Presentazione del corso:

INGEGNERIA DEL SOFTWARE

Docente: Michele Marchesi

michele@diee.unica.it

Sito Web del corso: agile.diee.unica.it/corsi

Obiettivi formativi

- L'ingegneria del software ha l'obiettivo di studiare e sviluppare strumenti e metodologie per la pianificazione dello sviluppo, realizzazione e valutazione di applicazioni software complesse. Il corso ha due obiettivi formativi:
  - fornire una panoramica generale di tali strumenti e metodologie;
  - approfondire anche praticamente le tecniche di analisi e progettazione di software ad oggetti

Propedeuticità e Capacità Operative

- **Propedeuticità**:
  - Fondamenti di informatica, LPOO1
  - Conoscenza degli elementi di base della programmazione e dei principi fondamentali delle metodologie di programmazione object-oriented.

- **Alla fine del corso, gli studenti:**
  - Conosceranno i principi dell'ingegneria del software
  - Saranno in grado di utilizzare UML e di effettuare l'analisi ad oggetti di sistemi non complessi.

Strumenti e bibliografia

- Argo UML http://argouml.tigris.org/

- Materiale distribuito durante il corso.
- M. Fowler, K. Scott, UML Distilled – 3 Edizione, Addison Wesley.
- R. Pressman, Principi di Ingegneria del Software, 3a ed., McGrawHill (per approfondimenti).

L’Ingegneria del Software

- **La IEEE Computer Society definisce l’ingegneria del software come:**
- **(1) L’applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, all’operatività e alla manutenzione del software; cioè l’applicazione dell'ingegneria al software.**
- **(2) Lo studio degli approcci definiti al punto precedente.**

L'Ingegneria del Software*

- **Si occupa di *teorie*, *metodi*, *strumenti* per *progettare*, *costruire* e *mantenere* sistemi software complessi …**
- ***…cosi' grandi da richiedere un TEAM (gruppo) di sviluppatori***

  --> problemi di comunicazione fra persone, anche non tecnici
- ***...da realizzare entro un dato TEMPO e BUDGET***
- ***...ed aventi un dato livello di QUALITA' ed AFFIDABILITA'***

- **David Parnas:**

  "Multi-person construction of multi-version software".

(*) Questi lucidi sono in parte derivati dai lucidi del prof. Bolognesi, univ. di Siena: http://www.iei.pi.cnr.it/~bolognesi/Siena/

Importanza della disciplina

- **Sempre piu' sistemi sono controllati via software**
- **Le economie dei paesi evoluti dipendono dal software (nell'industria, nelle amministrazioni)**
- **...e le spese di produzione del software rappresentano una frazione significativa del loro P.I.L.**

- **Dunque i progressi in questo settore possono avere un forte impatto economico**

Che cosa succederebbe se i calcolatori smettessero di funzionare?

- **Ipotesi non totalmente peregrina. La distruzione massiva del moderno hw potrebbe essere causata:**
  - **da una tempesta solare delle dimensioni dell'*Evento Carrington* del 1859**
  - **da una o più esplosioni nucleari ad alta quota, provocanti una perturbazione della ionosfera e quindi un EMP (impulso elettromagnetico)**
- **Solo l'hardware protetto a standard militare potrebbe sopravvivere**

Effetto di un EMP

- **Tutto ciò che è controllato da hw e sw smette di funzionare:**
  - **centrali elettriche, e in cascata tutti i dispositivi elettrici, anche i pochi sopravvissuti all'EMP**
  - **sistemi di comunicazione (telefoni, cellulari, antenne, Internet)**
  - **veicoli a motore, treni, navi, aerei (anche in volo), motori elettrici e a scoppio comandati da elettronica**
  - **Luci, elettrodomestici, pompe dell'acqua potabile e delle fognature, erogazione del gas, sistemi di riscaldamento e condizionamento**

Dopo un giorno

- La nazione è nel caos. La gente non riesce a comunicare, quasi tutti i veicoli sono fermi, eccetto biciclette e qualche auto storica
- I negozi, anche se chiusi, sono saccheggiati
- Le forze dell'ordine stentano a riorganizzarsi, in quanto la maggior parte dei loro sistemi di comunicazione non funzionano
- La gente è chiusa in casa, al buio e al freddo, senza possibilità di riscaldare il cibo...
- Molti impianti industriali sono in fiamme...

Dopo una settimana

- **La gente è affamata, i rifornimenti sono sempre fermi**
- **Dalle città c'è un esodo verso la campagna, a piedi**
- **Ci sono scontri tra abitanti della campagna e cittadini**
- **Le forze dell'ordine sono ancora poco organizzate; i cittadini iniziano a organizzarsi per bande**
- **Le centrali nucleari, senza controllo nè raffreddamento, iniziano ad andare in fusione del nocciolo e iniziano gli incendi delle barre delle scorie radioattive, ma nessuno ne viene a conoscenza**

… risparmiamoci gli scenari a un mese e a un anno!

Torniamo al sw -- Prospettiva storica

- **Il termine SE viene introdotto in una conferenza NATO in Germania (Garmisch) del 1964 sulla *crisi del software* derivata dalla accresciuta potenza dei computer di terza generazione.**
- **I costi dell'hardware crollavano, quelli del software crescevano: si sapevano scrivere buoni programmi (strutturati, efficienti), non buoni sistemi.**
- **Molti insuccessi in 30 anni, anche recenti**
  - **Esplosione del missile Ariane 5 -> danno fisico e di immagine per miliardi di $. Dovuto a un difetto sw nel sistema di controllo (errata conversione tra float e int)**
  - **DENVER Airport: difetti software -> ritardo di 9 mesi nell'apertura dell'aeroporto. Perdite: mezzo milione di dollari al giorno.**
- **Disciplina ancora in evoluzione**

Le 10 competenze dell'Ing. del Sw.

1. **Software requirements – *requisiti***
2. **Software design – *progettazione***
3. **Software construction – *scrittura***
4. **Software testing**
5. **Software maintenance – *manutenzione***
6. **Software configuration management**
7. **Software engineering management**
8. **Software engineering process**
9. **Software engineering tools and methods**
10. **Software quality**

Settori dell'ingegneria del sw

- **Processo di produzione:**
  - **Processo: attività, ruoli, manufatti, eventi**
  - **Vari tipologie: pianificato, RAD, agile, Lean...**
- **Architettura:**
  - **Moduli e componenti in cui è diviso un sistema sw, e loro relazioni**
  - **Molti tipi di architetture, a vari livelli**
- **Progettazione**
  - **Strumenti e notazioni per progettare l'architettura**
  - **Parte dai requisiti, e definisce quali sono I moduli, le loro relazioni e interfaccie, a vari livelli di astrazione**

Problematiche dell'ingegneria del sw

- **Molto di più che produrre sw funzionante:**
  - **Prodotti richiesti ed usabili**
  - **Con clienti disposti a spendere per averli**
  - **Sviluppati ad un costo economicamente conveniente**
- **Responsabilità per correttezza e sicurezza:**
  - **Obbligo etico di produrre sw funzionante, adatto e sicuro per gli utenti**
- **Il processo di produzione del sw è non lineare:**
  - **Un processo lineare è ripetibile (es., la produzione di un'auto o la costruzione di una casa)**
  - **Il processo sw richiede sviluppo per iterazioni con feedback *(inspect and adapt)*, testing**

Problematiche dell'ingegneria del sw

- **La S.E. definisce un approccio sistematico ed organizzato, secondo "best practice":**
  - **Pratica: passi per eseguire attività chiave del progetto**
  - ***Best Practice*: provata efficace in modo qualitativo e quantitativo in molti casi**
- **Attività fondamentali in cui ci sono BP:**
  - **Raccolta dei requisiti**
  - **Analisi**
  - **Progettazione (di alto e basso livello)**
  - **Codifica**
  - **Testing, validazione ed integrazione**
  - **Manutenzione ed evoluzione**

Processo (di produzione del) software

- **Un insieme di attività il cui obiettivo è lo sviluppo e l'evoluzione di un sistema software**
- **Attività fondamentali:**
  - ***Specifica*** **- cosa il sistema deve fare, e sotto quali vincoli – comprende raccolta dei requisiti e analisi**
  - ***Sviluppo*** **- produzione del software – progettazione, codifica e rilascio**
  - ***Validazione*** **- appurare che il sistema corrisponda alle aspettative del cliente - testing**
  - ***Evoluzione*** **- modifiche del sistema in risposta a evoluzioni dei requisiti iniziali – comprende tutto**

Esistono vari modelli (tipi) di processo software

- **Un modello è descritto in termini di *attività* principali, loro relazioni logiche e/o temporali, *manufatti* prodotti, *ruoli* delle persone**
- **Vi sono due modelli fondamentali**
- **Plan-driven:**
  - **Basati su pianificazione accurata anticipata**
  - **Do it right the first time**
- **Agili**
  - **Basati su iterazioni e adattamento**
  - **Do it right the last time**

I *costi* del software

- **Nello sviluppo di *sistemi integrati* i costi della parte *software* (spesso la più creativa) tendono a dominare su quelli *hardware*.**
  - **Il prezzo del software installato su un PC può superare di molto quello dell'hardware…**
- **La manutenzione del sw costa più dello sviluppo (anche il 70% del costo totale di un sistema).**
- **I costi del software sono legati solo alla sua ideazione e realizzazione. I costi di produzione su larga scala, a sviluppo completato, sono trascurabili.**
  - **quelli di un prodotto industriale (auto, edificio,...) no**
- **I costi dipendono fortemente dai *requisiti non -funzionali***
  - **es: prestazioni, affidabilità, portabilità…**

Le sfide dell'I.S.

- **Il sw è immateriale e quindi *duttile***
  - **Pro:**
    - **mezzo sommamente creativo**
    - **grande libertà**
  - **Contro:**
    - **non vi sono vincoli fisici**
    - **limitate basi scientifiche/matematiche**
- **Requisiti mutevoli**
  - **Il cliente non sa bene che cosa vuole**
  - **Cambiamento tecnologico continuo**
  - **Risposta alle azioni dei concorrenti**

Le sfide dell'I.S.

- **Gli ingegneri del sw sono ottimisti:**
  - **Tendono a sottostimare le difficoltà**
  - **Tendono a inserire sempre nuove funzionalità**
  - **Sono quindi sempre in ritardo**

- **Tempi di consegna sempre piu' brevi**
  - **Ambiente molto competitivo**
  - **Necessità di operare scelte aggressive**
  - **Cambiamenti continui**

Le sfide dell'I.S.

- **Utilizzo di *legacy systems***
  - **I vecchi sistemi devono essere manutenuti, aggiornati, integrati nei nuovi sistemi (esigenza economica)**
- **Eterogeneità**
  - **I sistemi tendono ad essere distribuiti e ad utilizzare diverse piattaforme hardware/software**

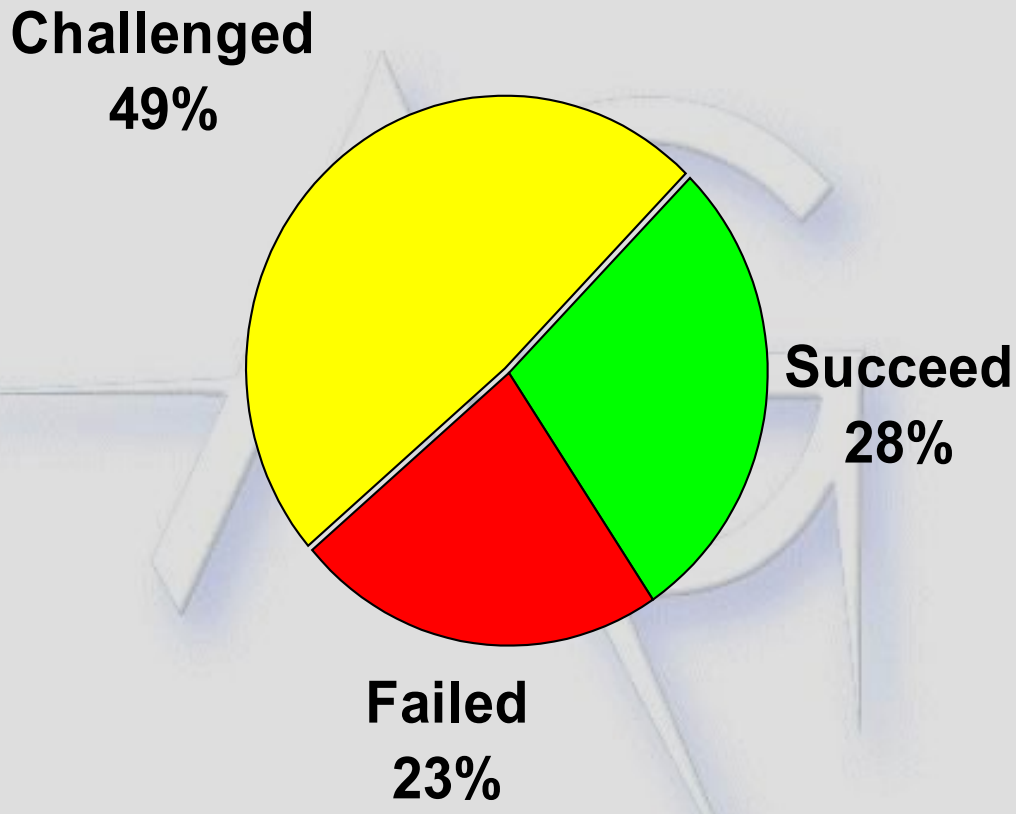
Le sfide dell'I.S.

- **I nuovi modelli di sistema informatico:**
  - **Web programming**
  - **Applicazioni per terminali mobili (smartphone e tablet)**
  - **Internet delle cose**
  - **Cloud Computing: il centro di elaborazione dati non è più locale, ma è nella rete**

La crisi del software

- **Ricerca CHAOS dello Standish Group**
  - **Analisi quantitativa del successo dei progetti sw**
  - **Iniziata nel 1994**
  - **Più di 30.000 progetti analizzati**
- **3 livelli di successo (o fallimento):**
  - **Successo: progetto finito nei tempi e costi previsti, con la maggior parte delle funzioni**
  - **Criticità: progetto che supera tempi e/o costi, e/o con funzionalità ridotte rispetto a quelle previste**
  - **Fallimento: progetto cancellato, o sistema prodotto non usato**

Risultati per l'anno 2000



Risultati per gli anni 1994-2009

Standish project benchmarks over the years

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Risultati per dimensioni del progetto (anno 2000)

Size	Successful	Challenged	Impaired
Under $500K	38%	44%	19%
$501K - $3M	27%	52%	21%
$3M - $6M	16%	55%	29%
$6M - $10M	4%	57%	39%
Over $10M	0%	66%	34%

Le proprietà del prodotto software

- **Il *prodotto sw* comprende tutto l'insieme dei manufatti del processo di sviluppo:**
  - **requisiti, specifiche, progetti, programmi,**
  - **file di configurazione**
  - **documentazione di sistema, documentazione utente**
  - **sito web per il *download* di nuove versioni, correzioni, informazioni…**
- **Prodotto 'logico', non 'fisico': molto *malleabile***
  - **Si può modificare molto facilmente, anche indipendentemente dal progetto o dalla specifica…(!)**
    - **… mentre per modificare un aereo si riparte sicuramente dal progetto**

Due tipi di prodotto software

- Prodotti *generici ('off-the-shelf')*
  - Sistemi 'stand-alone' prodotti da una azienda di sviluppo software (es: Microsoft) e offerti a un pubblico vasto, non necessariamente specializzato (es: Word, Excel,…).
  - In crescita dagli anni '80, con la diffusione dei PC
  - Requisiti e specifica sono prodotti dallo stesso sviluppatore
- Prodotti *specifici ('bespoke')*
  - Sistemi richiesti da un committente, o cliente, che è l'origine di requisiti e specifica
- Il maggior volume di affari è sui prodotti *generici*,
- Il maggior sforzo di sviluppo su quelli *specifici*.
- In Italia, si produce quasi solo software specifico.

La protezione del software

- Il software è un'opera dell'ingegno, e come tale è protetta dalle leggi sul diritto d'autore
- Il diritto d'autore garantisce che il software non possa essere copiato (tale e quale o sostanzialmente):
  - codice
  - icone, immagini, aspetto dell'interfaccia utente
- Esso dura 75 anni dalla morte dell'autore
- Di un programma si può però copiare il funzionamento, scrivendo dell'altro codice in modo indipendente
- In Europa il software non si può brevettare (protezione di 20 anni dalla registrazione)
- Negli USA non solo vige il diritto d'autore, ma anche il brevetto

Software proprietario

- All'inizio, il software era poco rilevante rispetto all'hardware
- Esso veniva sviluppato in specifici centri di calcolo
- Con l'avvento dei mini e personal computer, si cominciò a sviluppare software rivendibile per molti
- Ciò portò alla “chiusura” del software, proteggendo il codice sorgente per evitare la concorrenza
- Negli anni '80, alcuni programmatori, guidati da Richard Stallman reagirono a questa situazione, fondando la Free Software Foundation e introducendo il concetto di Open Source

Il software open source

- **Software distribuito con una *licenza* che permette le 4 libertà fondamentali:**
  - **Libertà 0: Libertà di eseguire il programma per qualsiasi scopo.**
  - **Libertà 1: Libertà di studiare il programma e modificarlo.**
  - **Libertà 2: Libertà di copiare il programma in modo da aiutare il prossimo.**
  - **Libertà 3: Libertà di migliorare il programma e di distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.**

Le licenze open source

- **GNU GPL (GNU General Public License): impone che ogni prodotto software derivato - ovvero, che modifica o usa codice sotto GPL - venga a sua volta distribuito con la stessa licenza.**
- **GNU LGPL (Lesser GPL): impone di ridistribuire sotto LGPL solo il sw. originariamente LGPL. Utile per librerie sw.**
- **BSD (Berkeley Software Distribution): garantisce le 4 libertà del software ma prevede che un programma protetto da licenze BSD possa essere modificato e ridistribuito usando la stessa o qualunque altra licenza**
- **MPL (Mozilla Pubblic License): il codice sorgente copiato o modificato sotto la licenza MPL deve rimanere MPL ma è possibile combinare codice MPL e codice proprietario, che resta tale**
- **Apache, MIT, CPL, EUPL (European Union Public License)**

I progetti Open Source

- **Sono tipicamente sviluppati tramite Internet, da *comunità***
- **Sono gestiti da fondazioni, o talora da società private (*dual licensing*)**
- **Comunità degli sviluppatori e degli utenti**
- **Tecniche specifiche di ingegneria del software per gestire sviluppo, testing, rilasci, manutenzione**
- **Vi sono progetti immensi e di grande successo:**
  - **GNU Linux**
  - **Apache**
  - **OpenOffice**
  - **Eclipse**
  - **...**

Il modello di business Open Source

- **Il software OS è tipicamente gratuito (FLOSS: Free Libre Open Source Software)**
- **Lo sviluppo non è retribuito**
- **I motivi economici per cui si collabora a un progetto FLOSS:**
  - **per farsi esperienza e curriculum**
  - **per sviluppare competenze e *rivendere servizi* sul software**
  - **perché sviluppare il sistema in modo proprietario sarebbe più costoso (es. apporto IBM, HP, Oracle, … a progetti OS)**
- **Alcuni FLOSS sono prodotti da società e hanno doppia licenza:**
  - **licenza GPL per un uso non supportato nè garantito**
  - **licenza proprietaria che include supporto, aggiornamenti, garanzie**

Proprietà interne ed esterne del software

- **Si distinguono proprietà:**
  - **Esterne: quelle visibili dall'utente finale.**
  - **Interne: quelle che riguardano chi lo sviluppa, lo mantiene, lo cambia**
  - **Le interne aiutano a ottenere le esterne: esempio, la *verificabilità* aiuta a ottenere la *affidabilità*.**
- **L'importanza di una data proprietà dipende dal tipo di prodotto e dal suo ambiente di utilizzo.**
  - **In sistemi safety-critical e real-time, affidabilità, sicurezza e efficienza sono cruciali**

Correttezza

- **Un programma, o la implementazione di un sistema, è funzionalmente corretto se si comporta sempre come definito nella *specifica dei requisiti funzionali*.**
- **Viene verificata con metodi sperimentali (*testing*) o analitici (*formal verification*).**

Affidabilità *(reliability)*

- **Proprietà statistica, espressa in termini della probabilità che il sistema operi come previsto in un determinato intervallo temporale.**
  - **Non implica correttezza: un programma può essere scorretto (non soddisfa le specifiche del cliente), ma affidabile (opera sempre)**
  - **Tasso di errore (Failure rate): numero di fallimenti per unità di tempo, o per n. di transazioni o di esecuzioni del programma**
- **Disponibilità del sistema o dei dati (percentuale di tempo in cui il sistema è disponibile – es. 24/7)**

Sicurezza (per sistemi *"safety critical"*)

- **Sicurezza assoluta di funzionamento nel caso di sistemi in cui il fallimento software può causare perdite umane**
- **Si può ottenere anche con i *metodi formali* di sviluppo del software**
- **Esistono precise regolamentazioni internazionali sullo sviluppo di software *life critical* nei vari settori (aeronautico, ferroviario, medicale, ecc.)**
- **Implica spesso revisioni e certificazioni ottenute da enti di controllo terzi**

Protezione *(Security)*

- **Protezione da accessi non autorizzati, e da conseguenti possibili danni economici**
- **Particolarmente critica per sistemi in rete e su Internet**
- **Comprende la protezione da minacce informatiche, quali *virus*, *worm*, *trojan*, ecc.**
- **Comprende la protezione da operazioni illegittime effettuate da utenti e sviluppatori con diritti di accesso**

Efficienza (*efficiency*, *performance*)

- **Un sistema è efficiente se usa le sue risorse in maniera economica. Tipicamente: usa poco spazio (disco, memoria centrale) e poco tempo per fornire i propri servizi.**
- **Questa proprietà migliora in genere col tempo, col miglioramento della tecnologia hardware (ci sono sempre più memoria e velocità).**
- ***Performance evaluation* mediante:**
  - **analisi di complessità degli algoritmi (caso medio, caso peggiore)**
  - **analisi matematica di un modello**
  - **simulazione sul modello.**
  - **misura sul campo (alla fine, è quello che conta!)**

Usabilità (*usability, user-friendliness*)

- **Documentazione e interfaccia utente (*user-interface*) *appropriate***
- **Proprietà soggettiva. Es.**
  - **Spiegazioni verbose e menù sono apprezzati dall'utente inesperto, non dall'esperto.**
- **Proprietà relativa: adeguamento alle soluzioni più diffuse**
  - **Mac --> Windows**
  - **In futuro…?**
- **In sistemi senza interfaccia utente (embedded): facilità di configurazione e adattamento all'ambiente hardware.**
- **Ricerca su nuovi paradigmi di interazione uomo-macchina**
- **Accessibilità: usabilità da parte di persone disabili**

Robustezza

- **Comportamento 'ragionevole' anche in *casi non previsti nei requisiti* (es. dati input scorretti o errore del disco).**
  - **Es: il sistema non perde i propri dati nemmeno se viene spento improvvisamente durante il funzionamento**
- **Se il problema è previsto nei requisiti, gestirlo correttamente rientra nella correttezza**
- **Ma i requisiti non possono specificare tutti i casi anomali!**

Interoperabilità

- **Capacità di cooperare con altri sistemi.**
- **Es. Un amplificatore Hi-Fi accetta giradischi e CD**
- **Es. 2 - Cooperazione fra le diverse componenti (Foglio elettronico, Word processor, Posta, Lucidi) di pacchetti software integrati.**

- **Viene ottenuta standardizzando le interfacce (*XML*).**
- **Sempre più importante con l'avvento delle reti e di Internet (tutto comunica con tutto!)**

Manutenibilità - evolvibilità

- **Deve essere possibile correggere e far evolvere il software per riflettere nuovi requisiti**
  - **Ma i cambiamenti devono interessare tutta la pila di documenti del processo!**
- **Es. - Il sistema di prenotazione aerea di molte compagnie (Alitalia inclusa) è mantenuto ed accresciuto dagli anni ‘60.**

Verificabilità

- **La specifica, il progetto, il codice vengono realizzati in modo da semplificarne la verifica di proprietà.**
  - **Certi linguaggi di specifica sono più verificabili di altri.**
  - **Il codice può venire arricchito con 'software monitors'.**
  - **E' aiutata dalla *tracciabilità* dei requisiti nei documenti di progetto e nel codice**

Riusabilità

- **Come *evolvibilità*, ma riferito a nuovi prodotti, anziché a versioni dello stesso prodotto.**
- **Applicabile soprattutto a parti del sistema.**
- **Librerie FORTRAN, C, Java disponibili sul mercato.**
- **Proprietà ben realizzata con l'approccio ad oggetti per librerie di "basso livello" (widget, interfaccia verso il DBMS…)**
- **Molto difficile da realizzare in modo economico e pratico per componenti di alto livello**

Portabilità

- **Quando l'applicazione può girare in diversi ambienti (piattaforma HW o SW)**
- **Si può ottenere separando nettamente le funzionalità indipendenti dalla piattaforma da quelle dipendenti**
- **Inoltre, il sistema deve utilizzare solo un sottoinsieme *stabile* di risorse dell'ambiente, o di istruzioni macchina.**

Scalabilità

- **Quando il sistema può “crescere” e “decrescere” a seconda delle necessità**

- **Crescita:**
  - **capacità di usare più memoria e più potenza di calcolo senza modifiche al codice**
  - **capacità di utilizzare più macchine/CPU che operano in parallelo, in numero che dipende dalle richieste**
    - **il sistema deve essere progettato per poterlo fare**

Riassunto delle proprietà del software

Esterne

Correttezza
Affidabilità
Sicurezza
Protezione
Efficienza
Usabilità
Robustezza
Interoperabilità
Portabilità
Scalabilità

Interne

Interoperabilità
Manutenibilità
Verificabilità
Riusabilità
Portabilità
Scalabilità

Conflitti tra proprietà del software

- **Non si possono ottenere tutte le proprietà al massimo grado!**

- **Esempio 1**
  - **Ottimizzare le interfacce utente** ***oppure*** **l'efficienza**

- **Esempio 2**
  - **Ottimizzare l'efficienza** ***oppure*** **la manutenibilità (mediante POO e riuso di componenti '*off-the shelf*')**

Requisiti di qualità in diverse aree applicative

- **Sistemi Informativi**
- **Sistemi *Real-time***
- **Sistemi Distribuiti**
- **Sistemi *Embedded***
- **Sistemi Web**

Sistemi Informativi

- **Sistemi orientati ai dati. La loro funzione è gestire informazioni e effettuare elaborazioni sui dati. Alcuni sono detti sistemi transazionali**
  - **Costruiti attorno a un *data base* che viene manipolato mediante *transazioni* (*create, retrieve, update, delete -- CRUD*).**
  - **Es.: banca, biblioteca, personale, contabilità, magazzino, comando e controllo, simulazione.**
- **Integrità dei dati (*robustezza*)**
  - **I dati non devono perdersi o diventare scorretti per malfunzionamenti del sistema**

Sistemi Informativi

- **Protezione**
  - **Protezione da accessi non autorizzati**
- **Disponibilità dei dati**
  - **Quando e per quanto tempo i dati diventano inaccessibili?**
- **Prestazioni**
  - **Transazioni per unità di tempo.**

Sistemi *Real-time*

- **Sistemi orientati al controllo**
  - **Devono rispondere a eventi esterni entro tempi prefissati, spesso (non sempre!) brevissimi.**
  - **Es1. Sistema controllo edificio: aumento temperatura → allarme**
  - **Es2. Controllo torri di cracking di una raffineria se certe reazioni durano troppo, la torre può esplodere!**
- **Tempi di risposta**
  - **Sono relativi all'efficienza in sistemi generici, ma alla correttezza in sistemi real-time**
- **Affidabilità e correttezza sono essenziali!**

Sistemi Distribuiti

- **Insieme di computer collegati da rete (nodi + connessioni)**
  - **Banda larga e basso tasso di errore → possibilità di distribuire le componenti del sistema.**
  - **Distribuzione di dati e/o funzioni**
- **Sistemi tipicamente Robusti**
  - **Tolleranza a fallimenti dei nodi**
  - **Tolleranza a fallimenti delle connessioni**
- **Rendere distribuito un sistema può migliorare alcune sue proprietà:**
  - **Replicazione dei dati → affidabilità, robustezza**
  - **Distribuzione dei calcoli → prestazioni**

Sistemi Embedded

- **La componente software è *immersa* in uno specifico hardware (non PC) e lo controlla.**
  - **Es. Aereo, frigorifero, ABS, robot, pacemaker**
- **Necessarie varie qualità:**
  - **Efficienza, visto limitata potenza dell'hw**
  - **Affidabilità, sicurezza per sistemi safety-critical**
- **Requisiti meno stringenti per l'usabilità.**
  - **Interfaccia uomo-macchina → interfaccia SW-HW**
- **Il Sistema è più evolvibile se si spostano funzionalità da HW a SW.**

Sistemi Web

- **Sistemi che operano in Internet, protocolli http e https**
- **Intrisicamente distribuiti**
- **Accesso e GUI tramite Web Browser**
- **I siti più popolari possono avere decine di migliaia di accessi al secondo!**
- **Necessarie molte qualità:**
  - **Efficienza e scalabilità, per i siti più popolari**
  - **Protezione da accessi non autorizzati (hacker)**
  - **Usabilità**
  - **Portabilità, per quanto riguarda i browser**
- **Requisiti meno stringenti per sicurezza e interoperabilità**

“Apps”

- **Sono le applicazioni per terminali mobili (Smartphone e Tablet PC)**
- **Utilizzano spesso l'accesso alla rete Internet**
- **Utilizzano funzionalità del terminale (GPS, accelerometri,...)**
- **Necessarie varie qualità:**
  - **Efficienza, visto anche la limitata potenza di alcuni TM**
  - **Protezione da accessi non autorizzati**
  - **Usabilità**
  - **Portabilità, ottenuta tramite lo standard HTML5**
- **Sono spesso applicazioni relativamente piccole**