

Studio di fattibilità:

Viene fatto uno studio preliminare sui costi che tale progetto comporterebbe e vengono valutate le alternative migliori.

Viene stilato un documento che definisce il problema ed eventuali scenari che illustrino le possibili soluzioni.

Viene fatto un preventivo dei costi valutato in tempo, attrezzature e modalità di sviluppo per il cliente.

Analisi e specifica dei requisiti:

Bisogna parlare con il committente e cercare di capire cosa si aspetta dal software.

In seguito a svariati colloqui con il committente si redige il Documento dei Requisiti, che deve essere il più completo e preciso possibile.

Il Documento dei Requisiti dice quali funzionalità deve avere il nostro software, e non ci devono assolutamente essere rimandi a come implementare il progetto.

Il cliente deve leggere attentamente il Documento dei Requisiti e firmarlo, e una volta avuta la sua approvazione si ha un punto di partenza per proseguire nel progetto: se il committente più avanti dirà che le specifiche sono errate o che ne vuole di più, noi possiamo impugnare il documento dei requisiti per fargli notare cosa ha firmato, e cosa voleva che noi facessimo, niente di più, niente di meno.

E' anche possibile fornire al cliente una versione demo del Manuale Utente, e bisogna fare un Piano di Test di Sistema per verificare che i requisiti siano stati soddisfatti.

Progettazione:

Si stila un documento di specifiche di progetto in cui si definisce com'è decomposta l'architettura software: il programma è suddiviso in uno o più programmi eseguibili e per ognuno vengono definite le funzionalità svolte e le relazioni con gli altri programmi, e a loro volta i programmi sono scomposti in uno o più moduli, e per ognuno di essi sono descritte le funzionalità svolte e i legami con gli altri moduli.

Questo documento può essere scritto con opportuni linguaggi di specifica (v. uml).

Ovviamente ogni eseguibile e ogni modulo deve essere opportunamente testato per controllare che soddisfi le specifiche richieste e tutti insieme devono subire un test complessivo per assicurarsi di avere soddisfatto le specifiche.

Utilizzo e manutenzione:

Il software, completo e testato, viene consegnato al cliente.

La fase di manutenzione dura il 50-80% dei costi complessivi del ciclo di vita del programma.

Manutenzione correttiva: si aggiustano gli errori che non erano stati scovati nelle fasi precedenti.

Manutenzione perfettiva: miglioriamo le caratteristiche del programma (più veloce, più efficiente,...).

Manutenzione adattativa: vengono aggiunte funzionalità.

Controllo di qualità:

Vengono fatti test per controllare il funzionamento del programma in casi limite e nei casi standard.

Strumenti CASE:

Per costruire programmi completi partendo dai diagrammi avvalendosi dei linguaggi di quarta generazione. Ma non possono creare software: traducono il progetto in forma testuale.

Questi linguaggi, utilizzabili anche da chi non sa programmare, possono generare solo programmi semplici e non possono essere utilizzati per la manutenzione.

Fattori di qualità:

Ci sono qualità esterne che devono essere garantite, e che esulano da come il programma sia fatto dentro e qualità interne, che esaminano la struttura interna del programma, e che influenzano le qualità esterne.

Correttezza: un software è corretto se rispetta i requisiti.

Robustezza: il software non deve crashare se ci sono situazioni anomale, ma deve saperle gestire adeguatamente.

Affidabilità: se è robusto e corretto.

Estensibilità: se può essere facilmente modificato (più un sistema è grande e articolato, più diventa difficile modificarlo perché ci sono diverse dipendenze che scatenano modifiche a catena).

Riusabilità: può essere riutilizzato in parte o completamente: permette di non dover affrontare di nuovo uno stesso problema.

Compatibilità: se è standard, cioè se può essere combinato anche con altri prodotti

Facilità d'uso: se anche un utonto ce la fa a usare il software

Efficienza: se utilizza bene le risorse di calcolo

Portabilità: indipendenza dall'architettura hw o sw

Verificabilità: quanto bene può essere sottoposto ai test

Integrità: deve saper proteggere i suoi documenti da modifiche non autorizzate

Innocuità: evitare che il sistema entri in stati critici

Alcune caratteristiche sono in contrapposizione tra loro: l'importanza di una o dell'altra dipende dal settore del sistema.

Modello a cascata:

Le fasi sopracitate (studio di fattibilità, analisi, progettazione, implementazione, collaudo, manutenzione) sono distinte, in cascata tra loro, con una retroazione finale.

Presuppone che l'introduzione di cambiamenti negli stadi avanzati sia troppo costoso. Le uscite di ogni fase sono i "semilavorati" e devono essere documentati, controllati e terminati entro determinate date. Questo modello presuppone l'immutabilità dell'analisi e del progetto.

Nella realtà invece l'analisi cambia con l'avanzare del progetto e le soluzioni progettuali più performanti arrivano quando comincia ad esserci una visione concreta e globale del sistema, quindi si tende ad ammettere una retroazione al termine di ogni stato (retroazione a un livello).

Per evitare problemi, prima di lavorare sul sistema vero e proprio si crea un prototipo, solo che è un approccio molto dispendioso da annullare il vantaggio economico che questo modello dovrebbe garantire.

Il prototipo è un modello approssimato (di tipo "usa e getta") dall'applicazione da mostrare al cliente per comprendere le richieste del cliente e migliorare la

definizione dei requisiti; per questo motivo si concentra sulle parti mal comprese. Dovrebbe essere pronto in tempi brevi e con costi minimi.

Modelli evolutivi:

Partendo da specifiche molto astratte si crea un prototipo che viene raffinato successivamente fino ad arrivare al prodotto finale.

Il vantaggio di questo modello è che ad ogni interazione ci si può confrontare col cliente per rivedere le specifiche e si possono rivedere le scelte progettuali.

Il modello evolutivo si è orientato verso cicli sempre più brevi e iterazioni sempre più veloci fino a sfociare nell'extreme-programming.

Svantaggi: non è disponibile documentazione, il sistema non è ben strutturato per le modifiche frequenti ed è un lavoro per un team ristretto ed affiatato.

Per questi motivi è utilizzabile per sistemi piccoli o che avranno vita breve.

L'Extreme Programming si basa sulla comunicazione tra gli sviluppatori stessi e tra gli sviluppatori e i clienti, e su una grandissima quantità di codice di testing, sulla semplicità del codice e sul coraggio ad effettuare il refactoring del codice.

Analisi dei rischi:

Il modello viene scelto con l'obiettivo di minimizzare i rischi inerenti alle varie attività.

Il modello a cascata infatti è comodo se i requisiti sono stabili, ed è adatto a sistemi di dimensioni medio-grandi; i modelli evolutivi, poco visibili, sono adatti per sviluppare nuove applicazioni e per piccoli sistemi.

Vengono allora utilizzati modelli ibridi che adottano un modello differente per ogni sottosistema a seconda degli obiettivi.

Modello a spirale di Bohem:

Considera esplicitamente il fattore rischio.

E' un modello di tipo ciclico in cui ogni ciclo rappresenta una fase; le fasi non sono definite a priori.

Quadranti:

I: Determinazione di obiettivi, alternative, vincoli

II: Valutazione delle alternative, identificazione e risoluzione dei rischi

III: Sviluppo e verifica del successivo livello di prodotto (utilizzando il/i modello/i più adatto/i)

IV: Pianificazione della fase successiva (se c'è)

E' un modello per descrivere gli altri modelli: può essere utilizzato per descrivere modelli incrementali o modelli a cascata.