni
  - Generalmente proprietà interna
- Come migliorarla?
  - progettazione modulare
  - linguaggi di programmazione adatti
  - norme di codifica
  - software monitor (codice inserito nel software per monitorare prestazioni o correttezza)

Manutenibilità

- Qualità interna del prodotto
- Manutenzione: attività di modifica del software dopo il suo rilascio
- Manutenibilità: facilità di manutenzione
  - Il 60% dei costi del software è dovuto alla manutenzione

Manutenibilità

- Tre tipi di manutenzione:
  - correttiva: corregge gli errori presenti
  - adattativa: modifica il software per permetterne il funzionamento quando cambiano le condizioni operative
  - perfettiva: modifica il software per migliorarne alcune qualità
    - aggiunta di funzioni (nuovi requisiti)
    - miglioramento delle prestazioni
    - miglioramento dell'usabilità

Legacy software

- Software ereditato
  - Software che esiste in un'organizzazione da lungo tempo
  - Alto valore strategico, alti investimenti
  - Obsoleto per tecnologie software e hardware utilizzate
  - Difficile da modificare e mantenere
  - Reverse engineering o reengineering

Manutenibilità

- Si può considerare l'insieme di due diverse qualità:
  - riparabilità: facilità di eliminare difetti del software (manutenzione correttiva)
  - evolvibilità: facilità di modificare il software per adattarlo a un nuovo ambiente (manutenzione adattativa) o migliorarne le qualità (manutenzione perfettiva)

Manutenibilità

- Riparabilità: migliorabile con progettazione modulare
  - debole accoppiamento fra moduli
  - linguaggi di programmazione di alto livello
  - ambienti di programmazione che rendano più facile individuare gli errori

Manutenibilità

- Evolvibilità:
  - favorita da progettazione modulare
  - le modifiche dovrebbero iniziare dalla specifica e non all'implementazione
  - l'evolvibilità tende a decrescere con l'introduzione di nuove versioni

Evolvibilità

- Molto importante: alcuni software degli anni 60 sono tutt'ora in uso, ad es:
  - OS 360, usato dalle banche
  - SABRE, sistema di prenotazione dei voli aerei
    - inizialmente sviluppato da IBM per American Airlines
    - attualmente utilizzato da più di 400 compagnie aeree, 30.000 agenzie di viaggio e da alberghi, ferrovie e noleggi auto

Riusabilità

- la simulazione

  - Possibilità di riutilizzare un prodotto software (o parte di esso) nella creazione di un altro, eventualmente con modifiche marginali.

  - Librerie di componenti riusabili. Ad es:
    - librerie matematiche (inizialmente in Fortran, ora in C e C++)

  - librerie per interfacce grafiche

  - librerie per la simulazione

Riusabilità

- Facilitata da progettazione modulare.
  - Uno degli scopi della programmazione orientata agli oggetti
- Anche:
  - riuso degli artefatti nei processi (specifiche, manuali)
  - riuso dei processi

Portabilità

- Qualità interna del prodotto :
  - Capacità del software di funzionare su piattaforme hardware e software diverse
  - Hardware: differenti architetture o dispositivi (pc, palmari, cellulari)
  - Software: differenti sistemi operativi o
  - software di base come DBMS o interfaccia grafica

Portabilità

- Economicamente importante
  - Facilitata da progettazione modulare (in modo da confinare le dipendenze dalla piattaforma in pochi moduli)
  - Esempi: UNIX e Linux
  - Importante nei sistemi distribuiti

Comprensibilità

- Qualità interna del prodotto
  - Dipende dalla progettazione e dalla documentazione del sistema e dagli strumenti usati
  - Dipende anche dalla difficoltà del problema che il software risolve
  - Necessaria per la manutenibilità

Interoperabilità

- Capacità di un software di cooperare con altri
  - Facilitata dalla definizione e dall'uso di interfacce standard (ad es. plug and play nei sistemi operativi)
  - Permette la creazione di soluzioni modulari (ad es. plugin nei browser web)
- sistemi aperti (ad es. CORBA)

Produttività

- Proprietà del processo, inverso del tempo necessario a produrre un software
  - Varia da individuo a individuo
  - La produttività di una squadra dipende dalla produttività degli individui
  - Riuso per aumentarla (tradeoff)
  - Influenza i processi, e viceversa
  - Difficile da misurare (metriche)

Tempestività

- Qualità di un processo: capacità di rendere disponibile il software al momento giusto.
  - Commercialmente, può essere più importante di altre qualità
  - Richiede una precisa pianificazione temporale (milestone)
    - Possibilità: consegna incrementale
  - Richiede processo incrementale e progetto
  - scomponibile

Visibilità

- Qualità del processo: misura quanto lo stato del processo è documentato in ogni momento.
  - Occorre documentare
    - i passi
    - i prodotti intermedi
  - Favorisce la coordinazione (evita i conflitti)
  - Serve a rispondere a richieste del management o del committente sullo stato di avanzamento
  - Rende più facile la sostituzione di membri della squadra (turnover).

Qualità in tipologie specifiche di prodotti software

- Alcune qualità sono particolarmente significative per determinate tipologie di prodotti
- Qualità in:
  - Sistemi informativi
  - Sistemi real-time
  - Sistemi distribuiti
  - Sistemi embedded

Sistemi informativi

- Sistemi atti a gestire informazioni
  - Spesso l'informazione è la risorsa più importante di un'organizzazione
  - Qualità:
    - Integrità dei dati
    - Sicurezza
    - Disponibilità dei dati
    - Prestazioni delle transazioni
    - Usabilità
  - Personalizzazione (end-user computing)

Sistemi real-time

- Sistemi in cui la correttezza dipende anche dal tempo in cui avviene l'elaborazione

(automazione di fabbrica, sorveglianza, controllo aereo, gestione del mouse,…)

- Spesso utilizzati in applicazioni in cui si richiede
  - alta affidabilità (sistemi mission critical)
  - safety (impossibilità del verificarsi di situazioni pericolose)

Sistemi distribuiti

- Sistemi software che funzionano su macchine diverse connesse in rete
- Consentono di migliorare affidabilità e prestazioni per mezzo di replicazione e distribuzione
- Qualità peculiari:
  - portabilità
  - livello di distribuzione
  - capacità di tollerare il frazionamento
  - capacità di tollerare malfunzionamenti di una parte dei nodi

Sistemi embedded

- Sistemi in cui il software non ha interfacce verso l'utente finale
  - Usati in aerei, robot, elettrodomestici, automobili, cellulari, …
  - Scelta tra implementazione di funzionalità in hardware o software
  - Spesso parti di sistemi informativi, in tempo reale, distribuiti

  Es. sistema di monitoraggio pazienti in un ospedale

Misura delle qualità

- Individuate le qualità, è necessario verificare se un prodotto o processo le possiede (quality assurance)

- MA nell'ingegneria del software, oggi solo alcune qualità (es. prestazioni) si possono misurare con precisione.

- La misura di altre qualità è un'area di ricerca molto attiva (metriche software).