
# CONCETTI GENERALI DELL'INGEGNERIA DEL SOFTWARE

*Antonio Tufano*

# Indice

1. **INTRODUZIONE** ..... **3**
2. **IL SOFTWARE** ..... **5**
3. **L'INGEGNERIA DEL SOFTWARE** ..... **6**
4. **PRINCIPI DELL'INGEGNERIA DEL SOFTWARE** ..... **8**
5. **IL PROCESSO SOFTWARE** ..... **9**
6. **LA QUALITÀ DEL SOFTWARE** ..... **11**
7. **I MITI DEL SOFTWARE** ..... **13**

# 1. INTRODUZIONE

Le economie di tutti i paesi sviluppati sono dipendenti dal software. Il Software rappresenta una delle voci rilevanti del PIL di tutti i paesi sviluppati. Ogni giorno un numero sempre più alto di sistemi contengono software o dipendono da software. Il software costa più dell'hardware, e il mantenimento costa più dello sviluppo. L'obiettivo è quello di sviluppare software tenendo in considerazione i costi.

Il software negli anni si è evoluto ed abbraccia un'ampia gamma di settori. Ad esempio

- Arte: applicazioni sviluppate da singole persone e utilizzate dagli stessi sviluppatori.
- Artigianato: applicazioni sviluppate da piccoli gruppi specializzati per un cliente.
- Industria: diffusione del software in diversi settori; crescita di dimensioni, complessità e criticità delle applicazioni; mercato e concorrenza; necessità di migliorare la produttività e la qualità; gestione dei progetti; evoluzione del software.

  Durante gli anni 60 si è avuta una crisi del software.

Fino ad allora il Software era rappresentato da programmi (sviluppati informalmente) – ad es., per risolvere sistemi di equazioni e si è trasformato in programmi per 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) e pertanto è diventato necessario un nuovo approccio, ingegneristico, con strumenti e tecniche opportuni. Si sono evidenziati pertanto diversi problemi relativi allo 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. Tutto questo ha rappresentato una vera e propria crisi del software che non si è rilevata essere solo temporanea. La disciplina dell'Ingegneria del Software aiuta a risolvere la crisi.

**Gli stessi problemi esistono tuttora – perché?** mentre migliora l'abilità generale nella produzione del software, aumenta anche la complessità dei sistemi software. Esistono oggi nuove tecnologie e maggiori capacità di calcolo motivano nuove esigenze e nuovi obiettivi. Le prestazioni dell'hardware crescono, e crescono le aspettative sui sistemi che comprendono hardware e software. A questo punto in che modo cresce la capacità di produrre software?

# 2. IL SOFTWARE

## Programmi vs Prodotti

**Un Programma** in genere l'autore è anche l'utente. Non è documentato, quasi mai è testato, non c'è nessun progetto e pertanto non è necessario un approccio formale.

Un **Prodotto software invece generalmente viene** usato da persone diverse da chi lo ha sviluppato. È un software industriale il cui costo è circa 10 volte il costo del corrispondente programma. Si rende così necessario un approccio formale allo sviluppo.

## Un Prodotto software include

- **Prodotti generici**: sistemi stand-alone prodotti da una organizzazione e venduti a un mercato di massa
- **Prodotti specifici**: sistemi commissionati da uno specifico utente e sviluppati specificatamente per questo da un qualche contraente



La fetta maggiore della spesa è nei prodotti generici ma il maggior sforzo di sviluppo è nei prodotti specifici. La differenza principale? Chi dà la specifica del prodotto (il produttore o il consumatore).

**Il Software** differisce dai prodotti industriali classici perché è: intangibile, malleabile, ad alta intensità di lavoro umano, spesso costruito ad hoc invece che assemblato, e la sua manutenzione corrisponde al cambiamento del prodotto.

# 3. L'INGEGNERIA DEL SOFTWARE

**"Software engineering"** è una disciplina che cerca di fornire le regole per il processo di produzione del software. Un ingegnere del software dovrebbe:

a. adottare un approccio sistematico e organizzato al proprio lavoro;
b. usare strumenti e tecniche appropriate, che dipendono dal problema che deve essere risolto, dai vincoli presenti e dalle risorse disponibili.

L'ingegneria del software è una attività di modellazione, è una attività per risolvere problemi, è una attività per acquisire conoscenza, non è un processo lineare ed è una attività rational-driven in quanto è necessario catturare e capire il contesto in cui le decisioni vengono prese e le motivazioni. L'Ingegneria del software differisce dal concetto di informatica in quanto: l'informatica è una scienza: il "cuore" sono i fondamenti teorici: linguaggi, algoritmi, complessità, formalismi, ecc. L'ingegneria del software ha a che fare con aspetti più "pratici": come pianificare e sviluppare la produzione di software di qualità. Ad un ingegnere del software le conoscenze di base dell'informatica servono quanto la fisica ad un ingegnere elettrico.

L'Ingegneria del software riguarda la costruzione di software di grandi dimensioni, di notevole complessità, sviluppati tramite lavoro di gruppo. I progetti software di questo tipo hanno tipicamente versioni multiple, lunga durata, frequenti cambiamenti, eliminazione di difetti, adattamento a nuovi ambienti, miglioramenti e nuove funzionalità.

Software Engineering è parte della System Engineering. La maggior parte del SW è collocata all'interno di un "sistema" misto HW/ SW. L'obiettivo finale di chi produce software è creare un sistema che

soddisfi globalmente i requisiti dell'utente. È necessario pertanto un coinvolgimento nella definizione dei requisiti del sistema. La conoscenza del dominio applicativo è essenziale per un efficace sviluppo del SW. Il SW è utile quando riesce a condensare nei suoi algoritmi la conoscenza del dominio applicativo, altrimenti inutile o dannoso.

# 4. PRINCIPI DELL'INGEGNERIA DEL SOFTWARE

**Rigore e formalità**

Rigore: concetto primitivo (precisione, accuratezza come complemento alla creatività dello sviluppo). Formalità: oltre il rigore. Fondamento matematico. In ogni fase del processo di produzione del SW bisogna definire il livello di rigore/ formalità necessario

**Separazione di aspetti diversi:** affrontare separatamente i vari aspetti di un problema complessi rispetto al tempo, rispetto a diverse proprietà del sistema, rispetto a diverse viste del sistema, rispetto a diverse componenti del sistema

**Modularità:** suddividere un sistema complesso in parti più semplici. Bottom- up vs top- down. Gli obiettivi sono: decomporre il problema (divide et impera), comporre le soluzioni ai sotto-problemi per comprendere meglio il sistema globale. Bisogna avere high cohesion (alta coesione) all'interno del modulo e low coupling (basso accoppiamento): poca interdipendenza fra moduli.

**Astrazione:** si identificano gli aspetti cruciali in un certo istante ignorando gli altri.

**Anticipazione del cambiamento**: la progettazione deve favorire l'evoluzione del SW.

**Generalità**: tentare di risolvere il problema nella sua accezione più generale: più semplice o più complesso del problema originario. Per far ciò è necessario un trade-offs tra fattibilità, costo, beneficio.

**Incrementalità**: lavorare per passi successivi. Definito un sottoinsieme iniziale, una consegna iniziale e dei feedback iniziali si possono aggiungere molte features in modo incrementale.

# 5. IL PROCESSO SOFTWARE

Insieme organizzato di attività che sovrintendono alla costruzione di un prodotto software da parte del team di sviluppo utilizzando metodi, tecniche, metodologie e strumenti adatti alla sua realizzazione. È suddiviso in varie fasi secondo uno schema di riferimento (il ciclo di vita del software). E' descritto da un modello: informale, semi-formale o formale (maturità del processo).

**Le attività richieste nel processo di sviluppo software** sono: specifica, progettazione, implementazione, validazione, installazione, manutenzione, smaltimento.

**I problemi nel processo di sviluppo del software** sono rappresentati principalmente da specifiche incomplete e o incoerenti, dalla mancanza di distinzione tra specifica, progettazione e implementazione, dalla 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.

Esistono dei Sistemi Software che intendono fornire un supporto automatico per le attività di un processo software. **CASE (Computer-Aided Software Engineering).**

Ci sono due tipologie: **Upper-CASE** - Strumenti che supportano le attività delle fasi di analisi e specifica dei requisiti e progettazione di un processo software. Includono editor grafici per sviluppare modelli di sistema, dizionari dei dati per gestire entità del progetto; **Lower-CASE** - Strumenti che supportano le attività delle fasi finali del processo, come programming, testing e debugging. Includono generatori di graphical UI per la costruzione di interfacce utente,

debuggers per supportare la ricerca di program fault, traduttori automatici per generare nuove versioni di un programma.

# 6. LA QUALITÀ DEL SOFTWARE

La qualità si può riferire sia al prodotto sia al processo. La qualità del prodotto si misura in termini di requisiti. La mancanza di conformità ai requisiti indica mancanza di qualità. Gli standard definiscono i criteri a cui attenersi nello sviluppo del software e non seguire tali criteri produce una qualità insufficiente. Esistono anche dei requisiti impliciti, spesso taciuti (es. facilità di manutenzione) che ai fini della qualità, sono da rispettare quanto quelli espliciti. Ci sono diversi fattori (o caratteristiche) di qualità.

Qualità interne (intrinseche alla struttura del software).

Qualità esterne (percepite dagli utenti).

Per valutare le varie caratteristiche del software sono necessarie delle metriche. Le caratteristiche interne sono direttamente misurabili e sono usate per misurare indirettamente le caratteristiche esterne su cui impattano. Le caratteristiche interne sono esaminabilità, accuratezza, uniformità comunicativa, completezza, concisione, consistenza, uniformità dei dati, tolleranza agli errori, efficienza di esecuzione, espandibilità, generalità, indipendenza rispetto all'hardware, strumentazione, modularità, operabilità, sicurezza, autodocumentazione, semplicità, indipendenza dal sistema software, tracciabilità, addestramento. In particolare le principali misure interne sono:

- **Efficienza** Un sistema è efficiente se usa le risorse HW/SW in modo proporzionato ai servizi che svolge.
- **Riparabilità** Un sistema è riparabile se la correzione degli errori è poco faticosa. La riparabilità si persegue attraverso la modularizzazione. Non conta tanto il numero, ma piuttosto il come sono organizzati tra loro e al loro interno.

- **Riusabilità** Facilità con cui è possibile riusare parti di sistema per realizzare un prodotto diverso.
- **Correttezza:** un software si dice corretto se si comporta in accordo a quanto previsto dalla sua specifica dei requisiti.
- **Affidabilità**: un sistema è tanto più affidabile quanto più raramente, durante l'uso del sistema, si manifestano malfunzionamenti.
- **Usabilità**: un sistema è facile da usare se un essere umano lo reputa tale.
- **Scalabilità**: un sistema è scalabile se può essere adattato a diversi contesti con forti differenze di complessità (per esempio database molto piccoli o molto grandi) senza che questo richieda la riprogettazione dello stesso sistema.

Mentre le caratteristiche esterne sono:

- **Robustezza**: la robustezza di un sistema è la misura in cui il sistema si comporta in modo ragionevole in situazioni impreviste, non contemplate dalle specifiche.
- **Portabilità**: un sistema è portabile se è in grado di funzionare in ambienti diversi.
- **Verificabilità:** un sistema è verificabile se le sue proprietà: correttezza, affidabilità, sono facili da verificare.
- **Manutenibilità:** facilità di apportare modifiche a sistema realizzato.

# 7. I MITI DEL SOFTWARE

**Miti del software** riguardano 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.

**Miti della clientela** un'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 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.

In conclusione l'ingegneria del software non si occupa di creare di documenti, piuttosto, si occupa di creare qualità. Una migliore qualità porta a minor lavoro e un minor lavoro porta a tempi di consegna più rapidi.