Strategia di sviluppo e ciclo di vita del software

Come scegliere la strategia di sviluppo più adatta in base alle caratteristiche del prodotto, alle modalità di rilascio, al coinvolgimento del cliente nel progetto ed ai costi del progetto software

Ercole F. Colonese

Versione 1.0 - Aprile 2006

2

Note sulla versione attuale (v1.0)

La versione attuale del documento (v1.0) è principalmente dedicata agli studenti universitari di ingegneria del software per fornire loro una breve descrizione delle caratteristiche principali dello sviluppo del software e come queste influenzino la strategia di sviluppo. L'approccio descritto è stato adoperato in un progetto reale di estrema criticità dimostrando la sua efficacia ma anche, allo stesso tempo, i propri limiti. L'applicazione dei vari modelli, anche più di uno nelle diverse fasi del ciclo di sviluppo, ha dimostrato che essi sono realmente utili quando applicati coerentemente; ha anche evidenziato quanto la cultura manageriale, in termini di gestione dei progetti, delle relazioni con i fornitori, dei gruppi di lavoro, delle relazioni con il cliente, e supporto all'applicazione dei modelli scelti con investimento in termini di formazione ed utilizzo di tool, rimane ancora da accrescere se si vuole trarre il massimo beneficio dall'applicazione delle metodologia disponibili.

Versione del documento

2006	2007	2008	2009	20010
V1.0				

Manuali di sviluppo software

Il presente manuale fa parte di una serie di documenti, in parte già disponibili ed in parte in fase di realizzazione, che mirano a fornire una vista dei vari temi dello sviluppo del software in base alle esperienze pratiche maturate in molti anni di lavoro. L'elenco dei manuali e del relativo stato di completamento è riportato qui di seguito. La sequenza dei manuali non segue il ciclo di vita del software ma piuttosto l'esigenza di rendere disponibili i documenti in particolari momenti lavorativi dell'autore (corso di formazione, lezioni presso l'università, necessità da parte di un cliente, ecc.).

1. Lo sviluppo del software oggi … (in fase di completamento)
2. **Strategia di sviluppo e ciclo di vita del software(disponibile)**
3. Project Management (disponibile)
4. Collaudo del software (Testing) (disponibile)
5. Collaudo del software per la Rete (e-Testing) (in fase di completamento)
6. Qualità del software (disponibile)
7. ISO/IEC 9126 (disponibile)
8. Gestione della qualità del software (disponibile)
9. Metriche del software (in fase di completamento)
10. Usabilità del software (disponibile)
11. Usabilità dei siti Web (disponibile)
12. Metodi e tecniche dello sviluppo software (in fase di preparazione)
13. Introduzione al CMMI (in fase di completamento)
14. SWEBOK (disponibile)
15. Glossario dei termini dello sviluppo software (disponibile)
16. Bibliografia dell'ingegneria del software (disponibile)

I manuali completati sono disponibili nel sito dell'autore (http://www.colonese.it/pubblicazioni/htm).

Sommario

Indice degli argomenti

Indice delle figure

Indice delle tabelle

1 Strategia di sviluppo e modelli di ciclo di vita del software

1.1 Introduzione

I vari modelli di ciclo di vita[1] che l'ingegneria del software ha proposto negli anni sono una risposta alle diverse esigenze che i progetti hanno evidenziato a seguito di problemi incontrati. I modelli sono quindi soluzioni di problemi specifici e non il superamento di modelli precedenti perché antiquati. Non è vera quindi la convinzione di molti che, per esempio, il modello "a cascata" (waterfall) sia superato perché "vecchio". Esso rimane ancora un modello validissimo nei progetti in cui i requisiti siano ben definiti e stabili nel tempo ed il prodotto finale si possa fornire tutto insieme in un unico rilascio. Il problema, semmai, è che oggi ben pochi progetti hanno tali caratteristiche. Il modello "iterativo-incrementale", viceversa, è sicuramente più efficace nei casi, oggi sempre più frequenti, in cui i requisiti siano poco chiari e stabili all'inizio, e lo diventino solo nel tempo man mano che il progetto si affina.

Non esiste quindi un modello ideale, valido per tutti i progetti. E' necessario scegliere quello più appropriato al progetto. Si potrebbe anche essere costretti a cambiare modello quando le ipotesi iniziali dovessero mutare tanto da invalidare la scelta iniziale. Inoltre, si possono adottare anche modelli diversi nelle varie fasi del ciclo di del progetto. In un progetto scomponibile in parti separate da rilasciare in tempi successivi, ad esempio, si può utilizzare l'approccio prototipale nella fase di analisi per la definizione e la validazione dei requisiti, quando questi non siano ben chiari; si può utilizzare il modello iterativo per la progettazione e la realizzazione dei

[1] Relativamente ad un progetto software, è più corretto parlare di "ciclo di sviluppo" piuttosto che di "ciclo di vita". La prima espressione, infatti, si riferisce solo alla "realizzazione" del prodotto secondo un processo definito ("ciclo di sviluppo"). Il secondo termine, invece, indica l'intera "vita" di un prodotto e, in analogia, le principali fasi sono: il suo "concepimento" (l'idea iniziale di business), la sua "nascita" (la realizzazione secondo il "ciclo di sviluppo" adoperato), il rilascio (il lancio sul mercato o la sua messa in esercizio), la sua "crescita" (la manutenzione e l'evoluzione con rilasci successivi), la sua "morte" (il ritiro dal mercato). Qui si utilizza impropriamente il termine "ciclo di vita" in conformità all'uso comune che generalmente si fa del termine, anche se improprio.

singoli rilasci; si può utilizzare il modello di sviluppo rapido (RAD) per la progettazione e la realizzazione di quei componenti di piccole dimensioni e le cui caratteristiche funzionali e tecniche siano perfettamente chiare al team di sviluppo; si può infine utilizzare parte del ciclo a cascata per un singolo componente i cui requisiti siano chiari, stabili e concordati tra le parti.

Ma come si sceglie il modello di ciclo di vita più adatto al progetto? A guidarci è la strategia di sviluppo che scegliamo di adottare per il progetto. Vediamo prima, quindi, come si decide la strategia.

La scelta del modello di ciclo di vita è importante perché da esso dipende il processo di sviluppo da adoperare e, da questo, il progetto stesso, come mostra la figura che segue.

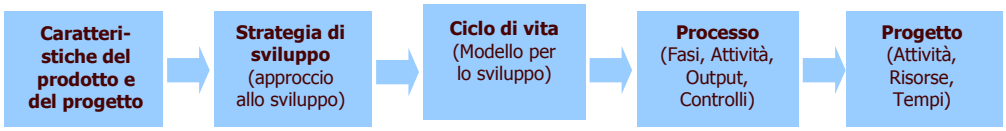

Figura 1. Relazione tra strategia, ciclo di vita, processo e progetto.

Riassumiamo la sequenza delle operazioni ai fini della scelta finale:

- Analisi delle "caratteristiche del prodotto e del progetto" (requisiti, rilasci, coinvolgimento degli utenti, costi di sviluppo, visibilità del cliente).
- Individuazione della "strategia di sviluppo" più adatta (Ciclo singolo, Cicli multipli, Prototipale, Evolutivo, Incrementale, A fasi).
- Identificazione del "ciclo di vita" che meglio interpreta la strategia di sviluppo individuata (a cascata, a fasi, prototipale, a spirale, a stadi);
- Definizione del "processo di sviluppo" con identificazione delle fasi da seguire, le attività da svolgere, gli artefatti da produrre, i controlli da eseguire, le evidenze da produrre, le misure da effettuare, le tecniche ed i metodi da adoperare, gli strumenti da utilizzare.
- Definizione del "progetto" con relative stime dei tempi e dei costi, attività da svolgere e loro sequenza e durata, risorse da coinvolgere in numero e durata, livelli di qualità da raggiungere ed controlli da effettuare, controllo del progetto in termini di attività svolte, prodotti realizzati, costi maturati e qualità raggiunta.

1.2 Alcune esperienze pratiche

La metodologia proposta è stata sperimentata presso un gruppo di piccole e medie aziende informatiche italiane. I progetti su cui si è definita la strategia di sviluppo e si è adottato un particolare modello di ciclo di vita presentavano requisiti mediamente chiari e stabili ed i relativi tempi di consegna erano compatibili con la complessità del progetto e con la produttività media garantita dal gruppo di sviluppo. La sperimentazione ha quindi dimostrato l'efficacia dell'approccio metodologico descritto ma non ha evidenziato i punti più critici trattati dal modello.

> *Le aziende che hanno partecipato alla sperimentazione della metodologia hanno costruito una tabella in cui, per ciascuna tipologia di progetto, sono elencate le attività da svolgere, gli output da produrre ed i controlli e le verifiche da effettuare. Solo caratteristiche molto particolari di progetti, o esigenze esplicite del cliente, autorizzano a definire un processo diverso. E' comunque richiesta la valutazione del processo da parte dell'Assicurazione qualità e l'autorizzazione da parte della direzione. La tabella riportata di seguito mostra gli output da produrre ed i criteri di uscita per ciascuna fase. Nell'esempio è riportato il processo completo relativo a progetti complessi e/o di grandi dimensioni. Analoghe tabelle definiscono cosa produrre in progetti meno complessi e/o di dimensioni minori.*

La strategia di sviluppo deve permettere di selezionare più modelli di sviluppo nelle diverse fasi del progetto:

- Per esempio, si può adottare un approccio "prototipale" durante la fase di analisi per la validazione dei requisiti.
- L'approccio "iterativo-incrementale" può essere invece adottato per realizzare due componenti principali da rilasciare in due momenti successivi; nel primo rilascio, per esempio, si realizzano le funzionalità principali del prodotto, nel secondo rilascio si sviluppano nuove funzionalità e si arricchiscono quelle rilasciate precedentemente.
- L'approccio "prototipale" può ancora essere seguito per la progettazione e la realizzazione di un componente particolarmente critico e poco chiaro, nella sua struttura ed utilizzo, sia al cliente che al gruppo di sviluppo;
- L'approccio "iterativo-incrementale" può ancora essere seguito nella fase di integrazione e collaudo dei vari componenti rilasciati i tempi successivi.

1.3 Riferimenti

I modelli presi a riferimento per lo sviluppo del presente volume sono gli standard:

- ISO/IEC 12207:2003 "Tecnologia dell'informazione – Processi del ciclo di vita del software";
- ISO/IEC 9126-1:2001 "Software Engineering – Product Quality – Part 1: Quality Model".

La norma ISO/IEC 12207:2003 definisce un insieme di processi, attività e compiti progettati per essere personalizzati rispetto ai diversi progetti software, permettendo l'eliminazione di processi, attività e compiti non applicabili. La norma, inoltre, non prescrive l'adozione di una metodologia particolare per lo sviluppo del software, né impone uno specifico ciclo di vita (a cascata, a spirale, incrementale o iterativo), tutti elementi che possono variare in funzione dei processi produttivi e delle metodologie in uso presso ciascun fornitore.

Lo standard prescrive invece che siano svolti in modo controllato e formalizzato alcuni processi base, qualunque sia il ciclo di vita adattato. I processi di base sono classificati in tre tipologie diverse:

- **Processi primari**. Si tratta dei processi, detti di "core business", utilizzati per la realizzazione dei prodotti richiesti dal cliente ed orientati all'utente finale. I prodotti sono sviluppati all'interno di progetti.
- **Processi di supporto**. Si tratta dei processi di supporto, ad uso dell'organizzazione che realizza il progetto, per assicurare il successo del progetto stesso e la qualità attesa del prodotto.
- **Processi organizzativi**. Si tratta dei processi di tipo manageriale, seguiti generalmente fuori dall'ambito del progetto, e riguardanti la pianificazione ed il controllo del progetto, nonché la gestione delle risorse umane e materiali necessarie per la conduzione delle attività.

In particolare, seguendo le indicazioni della norma, i processi del ciclo di vita di una fornitura sono scomposti in attività e, per ciascuna attività, sono identificati i relativi prodotti da realizzare (documenti, codice, procedure, manuali, formazione).

Solo a titolo informativo, si riporta nella figura di seguito lo schema dei processi definiti dalla norma.

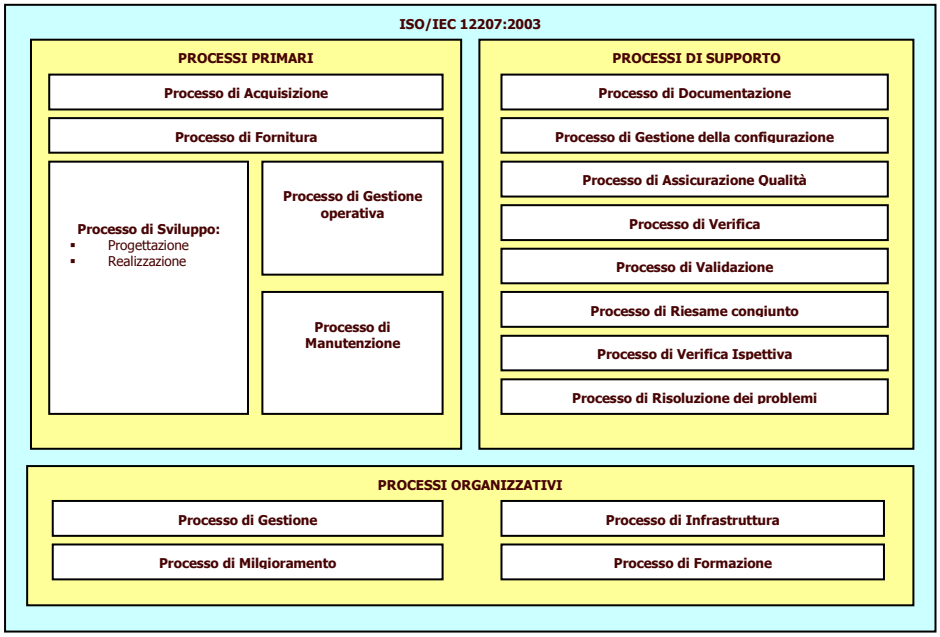

Figura 2. Processi primari, di supporto ed organizzativi della norma ISO/IEC 12207:2003.

La norma ISO/IEC 9126-1:2001 definisce invece il modello per la qualità del software cui si fa riferimento nella definizione delle caratteristiche e sotto-caratteristiche del software (funzionalità, usabilità, efficienza, portabilità, ecc.). Tali elementi possono essere adottati come "requisiti di qualità" del software da realizzare. La norma individua tre classi differenti di qualità: qualità "interna", qualità "esterna" e qualità "in uso".

- La qualità interna si riferisce alle caratteristiche interne del software che si possono rilevare analizzando il codice, la documentazione tecnica e di progetto e la documentazione per l'utente.
- La qualità esterna si riferisce alle caratteristiche esterne del prodotto e sono rilevabili in fase di esecuzione del software negli ambienti in cui sarà utilizzato (ambiente di esercizio).
- La qualità in uso si riferisce alle caratteristiche del prodotto rilevabili in uno specifico contesto (contesto lavorativo dell'utente) e misura quanto il software realizzato supporta l'utente nello svolgere il proprio lavoro raggiungendo i propri obiettivi lavorativi.

Le due tabelle riportate di seguito riassumono le caratteristiche e sotto-caratteristiche di qualità interna, esterna ed in uso definite dalla norma in oggetto per i prodotti software.

Figura 3. Caratteristiche della qualità interna ed esterna del software secondo la norma ISO/IEC 9126.

QUALITA' INTERNA ED ESTERNA DEL SOFTWARE					
FUNZIONALITA'	**AFFIDABILITA'**	**USABILITA'**	**EFFICIENZA**	**MANUTENIBILITA'**	**PORTABILITA'**
Adeguatezza	Maturità	Comprensibilità	Prestazioni temporali	Analizzabilità	Adattabilità
Accuratezza	Tolleranza ai guati	Apprendibilità	Utilizzo delle risorse	Modificabilità	Installabilità
Interoperabilità	Ripristinabilità	Operabilità	Conformità all'efficienza	Stabilità	Coesistenza
Sicurezza	Conformità all'affidabilità	Attrattività		Collaudabilità	Sostituibilità
Conformità alla funzionalità		Conformità all'usabilità		Conformità alla manutenibilità	Conformità alla portabilità

Figura 4. Caratteristiche della qualità in uso del software secondo la norma ISO/IEC 9126.

QUALITA' IN USO DEL SOFTWARE			
Efficacia	Produttività	Salvaguardia	Soddisfazione

Per quanto concerne il processo di sviluppo si fa riferimento a quello detto "unificato" e che nell'operatività pratica è formalizzato dalla Rational come Rational Unified Process (RUP). Esso è descritto nel dettaglio nel Capitolo 5 "Il Processo di sviluppo".

2 Elementi della strategia di sviluppo

La strategia di sviluppo del software determina le modalità con cui si intende condurre il progetto e realizzare il prodotto finale. Gli elementi da considerare per definire la strategia di sviluppo più adeguata sono: le "caratteristiche esterne del prodotto", le "modalità di rilascio del prodotto", il "coinvolgimento del cliente nel progetto" ed i "costi del progetto". Vediamo nel dettaglio ciascun elemento.

2.1 Caratteristiche del prodotto

Occorre prendere in considerazione se le caratteristiche del prodotto finale risultino chiare e consolidate piuttosto che vaghe e instabili. Esse definiscono come il prodotto finale dovrà apparire ai suoi utilizzatori ed include, tra l'altro, le funzionalità da fornire agli utenti, le relative modalità operative per usufruirne, gli eventuali limiti e vincoli da rispettare - tecnici, normativi, legislativi, operativi, ecc. -, gli eventuali standard cui aderire, le interconnessioni da realizzare, ecc. Tra le caratteristiche del prodotto riveste particolare importanza il livello di "prestazioni" e di "affidabilità" richiesto al prodotto: in caso di software critico per la vita umana, per esempio, occorre prendere in considerazione particolari caratteristiche che incidono sul progetto, sul ciclo di vita ed in maniera determinate sui costi; analogamente, in caso di software particolarmente performante occorrerà intervenire sulla progettazione con tecnologie, risorse e tecniche particolari influendo sui costi complessivi del progetto.

2.2 Modalità di rilascio del prodotto

Il numero dei rilasci previsti, i contenuti e le modalità di rilascio, determinano la strategia di sviluppo. La strategia di sviluppo per uno prodotto da rendere disponibile con un unico rilascio, per esempio, sarà diversa da quella in cui le funzionalità saranno rese disponibili in più rilasci successivi; e sarà ancora diversa nel caso di più rilasci successivi in cui in ciascuno potranno essere migliorate le funzionalità precedenti, e così via. Le

diverse strategie di sviluppo condizioneranno i diversi modelli di ciclo di vita.

2.3 Coinvolgimento del cliente nel progetto

Il grado di coinvolgimento del cliente (committente, utenti finali) e degli altri interessati al progetto determina le modalità di gestione delle attività del progetto stesso: revisioni ed approvazioni, partecipazione alle scelte, coinvolgimento in attività tecniche specifiche (riunioni congiunte per l'analisi dei requisiti, validazione di prototipi, esecuzione di test d'accettazione, ecc.), discussione periodica dello stato del progetto, ecc. Il coinvolgimento del cliente aiuta a migliorare l'interpretazione delle esigenze e permette di validare le soluzioni, ma incide sui costi del progetto. Viceversa, un coinvolgimento scarso del cliente riduce i costi di progetto ma richiede che i requisiti siano molto chiari e stabili.

2.4 Costi di sviluppo

Il budget iniziale ed il suo livello di flessibilità incidono sulla strategia di sviluppo. Occorre infatti capire se esso i costi del progetto siano stati calcolati sulle caratteristiche di massima del prodotto piuttosto che su requisiti chiari e stabili; inoltre, occorre considerare se siano previste variazioni economiche a fronte di possibili modifiche richieste dal grado di raffinamento dei requisiti in corso d'opera. Il budget è "fisso e blindato"? La risposta può sembrare ovvia ("i costi di sviluppo devono essere rispettati") ed invece non lo è nell'esperienza quotidiana. Per "flessibilità dei costi" s'intende la possibilità di negoziare eventuali variazioni del budget fissato a fronte di variazioni significative nelle caratteristiche iniziali del progetto (requisiti, prestazioni ed affidabilità, vincoli e limiti, standard, piano dei rilasci, ecc.). La rigidità dei costi condiziona la strategia di sviluppo. Nel caso in cui la flessibilità sia negata per vari motivi, per esempio, occorrerà adottare una tecnica particolare, detta di "time or content boxing": ciascun componente "nuovo" richiesto potrà essere sviluppato solo se sostituisce un altro di pari costo e tempo di realizzazione!

3 Tipi di strategia di sviluppo

Vediamo ora quali strategie di sviluppo sono disponibili. L'ingegneria del software propone sei diversi tipi di strategie per indirizzare le diverse necessità[2]:

- Sviluppo in un solo ciclo ("One Shot");
- Sviluppo in più cicli ("Multi Shot");
- Sviluppo evolutivo ("Evolutionary");
- Sviluppo in fasi ("Phased");
- Sviluppo prototipale ("Prototyping");
- Sviluppo incrementale ("Incremental").

[2] L'approccio alla cosi detta "programmazione estrema" (*Extreme Programming*) appartenente alle nuove tecniche dette "agili" (*Agile Methodology*); essa non è qui presa in considerazione perché, nonostante sia conosciuta già da tempo e abbia dimostrato la sua efficacia nei progetti in cui è stata adottata, richiede particolari competenze ed attitudini, tecniche ed organizzative, individuali e collettive, che, se non possedute e dominate con sufficiente perizia potrebbero compromettere l'esito del progetto. Si suggerisce quindi di approfondire tali tecniche con studi particolari e di adottarle in progetti pilota prima della loro diffusione nell'organizzazione.

3.1 Sviluppo in un solo ciclo

Con questa strategia ("One shot") l'intero prodotto software è sviluppato nell'unico ciclo previsto. Tutte le funzionalità previste sono messe a disposizione degli utenti nell'unico rilascio previsto. E' richiesto che tutti i requisiti siano ben chiari, completi e stabili fin dall'inizio del progetto e rimangano tali per l'intera durata dello sviluppo. Non sono numerosi i casi cui si applichi, ma non sono nemmeno rari. Dopo le fasi di analisi e progettazione, la codifica ed i test unitari possono essere svolti in parallelo. Segue la fase di integrazione e test dell'intero prodotto. Il rilascio è unico. Gli utenti partecipano generalmente solo alla fase di analisi.

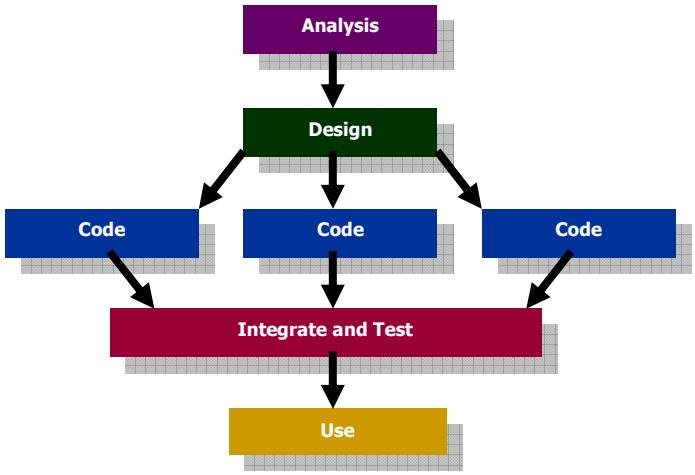

Figura 5. Strategia di sviluppo ad un ciclo ("One Shot").

Questa strategia di sviluppo ha costi inferiori rispetto ad altre ed è applicabile nel caso di requisiti ben definiti e stabili, e gli sviluppatori abbiano una buona conoscenza dei problemi da affrontare (contesto applicativo, organizzativo e tecnologico).

3.2 Sviluppo in più cicli

La strategia ("Multi shot") prevede la successione di più cicli di sviluppo indipendenti. Ciascun ciclo sviluppa un sottoinsieme di caratteristiche del prodotto finale e può modificare quanto sviluppato nei cicli precedenti.

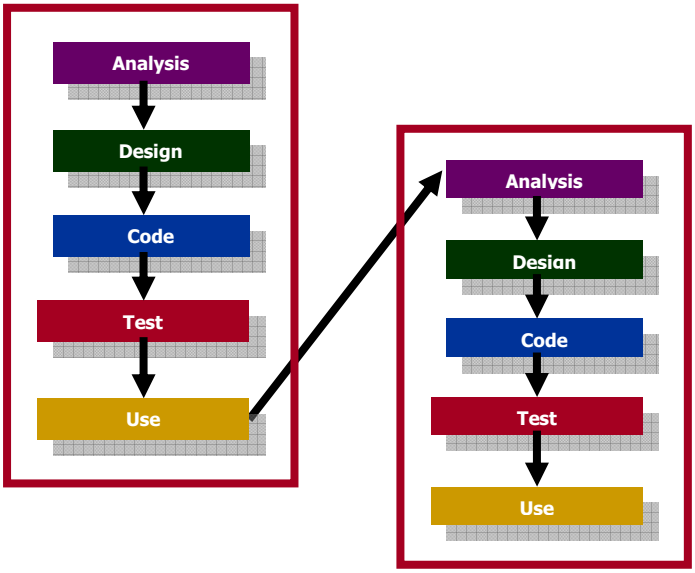

Figura 6. Strategia di sviluppo a più cicli ("Multi Shot").

Si adatta bene ai casi in cui solo alcuni requisiti iniziali siano chiari, mentre altri risultino inizialmente vaghi ed instabili. La strategia si applicata bene allo sviluppo di applicazioni interne all'azienda ("in house"), oppure per progetti di "ricerca".

3.3 Sviluppo evolutivo

Con questa strategia ("Evolutionary"), le funzionalità del prodotto sono realizzate in rilasci diversi e successivi. Eventuali variazioni necessarie a seguito dell'utilizzo del prodotto sono indirizzati nei rilasci successivi. L'approccio si applica ai casi in cui ci si aspetta che i requisiti possano variare in maniera consistente nel tempo. Il meccanismo da utilizzare per la gestione delle modifiche è cruciale in questo approccio e deve essere concordato all'inizio del progetto, prevedendo le modalità per il monitoraggio e la documentazione delle modifiche stesse.

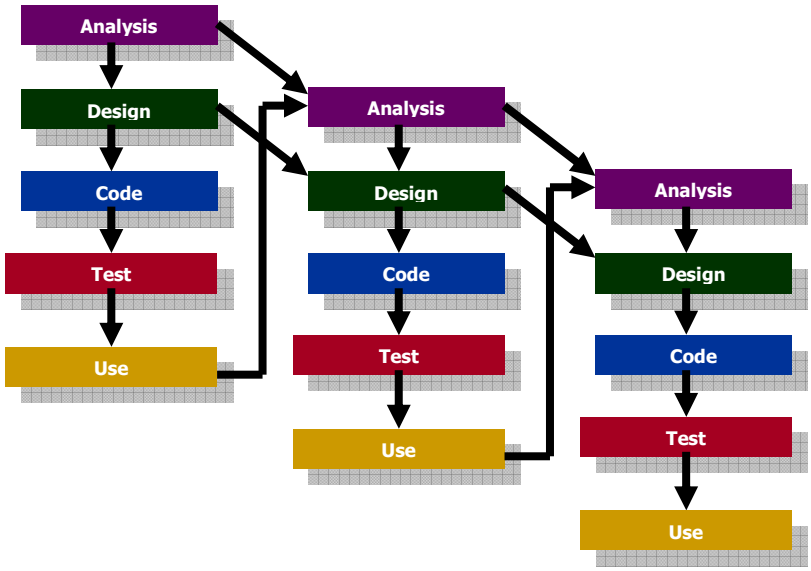

Figura 7. Strategia di sviluppo evolutiva (Evolutionary).

La strategia è adatta nei casi in cui il prodotto debba essere aggiornato nel tempo. Si parte con alcune funzionalità di base e consolidate per poi far evolvere il prodotto indirizzando nuove esigenze emerse con il tempo, l'utilizzo e l'evoluzione del mercato. Il tipico esempio è quello di un prodotto che nasce per porre la propria presenza sul mercato con un numero di funzioni minime di base e poi si evolve con rilasci successivi per indirizzare ulteriori necessità emerse dal suo utilizzo, dall'evoluzione del mercato e dai concorrenti.

3.4 Sviluppo per fasi

Quando le dimensioni di un prodotto sono troppo grandi perché questo possa essere sviluppato tutto in un solo ciclo occorre prevedere più fasi di sviluppo parallele ("Phased"), ciascuna delle quali realizzi una parte significativa del prodotto (componente). Ciascun componente è realizzato in maniera indipendente e costituisce una sorta di "rilascio interno". La fase iniziale di analisi e progettazione del prodotto decompone l'applicativo in sottocomponenti indipendenti e definisce in maniera chiara ed esaustiva le loro interrelazioni. Quindi si eseguono più fasi parallele di sviluppo dei singoli componenti secondo la priorità stabilita per ciascuna fase. La fase conclusiva di integrazione e collaudo di tutti i componenti realizzati permette il rilascio dell'intero prodotto.

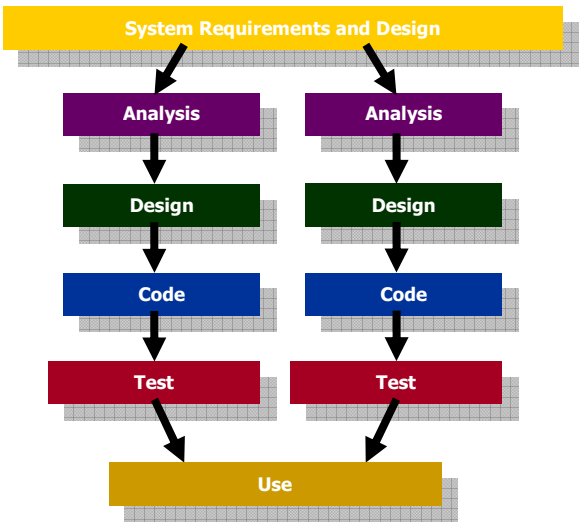

Figura 8. Strategia di sviluppo a fasi ("Phased").

Questa strategia è particolarmente adatta per i progetti che abbiano un budget limitato e tempi di realizzazione brevi.

3.5 Sviluppo prototipale

Tipicamente, un prototipo è realizzato ("Prototyping") per capire meglio cosa fare in un tempo relativamente breve rispetto ai tempi previsti dal progetto. Quando i tempi siano particolarmente brevi, il prototipo è realizzato solo a titolo dimostrativo e non è mai utilizzato come codice nel prodotto finale (prototipo "usa e getta").

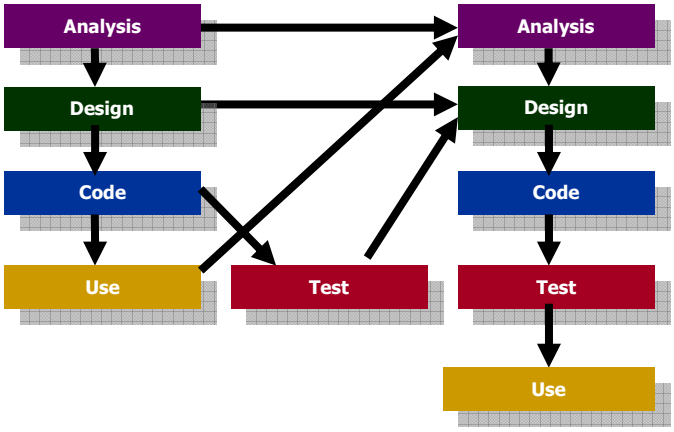

Figura 9. Strategia di sviluppo prototipale ("Prototyping").

La strategia consente di ridurre i rischi, rendere più efficace il coinvolgimento degli utenti, ridurre tempi e costi di realizzazione. E' particolarmente adatto a progetti con requisiti vaghi, a progettazioni innovative o alla sperimentazione di nuove tecnologie, a prodotti con caratteristiche particolarmente critiche (performance, sicurezza, usabilità, ecc.).

3.6 Sviluppo incrementale

La strategia incrementale ("Incremental") divide il prodotto in più parti incrementali autoconsistenti e di dimensioni contenute. Ciascuna parte, per esempio, può essere considerata come un rilascio indipendente in cui siano realizzate alcune funzionalità specifiche ed utilizzabili in maniera consistente dagli utenti. L'ordine dell'implementazione delle diverse parti (rilasci) è determinata all'inizio del progetto. Ciascuna fase del progetto si può considerare come un singolo sviluppo fatto secondo la strategia "One Shot". Ovviamente gli utenti devono concordare sull'insieme delle funzionalità rilasciate in ciascuna fase, cioè sui deliverable previsti in ciascun rilascio.

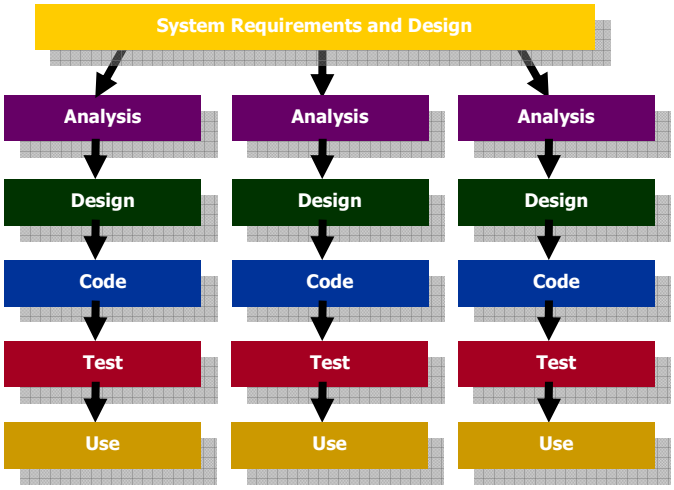

Figura 10. Strategia di sviluppo incrementale ("Incremental").

3.7 Definizione della strategia di sviluppo

Dal punto di vista pratico, la definizione della strategia di sviluppo più adatta al progetto si concretizza nella produzione di una tabella costruita con più passi successivi:

1. Valutare per primo il progetto in base alle sue caratteristiche salienti evidenziando quali siano gli elementi particolarmente importanti. L'esempio riportato nella tabella che segue mostra le caratteristiche ritenute importanti o essenziali (√) ai fini della scelta (es.: Requisiti ben definiti, Alta affidabilità, Costi contenuti) e quelle non richieste (✖) (es.: Rilascio anticipato, Coinvolgimento utenti, Visibilità da parte del cliente).
2. Valutare quindi ciascuna strategia di sviluppo secondo gli stessi criteri di valutazione definiti e riportare i risultati in righe separate.
3. Valutare infine quale strategia di sviluppo sia più adatta in base al numero di caratteristiche del progetto indirizzate da ciascuna strategia. Nell'esempio, la stragia con un unico ciclo ("One Shot") risulta essere quella più adatta al progetto.

Nota: Naturalmente, tutte le caratteristiche del progetto devono essere indirizzate dalla strategia selezionata. Non indirizzare una o più caratteristiche del progetto rappresenta un problema molto serio e aumenta il rischio di fallimento del progetto. Qualora una caratteristica importante non sia indirizzata dalla strategia ritenuta la più adatta, occorre modificare la strategia stessa in modo da indirizzare anche la scopertura evidenziata. Viceversa, se strategie diverse coprono tutte le necessità del progetto, si opterà per quella che indirizza meglio alcuni elementi ritenuti importanti come, ad esempio, il costo di sviluppo, piuttosto che un'altra caratteristica.

Figura 11. Criteri per la selezione del modello di ciclo di sviluppo.

Strategia	Caratteristiche del progetto					
	Requisiti ben definiti	**Alta affidabilità**	**Rilascio anticipato**	**Costi di sviluppo bassi**	**Coinvolgimento degli utenti**	**Visibilità del cliente**
Progetto in esame	√	√	✖	√	✖	✖
Ciclo singolo	√	√	✖	√	✖	✖
Ciclo multiplo	✖	✖	√	✖	✖	√
Prototipale	✖	✖	√	√	√	✖
Evolutivo	✖	√	√	✖	√	√
Incrementale	✖	√	√	✖	√	√
A fasi	√	√	√	✖	√	√

4 Modello di Ciclo di vita

Una volta stabilità la strategia più adatta alle caratteristiche del progetto, si passa a definire il modello di ciclo di vita che meglio traduce tali esigenze.

Un "ciclo di vita" (o "ciclo di sviluppo") rappresenta un modello prescrittivo che aiuta ad organizzare le attività del progetto. Esso stabilisce l'approccio interno che s'intende adottare e descrive il processo che si dovrà seguire. Il ciclo di vita descrive anche i contenuti del progetto in termini di attività da svolgere ed i relativi artefatti (deliverable). Permette di scomporre il progetto in attività singole (WBS - Work Breakdown Structure) da utilizzare come base per la pianificazione. L'ingegneria del software propone cinque modelli differenti di ciclo di vita; di questi esistono molte varianti. Qui di seguito sono illustrati solo brevemente quelli principali:

- Modello a cascata ("Waterfall");
- Modello a stadi ("Staged");
- Modello prototipale ("Prototyping");
- Modello iterativo ("Iterative");
- Modello a spirale ("Spiral").

Si rimanda invece alla letteratura specialistica il loro approfondimento e la descrizione di ulteriori modelli. In particolare, qui si fa riferimento anche al modello sviluppato da un'azienda leader che incorpora concetti iterativi ad un approccio incrementale:

- Modello iterativo-incrementale ("Iterative-Incremental").

L'applicazione del ciclo di vita in un progetto software è fatta seguendo il processo di sviluppo definito. Questo rappresenta la formalizzazione delle fasi da seguire, delle attività da svolgere, degli input richiesti e degli output da produrre, dei ruoli coinvolti, dei controlli e verifiche da eseguire, delle metriche da adoperare, dei metodi e delle tecniche da utilizzare, degli strumenti da adoperare e delle evidenze da produrre. Nel capitolo che segue è descritto il processo più noto, il Rational Unified Process (RUP).

4.1 Modello a cascata

Il modello "a cascata" ("Waterfall") è molto popolare per la sua semplicità e flessibilità; è anche il primo modello definito ed utilizzato nella maggior parte dei primi progetti di software. Il modello prevede molte fasi, tutte sequenziali, con sovrapposizione anticipata del loro inizio per ridurre il tempo totale del ciclo. La "Pianificazione" (Plan) del progetto è la prima fase, consiste nell'analisi del problema da risolvere e produce formalmente il Piano di progetto. La fase di "Analisi" (Analysis) elabora i requisiti, specifica nel dettaglio le esigenze e descrive le funzionalità del sistema. Nella fase di "Progettazione" (Design) gli sviluppatori definiscono come il prodotto indirizzerà tecnicamente i requisiti. La fase di "Codifica e test unitario" (Code) produce il codice sorgente e verifica la correttezza di ogni modulo singolarmente. Nella fase di "Collaudo" (Test) si effettua l'integrazione dei moduli nei sottosistemi previsti ed il prodotto finale è verificato nella sua completezza e correttezza. Segue la fase di "Rilascio" (Install) del prodotto in esercizio. L'utilizzo del prodotto da parte degli utenti è rappresentata dalla fase di "Esercizio" (Use), che prevede la correzione dei difetti scoperti e l'evoluzione del prodotto stesso con nuove funzionalità. Questa ultima fase è detta generalmente di "Manutenzione".

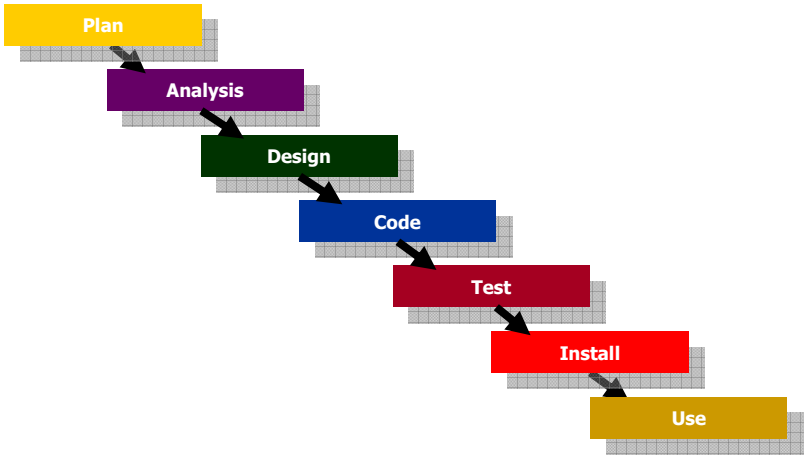

Figura 12. Modello di ciclo di vita a cascata ("Waterrfall").

Il modello è particolarmente efficace nei progetti in cui il prodotto finale debba essere consegnato in un singolo rilascio ed i requisiti siano ben chiari e stabili fin dall'inizio del progetto e questo non duri troppo a lungo per cui i requisiti potrebbero cambiare con ogni probabilità.

4.2 Modello a stadi

In questo modello ("Staged"), la fase di "Analisi dei requisiti" e quella di "Progettazione preliminare" sono condotte inizialmente per l'intero prodotto. L'applicativo è quindi scomposto in componenti per il loro sviluppo. L'identificazione dei componenti permette di assegnare le priorità del loro sviluppo. Quelli più importanti, o che sono prerequisito allo sviluppo di altri, sono realizzati per primi (primo stadio); gli altri sono sviluppati successivamente (secondo stadio, terzo stadio, ecc.).

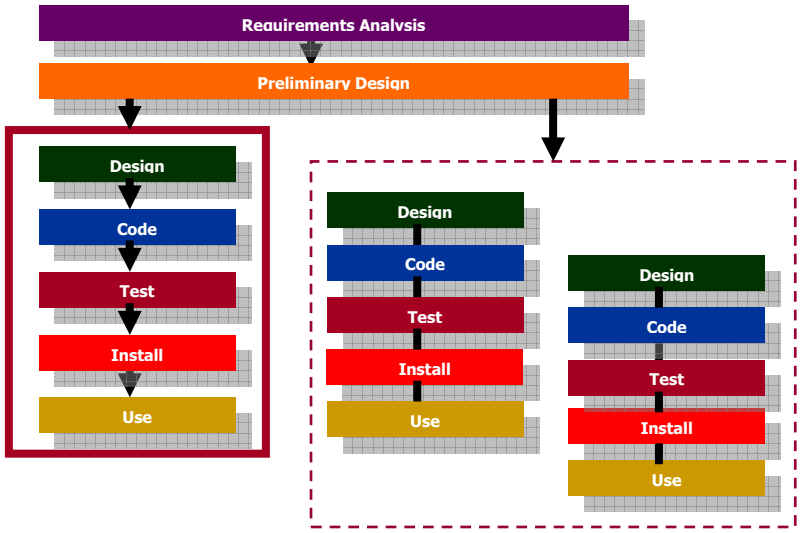

Figura 13. Modello di ciclo di vita a stadi ("Staged").

Sono possibili anticipazioni degli stadi in caso di scarsa dipendenza tra di loro.

Ciascuno stadio realizza uno o più componenti, cioè una parte di prodotto finale, in grado di essere utilizzato dagli utenti.

Il modello è particolarmente adatto in progetti in cui il prodotto finale possa essere realizzato con rilasci successivi, ma i requisiti e la struttura dell'intero prodotto possano essere definiti sin dall'inizio del progetto.

4.3 Modello prototipale

Il modello ("Prototyping") prevede due cicli di sviluppo distinti: uno per la realizzazione del prototipo ed uno per lo sviluppo del sistema; i due cicli si scambiano informazioni preziose ai fini dello sviluppo del prodotto finale. All'inizio del progetto si crea un prototipo di alcune parti del prodotto (interfaccia utente, componente critico, ecc.). In questa fase, il progetto si focalizza sulla pianificazione, progettazione, realizzazione, utilizzo e valutazione del prototipo. Tali attività sono evidentemente cicliche e si prosegue finché non si ottengono le informazioni necessarie. Appena il ciclo di prototitpazione termina, il progetto passa al ciclo (fase) realizzativo del prodotto finale.

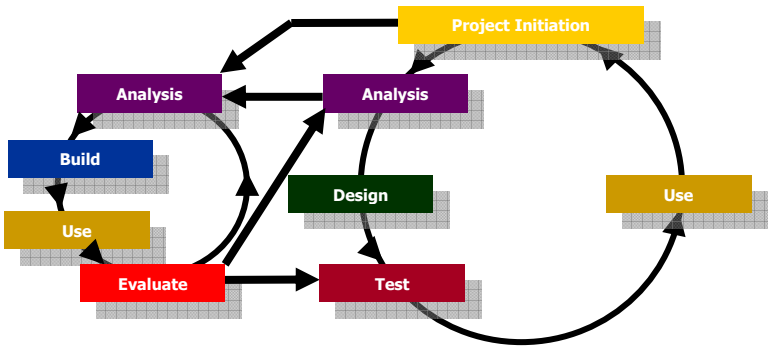

Figura 14. Modello di ciclo di vita prototipale ("Prototyping").

Il modello può essere utilizzato con successo per investigare diverse parti critiche del prodotto finale.

4.4 Modello iterativo

Il modello iterativo ("Iterative") rappresenta uno scambio continuo di "riscontri" (feedback) tra una fase di sviluppo e le attività svolte o in corso. Ciascuna fase, in base all'esperienza maturata, fornisce elementi di riscontro alle attività svolte nelle fasi successive e viceversa. Tale approccio è particolarmente adatto alla progettazione di applicazioni basate su tecnologie ad oggetti (object-oriented). Il numero delle iterazioni ed il livello di completamento delle fasi non è definito a priori dal modello; essi dipendono dal livello di approfondimento richiesto dal progetto. Il rischio è quindi quello di non finire mai il ciclo iterativo, specialmente in presenza di progettisti "innamorati della propria soluzione tecnica", come suol dirsi, piuttosto che del raggiungimento degli obiettivi del progetto. Ciononostante il modello mantiene tutta la sua validità come strumento di approfondimento continuo della progettazione.

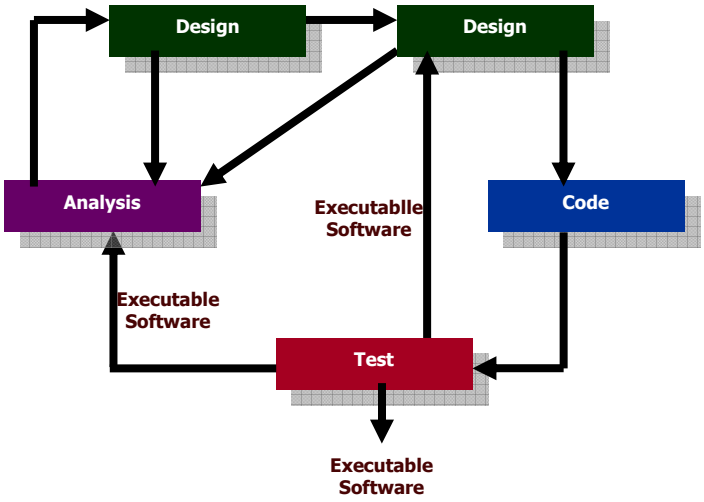

Figura 15. Modello di ciclo di vita iterativo ("Iterative").

Il modello è fortemente raccomandato per progetti di grandi dimensioni, complessi e con modifiche ai requisiti.

4.5 Modello a spirale

Il modello ("Spiral") vede il ciclo di vita come lo sviluppo di una serie di prototipi, ciascuno più evoluto e completo del precedente, fino a giungere al prodotto finale. Alla fine di ciascun ciclo di sviluppo, il prototipo realizzato è valutato e fornisce le informazioni necessarie ad iniziare il ciclo successivo, in cui sarà realizzato il prototipo più evoluto e completo.

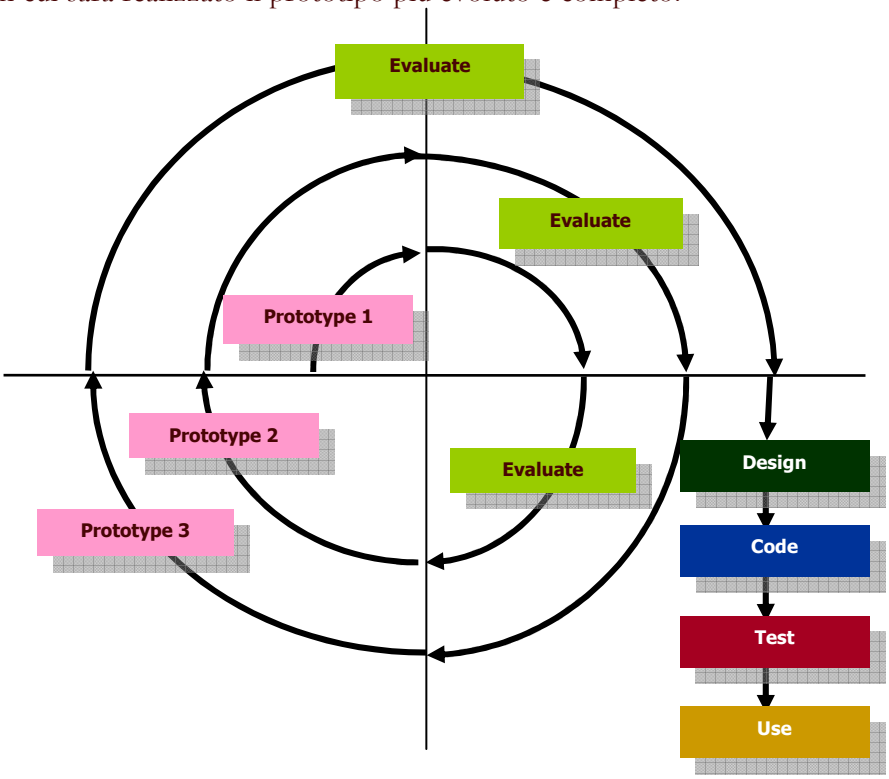

Figura 16. Modello di ciclo di vita a spirale ("Spiral").

Il modello è particolarmente utile nei progetti in cui i requisiti risutino vaghi e poco stabili, ed in quelli in cui si adottino nuove tecnologie ed ancora poco conosciute.

4.6 Modello iterativo-incrementale

Il modello ("Iterative-Incremental") è un'estensione del ciclo iterativo visto in precedenza. Esso vede il ciclo di vita come lo sviluppo di una serie di singoli "cicli completi di sviluppo", detti "iterazioni", ciascuno in grado di sviluppare un insieme di funzionalità complete da rilasciare in esercizio. Ciascuna iterazione "incrementa" le funzionalità del prodotto rese disponibili con i rilasci precedenti (da qui il nome "iterativo-incrementale"). Si parte da una fase di analisi completa (Analysis), seguita da una fase di progettazione dell'intero applicativo (Macro Design). Si pianificano le iterazioni con definizione dei contenuti e delle priorità. Ciascuna iterazione è un ciclo completo di sviluppo con una fase di progettazione di dettaglio (Detailed Design), una di codifica e test unitario (Code and Unit test) ed una di integrazione e collaudo (Integration and Test). Se necessario, l'iterazione può riciclare più volte tra progettazione di dettaglio, codifica e test per risolvere i problemi incontrati di volta in volta. L'integrazione include le funzionalità sviluppate nell'iterazione corrente con il resto del prodotto già sviluppato e messo in esercizio. Il collaudo è completo ed include test funzionali e di prestazioni del sistema. Le nuove funzionalità sviluppate nell'iterazione corrente sono rilasciate in esercizio (Release (Use)) in una nuova versione del prodotto. Si ritorna quindi all'inizio del ciclo di vita per una nuova iterazione.

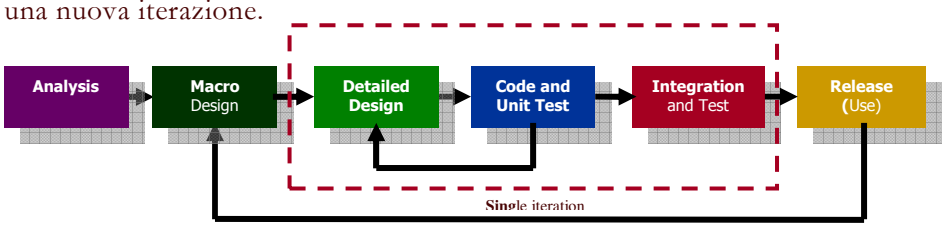

Figura 17. Modello di ciclo di vita iterativo-incrementale ("Iterative-Incremental").

Il modello permette di rendere disponibili di volta in volta singole parti di prodotto autoconsistenti, cioè utilizzabili dagli utenti nell'ambiente di esercizio. Ciascun rilascio accresce quindi le funzionalità del prodotto finale che si evolve così ad ogni iterazione.

Il modello è particolarmente utile nei progetti di grandi dimensioni, complessi, con requisiti non tutti chiari fin dall'inizio, ed in cui sia possibile, necessario e conveniente rilasciare parti consistenti di prodotto in rilasci successivi.

4.7 Sommario dei modelli di ciclo di vita

La tabella che segue riassume il campo di applicazione di ciascun modello. Essa può risultare utile ai fini della scelta di quello più adatto al progetto. Ovviamente, la scelta è condizionata da molti fattori di dettaglio, alcuni dei quali determinanti ai fini della scelta finale. La descrizione fatta in precedenza rimangono comunque linee guida di sicura efficacia.

Figura 18. Campo di applicazione dei modelli di ciclo di vita.

Modello di ciclo di vita	Campo di applicazione nei progetti
A cascata (*Waterfall*)	Il modello è di particolare efficacia nei progetti in cui il prodotto finale possa essere realizzato con un singolo rilascio ed i requisiti siano ben chiari fin dall'inizio del progetto e rimangano stabili.
A stadi (*Staged*)	Il modello è particolarmente adatto in progetti in cui il prodotto finale possa essere realizzato con rilasci successivi, ma i requisiti e la struttura dell'intero prodotto possano essere definiti sin dall'inizio del progetto.
Prototipale (*Prototiping*)	Il modello può essere utilizzato con successo per investigare diverse parti critiche del prodotto finale. E' adatto nei progetti in cui i requisiti siano poco chiari o si abbia necessità di sperimentare parti critiche del sistema.
Iterativo (*Iterative*)	Il modello è fortemente raccomandato per progetti di grandi dimensioni, complessi e con modifiche ai requisiti.
A spirale (*Spiral*)	Il modello è utile nei progetti in cui i requisiti risutino vaghi e poco stabili, in quelli in cui si adottino nuove tecnologie e ancora poco conosciute.
Iterativo-incrementale (*Iterativo-Incremental*)	Il modello è utile nei progetti di grandi dimensioni, complessi, con requisiti non tutti chiari fin dall'inizio e con la possibilità di mettere in esercizio le funzionalità in maniera graduale con più rilasci.

5 Il processo di sviluppo

5.1 Che cos'è un processo

E' ormai comunemente accettata la definizione di processo come l'insieme strutturato di procedure e attività che trasformano dati in ingresso (input) in informazioni elaborate in uscita (output).

Il processo di sviluppo software è, a sua volta, definito come un insieme di procedure e attività, raggruppate in fasi, che trasformano i requisiti (ricevuti come elementi in ingresso) in un prodotto software (risultato in uscita).

E' altresì consolidato il modello di processo di sviluppo del software (detto anche "ciclo di vita del software") costituito dalle fasi mostrate nella tabella che segue. Per ciascun fase sono definiti i suoi output e criteri che determinano il suo completamento (criteri di uscita dalla fase).

Figura 19. Sintesi del processo di sviluppo completo.

Fase	Output	Criterio di uscita
Studio di fattibilità	Requisiti di massima Soluzione di massima Piano di massima	Approvazione
Pianificazione	Piano di Progetto Piano di Qualità Requisiti utente (completato)	Approvazione
Analisi	Specifiche funzionali Piano dei test	Approvazione
Disegno	Disegno di dettaglio Specifiche di test	Consegna
Realizzazione	Codice sorgente Codice di test Manuale utente Manuale di gestione	Consegna
Collaudo	Applicazione	Accettazione
Rilascio in esercizio	Applicazione	Sistema in esercizio

La tabella che segue descrive la classificazione dei progetti secondo il livello di complessità, mentre la tabella successiva mostra quali output produrre per ciascuna tipologia di progetto.

Figura 20. Classificazione dei progetti secondo la complessità e/o dimensione.

Complessità	Dimensione	Caratteristica
Alta	Fino a 1000 FP	Include nuovi progetti di piccole dimensioni, interventi di manutenzione correttiva, migliorativa, evolutiva e adeguativa.
Media	Da 1000 FP a 20.000 FP	Include nuovi progetti di sviluppo di medie dimensioni ma anche interventi di manutenzione evolutiva consistenti (anche se difficilmente si pianificano interventi di tali dimensioni).
Bassa	Oltre 20.000 FP	Include nuovi progetti di sviluppo. Difficilmente saranno previsti interventi di manutenzione di tali dimensioni.

La tabella che segue mostra la personalizzazione suggerita per i progetti in base alla complessità. Gli output da produrre sono progressivamente semplificati man mano che la complessità diminuisce.

Figura 21. Output da produrre secondo la classificazione dei progetti.

	Output da produrre in base alla complessità del progetto		
Fase	**Alta**	**Media**	**Bassa**
Studio di fattibilità	• Requisiti di massima • Soluzione di massima • Piano di massima	• Progetto di massima: requisiti, soluzione e piano di massima (documento unico)	• Progetto di massima: requisiti, soluzione e piano di massima (documento unico semplificato)
Pianificazione	• Piano di Progetto • Piano di Qualità • Requisiti utente (completato)	• Progetto di dettaglio: requisiti e soluzione di dettaglio (documento unico) • Piano di progetto, di qualità e di test (documento unico)	• Progetto di dettaglio: requisiti, soluzione, piano di progetto, di qualità e di test (documento unico semplificato)
Analisi	• Specifiche funzionali • Piano dei test	• Specifiche funzionali	• Specifiche funzionali e disegno di dettaglio (documento unico semplificato)
Disegno	• Disegno di dettaglio • Specifiche di test	• Disegno di dettaglio • Specifiche di test	• Specifiche di test
Realizzazione	• Codice sorgente • Codice di test • Manuale utente • Manuale di gestione	• Codice sorgente • Codice di test • Manuale utente • Manuale di gestione	• Codice sorgente • Codice di test • Manuale utente • Manuale di gestione
Collaudo	• Applicazione	• Applicazione	• Applicazione
Rilascio	• Applicazione	• Applicazione	• Applicazione

5.2 Processo Unified Process (UP)

Il Processo Unificato (UP) é un processo industriale per lo sviluppo di software che ha l'obiettivo di produrre applicazioni di alta qualità, rispondenti alle necessità degli utenti, con un approccio disciplinato e con un controllo accurato dei tempi e dei costi.

Il Processo Unificato, ideato inizialmente da Ivar Jacobson, Grady Booch e James Rumbaugh, ed è stato sviluppato e commercializzato dalla Rational Software come Rational Unified Process (RUP). Il RUP raccoglie molte tra le "best practices" fondamentali dello sviluppo di applicazioni software, presentandole in una visione unitaria applicabile ad una vastissima tipologia di progetti ed organizzazioni. In particolare, il processo risulta:

- guidato dai casi d'uso (*Use Case*): sia la raccolta dei requisiti che i successivi passi sono guidati dai casi d'uso.
- fondato sull'architettura: fornire una visione generale del sistema adattabile al cambiamento dei requisiti.
- Iterativo e Incrementale: suddividere il progetto in miniprogetti, dove ogni miniprogetto rappresenta un'iterazione, il cui risultato è un'iterazione del prodotto da ottenere.

La figura riportata di seguito mostra lo schema delle fasi di cui si compone il processo RUP e l'impegno previsto per ciascuna di esse (forme colorate).

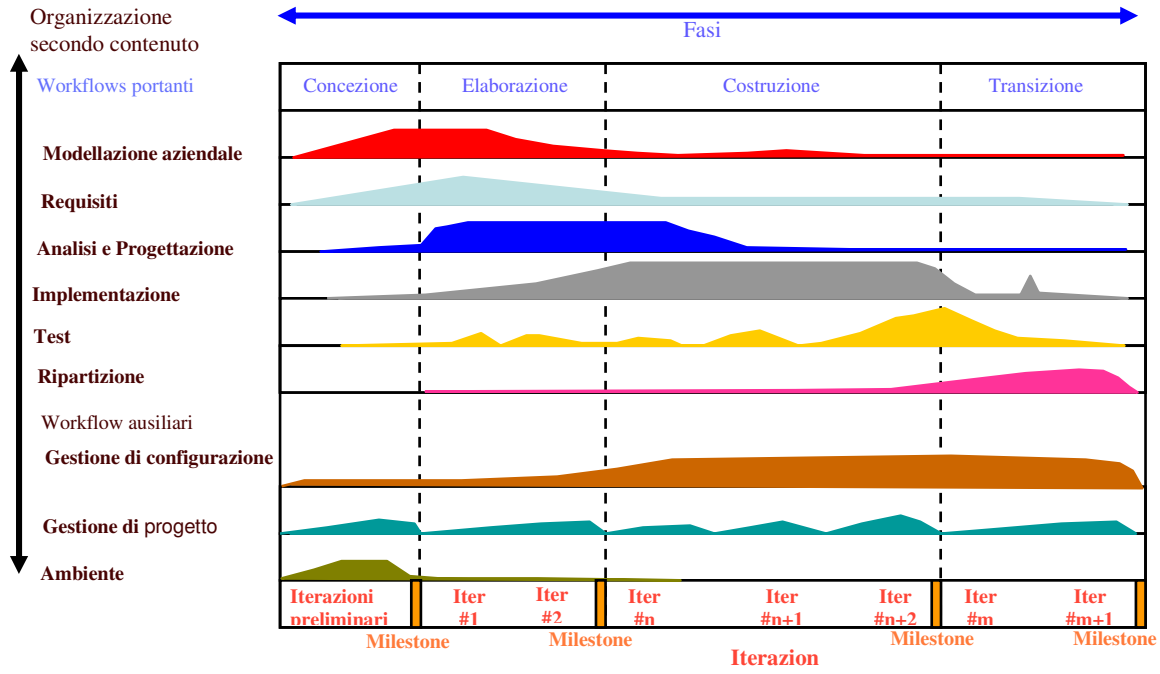

Figura 22. Processo RUP e schema delle fasi (tratto dal Web).

Bibliografia

Di seguito è riportata la bibliografia di base cui l'autore fa riferimento per le sue attività. Essa è distinta per argomento: Qualità, ISO9000, CMMI, Sviluppo professionale, Ingegneria del software ecc. Essa non pretende di essere esaustiva né definitiva; rappresenta invece una buona base di partenza per approfondire i temi trattati nel libro.

Qualità

[LEGA06] Riccardo Lega - 2006
La Patente Europea della Qualità (EQDL)
Franco Angeli

[LEONA00] Erika Leonardi - 2000
Capire la qualità – ISO9000 – Tutto quello che occorre capire per applicare con profitto le nuove norme
Il Sole 24 Ore

[BARBA98] Filippo C. Barbarino, Erika Leonardi - 1998
ISO9000 Sistema Qualità e Certificazione
Il Sole 24 Ore

[PINDE96] Mark Pinder & Stuart McAdam - 1996
La consulenza interna
Jackson Libri

[MUNR94] Lesly Munro-Faure, Malcom Munro-Faure - 1994
Qualità totale: tecniche di attuazione
Jackson Libri

[PALA94] Giorgio Pala - 1994
La qualità. Perché, per chi e come farla
Franco Angeli

[BANCI93] Alessandro Banci, Giuseppe Iacono - 1993
La qualità nei progetti software
Franco Angeli

[NOCEN93] Stefano Nocentini - 1993
Il sistema di qualità del software
ETASLIBRI

[WHEEL93] Donald J. Wheeler - 1993
Understanding Variation – The Key To Managing Chaos
SPC Press

[ZEITHL91] Valerie A. Zeithaml, A. Parasuraman, Leonard L. Berry - 1991
Servire qualità
McGraw-Hill

[RUMML90] Geary A. Rummler and Alan P. Brache - 1990
Improving performance
Jossey-Bass Publishers

[CROSB86] Philip B. Crosby - 1986
La qualità non costa
McGraw-Hill

ISO9000

[ISO9001:2000] ISO/IEC 9001:2000
Quality Management Systems – Requirements

Quality Management Systems – Fundamentals and Vocabulary

Quality Management Systems – Guidelines for performance improvement

[ISO9003:2004] ISO/IEC 9003:2004
Software and Systems Engineering – Guidelines for the application of ISO 9001:2000 to computer software

ISO9126-1:2001] ISO/IEC 9126-1:2001
Software Engineering – Product Quality Part 1: Quality Model

ISO9126-2:2002] ISO/IEC 9126-2:2002
Software Engineering – Product Quality Part 2: External Metrics

ISO9126-3:2002] ISO/IEC 9126-3:2002
Software Engineering – Product Quality Part 3: Internal Metrics

ISO9126-4:2002] ISO/IEC 9126-4:2002
Software Engineering – Product Quality Part 4: Quality In Use Metrics

[ISO12207:2003] ISO/IEC 12207:2003
Information Technology – Software Lifecycle Processes

[ISO10006:2005] ISO/IEC 10006:2005
Information Technology – Software Project Management

[ISO15271:98] ISO/IEC TR 15271:1998
Information Technology – Guide for ISO/IEC 12207 Software Lifecycle Processes

[ISO16326:99] ISO/IEC TR 16326:1999
Software Engineering – Guide for the application of ISO/IEC 12207 in Project Management

Il concetto moderno (ma ormai divenuto un classico) è che la qualità di un prodotto dipende dalla bontà del processo con cui è stato realizzato. Processi produttivi di alta qualità consentono di realizzare prodotti di elevata qualità. Ma un processo di qualità non produce sempre e necessariamente prodotti di qualità. L'ingrediente da aggiungere è il "controllo del processo". Il controllo è proprio quella attività che permette di verificare costantemente l'efficacia del processo e di ottenere, al termine, i risultati attesi. In tal senso vale la definizione di "processo maturo" o meglio di "livello di maturità di un processo". In tal senso i modelli attuali di maturità dei processi definiscono processi, attività, responsabilità, competenze, tecniche, metodi e metriche per disegnare e migliorare i processi in modo da garantire i risultati attesi.

Nel manuale si discutono alcuni elementi che permettono di scegliere la strategia di sviluppo più adatta ad un progetto software in base ad alcuni elementi critici come le caratteristiche del prodotto da sviluppare, le modalità con cui si intende rilasciare in esercizio la soluzione finale, il coinvolgimento del cliente nel progetto di sviluppo ed i costi stimati del progetto stesso. La strategia scelta permette quindi di definire il ciclo di vita (o il processo di sviluppo) più adatto: Waterfall, Iterativo-incrementeale, Prototipale, a Spirale, ecc. Si possono scegliere anche più di un modello per sottoprogetti in cui il progetto generale può essere scomposto. Dalla strategia di sviluppo deriva poi la strategia di test e collaudo. Tutto è legato in maniera coordinata, logica e … qualitativa!

Ercole Colonese svolge la sua attività di consulente essenzialmente nell'area dello sviluppo applicativo. La sua esperienza è maturata in moltissimi anni di lavoro presso i laboratori internazionali IBM di sviluppo prodotti software dove ha ricoperto ruoli tecnici e manageriali. Ha realizzato numerosi sistemi qualità aziendali certificati ISO9000 presso piccole, medie e grandi aziende, pubbliche e private. Ha implementato modelli di eccellenza (EFQM, Malcom Bauldrige ecc.). Ha condotto diversi progetti di reingegnerizzazione dei processi di sviluppo software in ottica di miglioramento delle performance e della qualità. Ha applicato con successo i modelli di maturità dei processi come il *Capability Maturity Model* del *Software Engineering Institute* (SEI-CMM e CMMI). Come docente, ha tenuto corsi sulla qualità del software e le metodologie di sviluppo presso il Learning Center IBM. Ha pubblicato diversi articoli sulle tematiche dello sviluppoi software ed il ruolo consulenziale. Attualmente è professore aggiunto di Ingegneria del software presso l'Università di Roma Tor Vergata.

e-mail: ercole@colonese.it www.colonese.it