1 Introduzione all'Ingegneria del Software.

In questa prima parte viene definita l'Ingegneria del Software o Software Engineering (SWE), vengono presentate le caratteristiche del ciclo di vita di un prodotto software e proposto un modello di realizzazione.

1.1 Evoluzione del software.

I primi calcolatori sono stati sviluppati negli anni 40 dello scorso secolo, allora non vi era distinzione tra hardware e software, la "programmazione" avveniva con il saldatore.
Negli anni 50 furono resi disponibili gli strumenti di programmazione primordiali, prima i linguaggi assemblativi, un grosso passo avanti dal punto di vista della programmazione rispetto il linguaggio macchina, poi linguaggi di alto livello come Fortran e Cobol.
Si cominciò così a sviluppare l'attività di programmazione e la produzione del software.
La programmazione comunque avveniva a livello individuale: un programma era di dimensioni limitate, e veniva sviluppato e mantenuto da una persona.
Il software veniva sviluppato per applicazioni scientifiche (Fortran) e per applicazioni gestionali (Cobol).
Questo modo di intendere a programmazione è continuato fino alla fine degli anni 60, periodo della crisi del software.

Come conseguenza della costruzione di macchine sempre più potenti e delle richieste del mercato, venivano realizzati programmi sempre più complessi che ad un certo punto travalicarono il campo della programmazione individuale.
In particolare questo si verificò per due tipi di applicazioni:

- i sistemi operativi per grossi calcolatori (IBM 360), il sistema operativo dei quali conteneva milioni di righe, mentre i programmi Fortran tipici del decennio precedente contenevano tipicamente dalle 100 alle 1000 righe.
- i sistemi informativi aziendali destinati a gestire in modo centralizzato tutte le informazioni delle funzioni aziendali, sistemi quindi enormi e molto complessi (fece clamore un sistema IBM commissionato per la Zanussi che non entrò in funzione perché non corrispondente alle effettive esigenze a causa di errori al suo interno).

Il problema dal quale derivava la crisi del software era il passaggio dalla programmazione individuale alla programmazione di squadra, ciò provocò una reazione negativa del mercato, il software subì una flessione causata dalla sfiducia degli acquirenti.
Una soluzione parziale al problema derivò dal cambiamento del modo di sviluppare il software: da prodotto artigianale il software doveva diventare un prodotto industriale.
Quando la programmazione era un'attività individuale, essa era costituita da abilità tecnica, da trucchi, quindi il programmatore si poteva definire un artigiano.
Per la complessità che il software via via richiedeva questa figura non era più sufficiente, il software doveva diventare il prodotto di un processo industriale, con compiti differenziati del personale coinvolto nel processo, metodologie di sviluppo, era già disponibile l'esperienza maturata negli altri settori industriali.
Nacque così l'ingegneria del software (SWE), un modo di vedere il software come prodotto industriale.
Il SWE è un modo di produrre software, il software non implica SWE: programmi di piccole

dimensioni non hanno bisogno della metodologia del SWE.
La crisi del software ha avuto come conseguenza la forte lievitazione del costo del software.
Tutte le caratteristiche di un prodotto software si possono ricondurre ad un parametro: il costo.
Il problema del SWE è quello di minimizzare i costi di un prodotto software.
Il costo non si riferisce solo a quello che viene pagato al produttore di software.
Ad esempio, se un prodotto viene rilasciato ad un cliente che lo prova accorgendosi che non funziona e lo rifiuta si ha un costo enorme, se un per un errore di programmazione del sistema informatico di una torre di controllo due aerei si scontrano si ha un costo non valutabile in vite umane.
Scopo del SWE è quindi di produrre prodotti di qualità che certamente costano di più inizialmente in termini monetari, ma che sono più sicuri ed affidabili. Si minimizza però il costo complessivo di un prodotto software.

Perché viene acquistato un prodotto software?
Viene acquistato se i benefici che produrrà saranno maggiori del suo costo di acquisto (analisi costi/benefici).
Analizziamo ora i tipici problemi che presentava il software prima dell'introduzione del SWE.

- Tempi di consegna. Non venivano rispettati (ad es. il successo del MacIntosch fu ritardato di molti anni a causa di ritardi software). Questo è un problema ancora attuale, anche se molto attenuato.
- Costi di produzione. Non venivano rispettati, non era raro che dal momento della commissione al momento della consegna i costi si fossero moltiplicati per 10.
- Bisogni dell'utente. Era frequentissima la mancata soddisfazione dei bisogni dell'utente: veniva consegnato un prodotto che faceva cose diverse rispetto i bisogni.
- Scarsa affidabilità. Il sistema corrispondeva ai requisiti dell'utente ma si bloccava spesso, oppure perdeva interi database.
- Rigidità. Al più piccolo cambiamento delle necessità dell'utente il sistema non era più adatto ed era difficile o impossibile modificarlo.

A cosa sono dovuti questi tipici problemi?

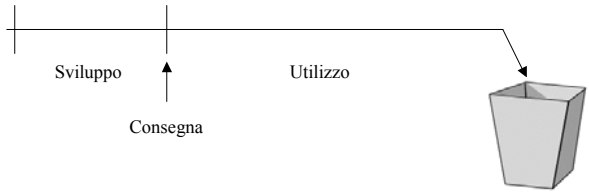
- Tempi di consegna e Costi di produzione. Questi problemi sono dovuti al fatto che il produttore non ha le idee chiare su come deve avvenire il processo produttivo.
  E' quindi necessario un modello teorico ed affidabile del processo di produzione del software.
- Bisogni dell'utente. Problemi derivano da una cattiva comunicazione tra cliente e produttore: il produttore non comprende ciò che il cliente vuole ottenere ed il cliente non comprende ciò che il produttore gli propone. Vi è quindi una mancata comprensione dei requisiti.
- Scarsa affidabilità. Deriva da errori non trovati in fase di testing. Può essere conseguenza di una cattiva organizzazione del programma, che rende il test più complesso, ma può anche interessare l'hardware del calcolatore (memoria) oppure le caratteristiche del sistema operativo (mancanza delle performance desiderate).
- Rigidità. Dipende da problemi di accoppiamento tra parti e mancanza di modularità. Si verifica per la poca indipendenza tra i moduli del programma e denota quindi un forte accoppiamento.

Per fare un buon prodotto software un produttore deve quindi riuscire a:
indovinare i costi,
consegnare il prodotto in tempo,
soddisfare i requisiti del cliente,
con un prodotto affidabile
e modificabile.

1.2 Ciclo di vita del software.

Definiamo ora un modello del processo produttivo del software.
Il ciclo di vita del software, concetto importato da altri settori manifatturieri, non tiene conto solamente della fase di produzione, ma anche di quella di utilizzo.
Il ciclo di vita di un prodotto software è suddiviso in due parti.

Ciclo di vita di un prodotto software



Lo sviluppo avviene nell'azienda del produttore, la sua durata può essere anche di qualche anno, ma è solitamente meno lunga della fase di utilizzo.
La fase di utilizzo inizia al momento della consegna del prodotto e dura fino a quando viene eliminato.
L'utilizzo avviene nell'azienda del cliente, ma il produttore è spesso chiamato in causa per interventi di manutenzione.

Cosa si intende per manutenzione?
Per manutenzione di un prodotto software si intendono i diversi tipi di intervento che si rendono necessari durante la fase di utilizzo:

- Eliminare errori residui, trovati dopo la consegna. E' molto difficile, se non impossibile, produrre software privo di errori. (ad esempio alcune misure indicano che i programmi di

mercato hanno in media 40 errori ogni 1000 righe di codice, la maggior parte dei quali si verificano solamente in situazioni molto particolari ed improbabili). Quindi il produttore sa che consegna un prodotto contenente degli errori, nel contempo deve fare in modo che siano facilmente eliminabili.

- Adattare a nuove configurazioni (hardware e software di base). Visto anche il suo costo, il tempo di utilizzo del software è più lungo del tempo di utilizzo dell'hardware. Il produttore non deve comunque vincolare l'utente ad operare con un hardware che può rapidamente diventare obsoleto. Questo punto deve essere esplicitamente contrattato tra produttore ed utente (ad esempio devono sempre essere dichiarate le caratteristiche minime dell'hardware necessario per un certo prodotto).
- Adattare a cambiamenti dell'ambiente applicativo. I requisiti del cliente cambiano nel tempo e deve essere possibile modificare almeno parzialmente il prodotto per soddisfare gli stessi.

La manutenzione è quindi il complesso degli interventi pianificati, progettati ed eseguiti su un prodotto software, già operante ed accettato dal committente (e quindi rilasciato dal processo di sviluppo), rivolti a:

- ripristinarne le funzionalità in seguito a malfunzionamenti (manutenzione correttiva);
- migliorarne le prestazioni, l'organizzazione o la qualità (manutenzione migliorativa);
- adeguarlo alle innovazioni tecnologiche (manutenzione adeguativa);
- renderlo rispondente alle nuove esigenze del committente, o a nuove normative, che comportino variazioni al software esistente, od implichino la necessità di integrazione con nuove funzioni, senza variarne gli obiettivi primari (manutenzione ordinaria);
- renderlo rispondente alle nuove esigenze del committente, o modifiche di normative o regolamenti che comportino lo sviluppo di nuove funzionalità o rendano necessari interventi di manutenzione sul prodotto già in gestione, che trascendano le modifiche di manutenzione ordinaria (manutenzione evolutiva).

Come cambiano le esigenze del cliente?

Automatizzare una parte del sistema informativo del cliente vuol dire fotografare la realtà informativa della sua organizzazione (analisi del flusso dell'informazione) e immetterne la logica in un calcolatore: questo è il software.
L'introduzione del prodotto nell'organizzazione modifica però la realtà informativa, cambiano pertanto modalità ed esigenze.
Nel contempo aumenta la cultura informatica dell'utente che vuole quindi sfruttare meglio il suo sistema informatico, viene stimolato a concepire nuove funzionalità da automatizzare.
Le esigenze in genere sono proporzionali alle risorse disponibili (ad esempio alla potenza di calcolo, allo spazio di memoria di massa disponibile: più spazio è disponibile più in fretta si riempie).

Quali sono le ricadute sui costi?

Ogni volta che il cliente interagisce con il produttore, deve sostenere dei costi.
I costi sono distribuiti durante tutto il ciclo di vita del software, anzi generalmente la maggioranza dei costi ricade nella parte di utilizzo.
Vi sono diversi tipi di costi:

- Costi diretti quando il cliente richiama il produttore per effettuare una modifica del prodotto
- Costi indiretti dovuti all'inutilizzabilità del sistema in seguito al verificarsi di un errore.

Pertanto abbattere i costi globali significa abbattere in particolare i costi di utilizzo.
L'idea che si trova alla base del SWE è aumentare i costi di sviluppo per diminuire i costi di utilizzo.

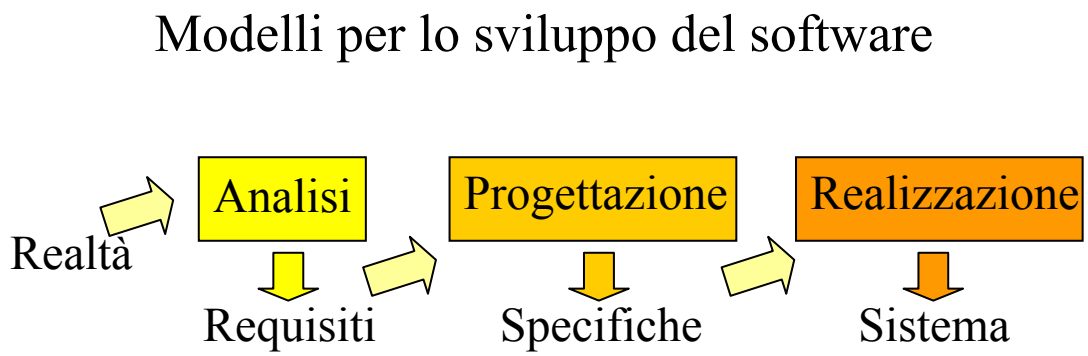
Sicuramente il costo complessivo diviene così più basso.
Il cliente deve spendere quindi di più inizialmente per un prodotto di qualità maggiore, mentre il costo di utilizzo diventa molto più lieve.

Un software che rende minimi i costi di utilizzo deve essere:

- non rigido,
- affidabile e
- modificabile.

1.3 Modelli per lo sviluppo del software.

La fase di sviluppo riveste una notevole importanza per il SWE. Consideriamo quindi un modello della fase di sviluppo del software.
Nel campo delle scienze esatte un modello è una formalizzazione precisa di una realtà mediante equazioni e viene utilizzato quando è corretto.

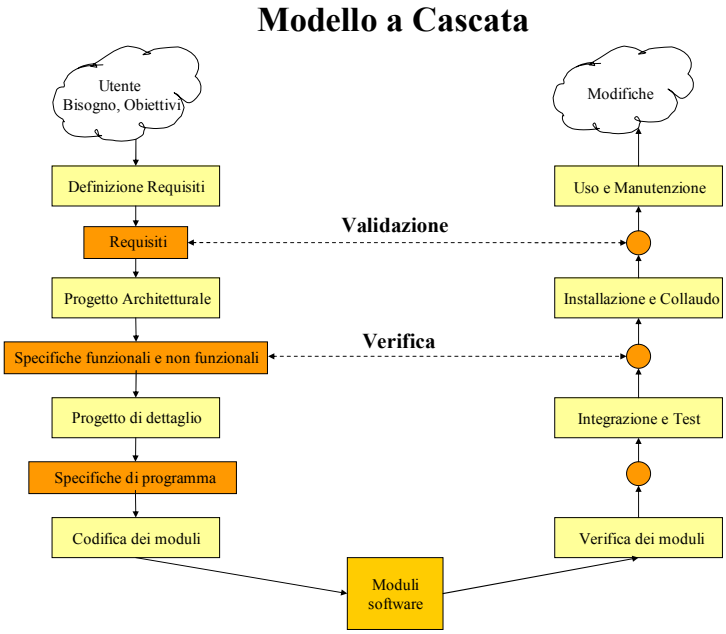


Nel campo che prendiamo in considerazione il modello è chiaramente meno preciso, la realtà si può discostare senza però farne cadere la validità.
Ad esempio un modello degli errori umani non può che essere impreciso. In questi casi un modello si applica quando è meglio di niente.
Ciò è particolarmente vero per il SWE visto che inizialmente l'assenza di modelli ha portato alla crisi del software.
Analizziamo alcuni modelli di sviluppo del software:

- il modello a cascata
- ciclo di prototipizzazione
- modello esplorativo
- altri modelli

1.3.1 Il modello a cascata.

Il modello a cascata riveste una particolare importanza per la sua utilizzabilità e diffusione.
Questo modello si snoda attraverso diverse fasi che ora analizzeremo nel dettaglio, riferendoci allo schema riportato.



Utente.
Il (futuro) utente decide di dotarsi di un software, nella sua mente possiamo distinguere due componenti:

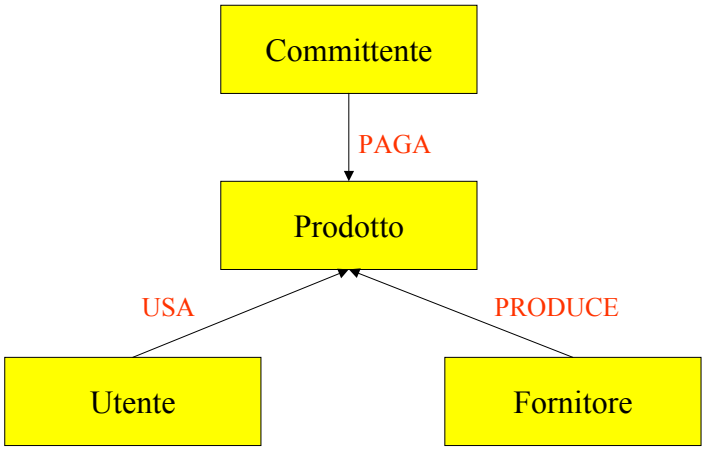
- Il bisogno: cioè la percezione di una differenza tra lo stato presente della realtà e lo stato nel quale si potrebbe trovare. Il bisogno è quindi uno stato di insoddisfazione circa lo stato presente. Ad esempio il cliente "vede" che un concorrente produce a costi dimezzati, il suo desiderio sarà quindi di soddisfare il suo bisogno di ottenere lo stesso risultato.
- Gli obiettivi. Quindi una strada da percorrere per superare un bisogno. Gli obiettivi che prendiamo in considerazione sono ovviamente solo quelli che riguardano l'introduzione di un sistema informativo nell'organizzazione del cliente.

Definizione dei requisiti.
Il nostro cliente ha deciso di informatizzarsi, va quindi dal produttore per commissionare il prodotto desiderato, che è ancora estremamente vago nella mente del cliente.
I ruoli di "cliente" e "produttore" non devono trarre in inganno, in quanto il "cliente" potrebbe avere all'interno della propria azienda una funzione sistemi informativi, alla quale commissionare il prodotto desiderato, il "produttore" può quindi essere chiamato più correttamente "fornitore".
Bisogna distinguere i vari ruoli:

- il fornitore o produttore
- il committente: chi paga
- l'utente: chi userà il prodotto

Bisogni ed obiettivi sono propri del committente, il fornitore deve costruire il prodotto in base a quanto richiesto dagli utenti a diretto contatto con il sistema informativo dell'azienda.

Ruoli nel Modello a Cascata



Il committente si reca quindi dal fornitore e gli illustra il suo problema.
Il fornitore cerca allora di capire quali sono i requisiti del prodotto che soddisfano i suoi bisogni ed obiettivi. Per fare questo deve trattare direttamente con gli utenti.
Viene pertanto analizzato lo stato presente dell'azienda del committente, il suo sistema informativo attuale, che può essere informatizzato o meno.
Questa è la fase di definizione dei requisiti che produce un documento dei requisiti.

Documento dei requisiti.
Si tratta del documento nel quale è descritto in modo preciso ma comprensibile al cliente/utente ciò che farà il sistema.
Se l'utente è soddisfatto dal prodotto che viene offerto dal fornitore, il documento dei requisiti può portare ad un contratto.
A volte però l'utente desidera avere qualcosa di più concreto, in questi casi è necessario lo sviluppo di un prototipo.

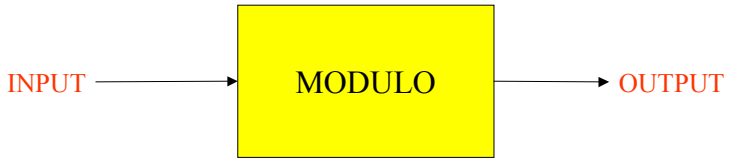
Progetto architetturale.
Durante la fase di progetto architetturale vengono definite le specifiche del prodotto. Vengono riformulati i requisiti in modo altamente formale per l'uso interno nel processo del fornitore, in modo da definire in modo preciso che cosa deve fare il software da produrre.
Il software può essere suddiviso in moduli ad alto livello.
Il risultato di questa fase è il documento delle specifiche.

Documento delle specifiche.
Il documento delle specifiche descrive in modo formale ciò che farà il software e le sue parti/moduli.

Le specifiche sono di due tipi:

- Specifiche funzionali che descrivono una funzionalità del software. Ogni modulo è visto come una funzione e nel caso più semplice viene descritto il rapporto tra l'input e l'output.

Modulo/Funzione



- Specifiche non funzionali che rappresentano i vincoli ai quali deve sottostare il software. Ad esempio: per compatibilità con un sistema esistente si vuole che l'input sia formulato secondo una particolare codifica (ad esempio XML); per un sistema real-time che l'output sia prodotto in meno di mezzo secondo.

Progetto dettagliato.
Durante la fase di progetto dettagliato vengono definiti in modo preciso i moduli che comporranno il software e la loro struttura interna. Viene definita la loro interfaccia ed i dati globali ai quali possono accedere.
Si fanno quindi le scelte su come realizzare il software, che vengono riportate nel documento delle specifiche di programma.

Documento specifiche di programma.
Il documento specifiche di programma contiene il dettaglio di ciò che deve essere realizzato e viene utilizzato nella successiva fase di codifica.

Codifica dei moduli.
Nella fase di codifica vengono prodotti i moduli software.
Si tratta di una fase piuttosto semplice, tecnica, tutte le decisioni principali sono state prese nelle fasi precedenti.

Moduli software.
Si tratta dei singoli moduli e componenti che andranno a costituire il prodotto finale.

Verifica dei moduli.
In questa fase viene effettuata la verifica dei singoli moduli per eliminare errori di realizzazione locali e modulari.
Si procede in due modalità:

- Testing: esecuzione del modulo su campioni di dati.
- Verifica statica: lettura del software con particolari criteri. Per sapere se il comportamento di un modulo è corretto si confronta con la parte di specifiche di programma che lo riguardano, le quali devono chiaramente essere formulate in modo molto preciso.

Integrazione e test.
Quando ogni modulo è individualmente corretto si passa all'integrazione dei moduli.
Viene quindi testato il sistema complessivo: viene confrontato il documento delle specifiche funzionali e non funzionali.
Verification: "Are we building hte product right?"

Installazione e collaudo.
Quando il sistema si comporta in accordo con il documento delle specifiche, si passa all'installazione dello stesso nel suo ambiente e si procede al collaudo.
Il collaudo comprova da parte dell'utente che il prodotto soddisfa i suoi bisogni ed obiettivi.
A questo scopo si confronta il sistema con il documento dei requisiti.
Validation: "Are we building the right product?"

Uso e mantenimento.
Infine si continua con la fase di uso e mantenimento che comporta un confronto con i nuovi bisogni dell'utente.

1.3.1.1 Costo degli errori.

L'esito finale della fase di sviluppo è la soddisfazione dei bisogni dell'utente.
Se l'utente non è soddisfatto ci sono stati degli errori nella fase di sviluppo.

Qual è l'incidenza dei vari tipi di errori sul costo del prodotto?.

Errore nella fase di definizione dei requisiti.
Un errore in questa fase si ripercuote sulle fase successive.
Anche se i successivi passi sono svolti in modo corretto il prodotto non potrà soddisfare l'utente. Pur essendo funzionante.
Ci si accorge di un tale errore al momento del collaudo. *Validation: "we don't build the right product".*

Errore nella fase di progetto architetturale.
Un errore questa fase riguarda le specifiche.
Se la specifica implicata è funzionale significa che è stata sbagliata la descrizione formale di una funzionalità. Ad esempio un modulo doveva effettuare il calcolo dell'età ed invece è stato fatto calcolare l'anno di nascita, il modulo è corretto , ma quando un altro modulo si aspetta la restituzione dell'età, ottiene invece l'anno di nascita.
Ci si accorge di questi errori al momento dell'integrazione e test del sistema integrato. *Verification: "we don't build the product right".*

Errore nella fase di progetto dettagliato.
Si tratta di errori riguardanti i moduli: un modulo non esegue ciò che gli viene richiesto.
Ci si accorge di questi errori in fase di verifica e test.

Errore durante la codifica.
L'errore viene segnalato direttamente dall'ambiente di sviluppo o compilatore.

Gli errori più costosi sono quelli che si verificano ai livelli più astratti: prima si sbaglia, più tardi intercetta l'errore, più costoso diventa effettuare la correzione dello stesso.
Gli errori precoci sono assolutamente da evitare, dedicando particolare cura alle prime fasi.
Spesso si procede subito con la codifica, è invece conveniente investire nelle fasi iniziali di analisi e definizione dei requisiti.
Una modalità che consente di ridurre il costo degli errori, specialmente quelli precoci, consiste nell'integrare il modello a cascata con una fase di prototipizzazione o con uno sviluppo top-down, in modo da anticipare parte delle attività delle fasi successive, compresa la rilevazione degli errori.
Nel diagramma del modello a cascata le varie attività appaiono come "scatole" monolitiche.
Progettazione, definizione dei requisiti, produzione delle specifiche, ecc. richiedono in realtà attività cicliche, iterative: si svolgono le attività affinando il risultato più volte fino ad ottenere un risultato soddisfacente.
Un'altra caratteristica del modello a cascata è che non vengono sviluppate strade alternative in parallelo: viene scelta una soluzione e sviluppata, se non porta ai risultati sperati la si abbandona per sviluppare ex novo una soluzione alternativa.
Seguire soluzioni alternative è utile in particolare nelle prime fasi per esplorare strade diverse prima di scegliere quella ritenuta più opportuna.

1.3.1.2 Applicabilità del modello a cascata.

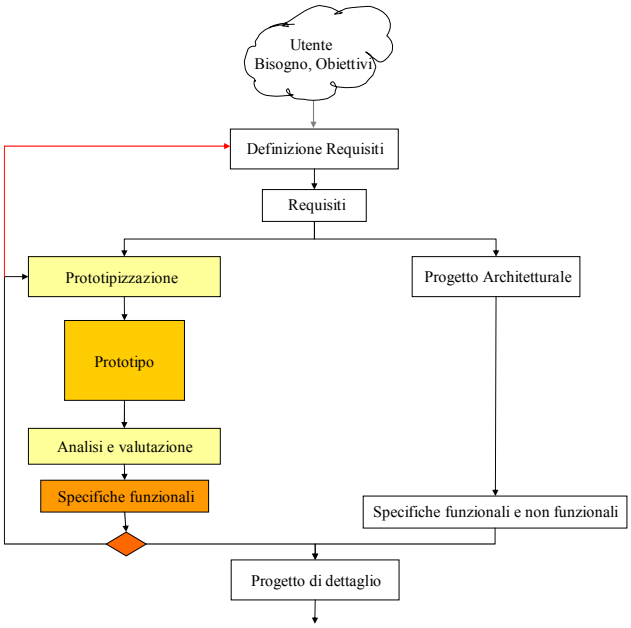
Il modello a cascata è applicabile se è possibile specificare completamente il sistema prima della fase di codifica.
Bisogna cioè saper fare il prodotto prima di realizzarlo.

Vi sono casi nei quali questa situazione non è applicabile (ad esempio per lo sviluppo di un sistema esperto di diagnostica oppure di un sistema di relazione con i clienti senza conoscere le caratteristiche di questi ultimi).
In questi casi bisogna utilizzare anche un modello esplorativo.

1.3.2 Ciclo di prototipizzazione.

Il ciclo di prototipizzazione viene aggiunto al modello a cascata per ridurre notevolmente (al limite azzerare) le probabilità di errore nella definizione dei requisiti ed in parte anche nella definizione delle specifiche.
Nello schema viene evidenziata la variazione che esso apporta al modello a cascata: si può considerare come una possibile esplorazione della fase di progettazione architetturale.

Ciclo di Prototipizzazione (integrato nel Modello a Cascata)



Al momento dell'attuazione di una richiesta del cliente, quando il fornitore non è certo di capire esattamente ciò che vuole l'utente, o su richiesta dell'utente stesso, viene anticipata una fase di codifica e di verifica all'inizio del ciclo di sviluppo del prodotto.
Viene quindi sviluppato un prototipo del prodotto: un sistema in grado di mostrare tutte le funzionalità del prodotto finale, ma che non rispetta le specifiche non funzionali.
Ad esempio lento e poco affidabile, oppure riguarda il solo sviluppo dell'interfaccia utente per la verifica delle modalità di interazione: ciò che l'utente vede è per lui il sistema.
Una caratteristica importante del prototipo è che deve costare poco rispetto al prodotto finito.
Il prototipo è sviluppato per capire le caratteristiche che l'utente vuole dal prodotto finale.
A questo scopo viene mostrato all'utente e commentato assieme al cliente.

Il prototipo, una volta realizzato e verificato con l'utente, esaurisce il suo scopo e viene buttato via: normalmente le sue parti non vengono utilizzate durante la realizzazione del prodotto finale, principalmente perché non ha i requisiti di qualità propri del prodotto finito.
Il ciclo di prototipizzazione è una variante del modello a cascata, va quindi applicato alla stessa classe di problemi.

1.3.3 Modello esplorativo

Se non si possono determinare le specifiche, non può essere definita la correttezza del sistema, ossia la corrispondenza con le specifiche.
In questi casi viene utilizzato il concetto di adeguatezza, ossia corrispondenza con i bisogni e gli obiettivi del cliente.
Il modello esplorativo consiste in una successione di prototipi che viene fatta convergere verso una soluzione adeguata, soddisfacente per il cliente.
Ad ogni passo viene costruito un prototipo e presentato alla valutazione critica del cliente, che lo può accettare come adeguato alle sue esigenze, oppure può indicare nuovi criteri e requisiti che il sistema deve possedere.
Nei passi finali il prototipo viene affinato ad ogni ciclo finché viene accettato dal cliente.
Questo modello presenta due caratteristiche salienti:

- Non vi è distinzione tra prototipo finale e prodotto. Questo normalmente non risulta vantaggioso per il fornitore.
- Vi è un forte coinvolgimento del cliente nella fase di sviluppo rispetto al modello a cascata. Con conseguenti costi anche per il cliente.

1.3.4 Altri modelli.

Vediamo ancora due modelli di sviluppo del software che interessano sia gli strumenti che la metodologia di approccio al progetto di sviluppo.

Trasformazioni formali.
Sta alla base degli strumenti di sviluppo integrato che comprendono anche la fase di progettazione.
Lo scopo è l'eliminazione del costo di correzione degli errori rilevati durante la fase di integrazione e test (verification) scrivendo le specifiche in modo completamente formale. Si vuole cioè arrivare alla costruzione del codice mediante una sequenza di trasformazioni (di tipo matematico).
Se le specifiche sono corrette è garantito in questo modo che il codice è corretto.
Il punto di arrivo è la generazione automatica del codice a partire dalle specifiche (strumenti RAD).
Questo metodo è applicabile ale specifiche funzionali ma non alle specifiche non funzionali.

Riutilizzo del software.
I metodi visti in precedenza riguardano la costruzione ex novo di un prodotto interamente nuovo per ogni specifica definita.
Con il miglioramento di strumenti e linguaggi di sviluppo, nel corso del tempo il software si accumula, risolvendo funzionalità di uso comune, e vengono messe a disposizione componenti di validità generale, raccolte spesso in librerie.
Il riutilizzo del software è un'idea che si basa sulla costruzione di nuovo software semplicemente assemblando componenti esistenti e sviluppando software di integrazione delle stesse.
Perché ciò possa avvenire vantaggiosamente sono necessari strumenti automatici di supporto:

- le componenti devono essere mantenute in un Database
- devono essere corredate da un'ottima documentazione
- è necessario un sistema di Information Retrieval per permetterne il recupero a seconda delle necessità