

G. Casadei A.G.B. Teolis

# LA GESTIONE DEI PROGETTI INFORMATICI

## 4. I modelli

4.1 Modello a cascata

4.2 Modello a V

4.3 Modello incrementale

4.4 Modello a prototipi

4.5 Modello a spirale

4.6 I modelli "leggeri"

4.7 Esercizi

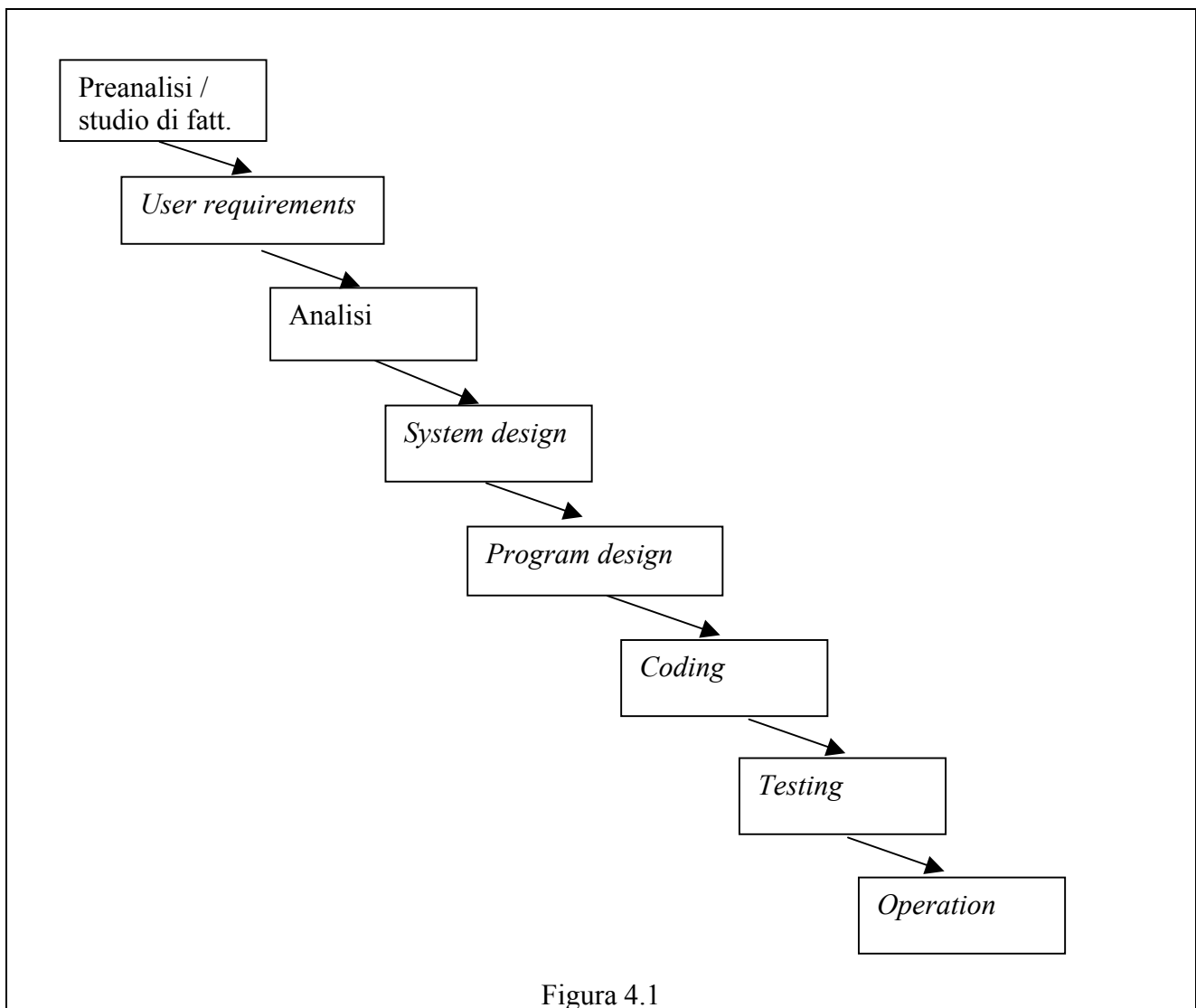
**REVISIONE DEL 10/11/2003**

## 4. Modelli

Le situazioni in cui si ipotizza di realizzare un progetto *software* sono le più varie, caratterizzate dai più diversi vincoli come per esempio: l'obbligo di terminare entro un tempo fissato, obiettivi innovativi (e rischiosi), specifiche, caratteristiche o prestazioni difficili da definire, necessità di una realizzazione anche parziale ma rapida. Per garantire il successo del progetto è opportuno adeguarne la conduzione alle diverse situazioni: a tal fine l'esperienza ha consentito di individuare un numero discreto di tipi o più propriamente modelli per la conduzione di progetti. Ciascuno di questi modelli è caratterizzato da una sua specifica strutturazione in fasi e da un ambito tipico di applicazione. I cinque modelli sono: modello a cascata (*waterfall model*), modello a V (*V model*), modello incrementale (*incremental delivery*), modello a prototipi (*prototyping model*) e modello a spirale (*spiral model*).

### 4.1 Modello a cascata

Il modello a cascata è caratterizzato da 8 fasi di sviluppo, organizzate come mostrato in Figura 4.1.



La caratteristica del modello non è tanto costituita dal numero o dalla descrizione delle fasi quanto dal fatto che si passa alla fase successiva solo quando la precedente è completamente terminata.

Ogni fase deve terminare con un documento formale approvato dall'organo responsabile dell'intero progetto. In questo modello, eventuali ripensamenti (che implicano rimettere in discussione i risultati di fasi precedenti) hanno un impatto molto rilevante sui tempi e sui costi e quindi sono una eventualità da prevenire con molta cura. Nella pratica, può accadere che, a seguito di situazioni particolari, si sia tentati di sottovalutare l'importanza di qualcuna di queste fasi e quindi di ometterla; anche questo deve essere assolutamente evitato perché verrebbero meno la produzione e l'approvazione del documento necessario per avviare la fase successiva.

### Le fasi del modello a cascata

- a) **Preanalisi/Studio di fattibilità.** Questa fase ha come obiettivo quello di stabilire l'opportunità economica del progetto, esaminare le alternative possibili per la realizzazione, valutarne i costi e i benefici e sceglierne una; tipicamente, tra le alternative da prendere in considerazione, sono: "non fare nulla" (tenendo conto che anche questa scelta ha un costo, per esempio di gestione o di manutenzione), una "soluzione di minima" e una "soluzione di massima"; tutte devono essere prese in considerazione per un periodo da 3 a 5 anni. Altrove, sono illustrati due indici ai quali è opportuno che l'*output* di questa fase sia uniformato. Il progetto può finire in questa fase, quando l'organo decisionale sceglie l'alternativa "non fare nulla"; altrimenti i dati raccolti devono consentire una pianificazione (anche solo di massima).
- b) **Raccolta dei requisiti dell'utente.** Il documento *output* di questa fase è il più critico per la riuscita del progetto perché deve raccogliere in maniera strutturata le aspettative e le richieste di tutte le categorie di utenti (si veda la descrizione dei ruoli). Esso deve essere sufficientemente sintetico per risultare leggibile e quindi di approvazione significativa da parte di persone in generale non tecnicamente esperte sui problemi connessi con la realizzazione di sistemi *software*; tuttavia deve essere sufficientemente dettagliato da descrivere in *modo esplicito* tutte le condizioni che il sistema da realizzare deve soddisfare.
- c) **Analisi.** In questa fase, condotta tipicamente da esperti informatici e da esperti delle problematiche coinvolte, vengono descritti tutti i dati (entità e relazioni) e le funzioni da automatizzare.
- d) **System design.** In questa fase viene prodotto il documento a maggior contenuto tecnologico, nel quale viene descritta, in ogni dettaglio, l'architettura *hardware* e *software* del sistema da realizzare e quella del sistema di sviluppo.
- e) **Program design.** In questa fase, tenuto conto delle strutture dei dati e delle funzionalità descritte nella fase c) e degli ambienti tecnologici descritti nella fase d), sono documentati dettagliatamente, tutti i moduli *software* (transazioni, programmi, procedure, ecc.) da implementare; il suo *output* è, di solito, il documento di mole maggiore fra quelli prodotti dalle otto fasi.
- f) **Programmazione/codifica.** In questa fase, l'*output* prodotto è rappresentato dai programmi, dalla documentazione che deve essere usata per la loro manutenzione e dai manuali d'uso per l'utente.
- g) **Test.** In questa fase vengono generati e costruiti data base di prova, "popolati" opportunamente per verificare le funzionalità e le prestazioni del sistema. La documentazione prodotta documentata le prove effettivamente eseguite e il loro esito. Viene inoltre prodotta la documentazione per la installazione e la gestione (*backup, checkpoint*).
- h) **Messa in esercizio.** In questa fase deve essere completata la installazione dello *hardware* e del *software* previsti e, inoltre, vengono generati e "popolati" i data base di gestione. Infine, vengono addestrati e resi operativi gli utenti finali. Non raramente, questa è la fase più lunga e delicata del modello a cascata.

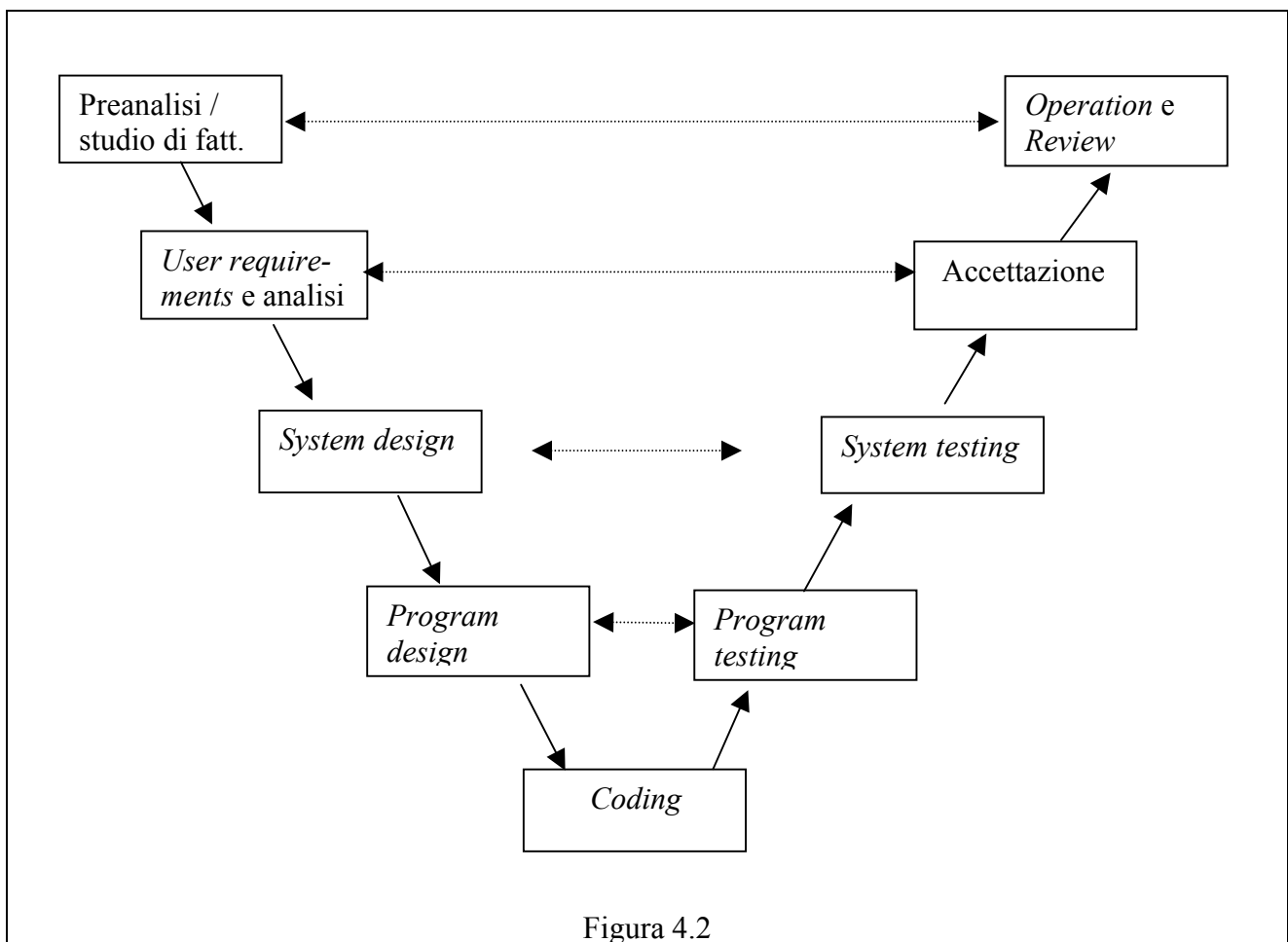
### Caratteristiche del modello a cascata

I vantaggi di questo modello discendono essenzialmente dal fatto che le fasi hanno un contenuto di lavoro sostanzialmente certo e gli eventuali ricicli previsti sono solo "locali" (cioè riguardano al più solo la fase precedente) e richiedono poco lavoro aggiuntivo; quindi, in generale, le previsioni sui costi e sui tempi sono molto attendibili. Le risorse specialistiche coinvolte in ogni fase sono liberate

alle fine della fase (e sicuramente non più necessarie alla fine della fase successiva). Il modello si applica quando i requisiti dell'utente sono strettamente determinabili prima dell'inizio dell'implementazione ed è ragionevole aspettarsi che non cambino per tutta la sua durata. L'architettura deve essere "nota" e in particolare non deve riservare sorprese per ciò che riguarda le prestazioni. Inoltre il tempo a disposizione è sufficiente a procedere sequenzialmente ed è accettabile da parte dell'utente che *tutte* le funzionalità vadano in esercizio *tutte* insieme alla fine del progetto. Un caso tipico in cui è ragionevole applicare il modello a cascata è la cosiddetta "seconda informatizzazione" per aggiornamento tecnologico. Un esempio è dato dal rifacimento di un sottosistema del sistema informativo di una azienda: in questo caso gli utenti sanno cosa aspettarsi (le funzioni del vecchio sistema più quelle richieste dall'esperienza maturata). È estremamente dannoso adottare il modello a cascata se c'è il ragionevole sospetto che i requisiti dell'utente non sono chiaramente enunciabili, possono variare nel tempo oppure la tecnologia non è consolidata e può riservare degli imprevisti sui costi e sulle prestazioni.

#### 4.2 Modello a V

Come si è visto, il modello a cascata, che nelle implementazioni correnti è uno dei più usati, si basa su due presupposti fondamentali:



- il committente accetta che tutto l'*output* del progetto venga messo in esercizio in blocco nell'ultima fase temporale del progetto;
- i requisiti degli utenti sono chiari e la tecnologia è ben nota sia al committente sia agli sviluppatori.

Può accadere che il secondo presupposto non sia valido, cioè il committente accetta ancora che i risultati siano consegnati in blocco alla fine, ma o non è in grado di esplicitare in modo definitivo i requisiti degli utenti o la tecnologia, di sviluppo o di esercizio, non è stabile oppure non è ben conosciuta. Per far fronte a questa situazione, è necessario modificare il modello a cascata nel cosiddetto modello a V che è caratterizzato da interazioni anche non locali tra le diverse fasi. In questo modello saranno quindi possibili dei ricicli non solo alla fase precedente e, cosa ancor più importante, esiste una fase finale di revisione critica di tutto il progetto. Questa fase finale, le cui motivazioni saranno ampiamente trattate successivamente, è necessaria e cruciale affinché il committente e i realizzatori inneschino un processo di “apprendimento” globale che permetterà loro di gestire i progetti futuri con la consapevolezza e le conoscenze acquisite dall’elaborazione dell’esperienza. Il modello a V è caratterizzato (tipicamente) da 9 fasi di sviluppo, organizzate come mostrato in Figura 4.2.

Il nome del modello deriva dal fatto che le nove fasi sono “in qualche modo” simmetriche nel senso che da una fase si può riciclare non solo a quella precedente, ma anche a quella alla stessa altezza se le si dispone in uno schema a “V”. Ogni fase deve terminare con un documento formale approvato dall’organo responsabile dell’intero progetto; in questo modello, quindi, i ripensamenti hanno un impatto prevedibile, sia sui modi di condurre il progetto sia sui tempi e sui costi. Come si evince dalla figura 4.2, le ultime quattro fasi sono cruciali perché al termine di ciascuna di queste occorre decidere se passare alla successiva o se innescare un ciclo anche parziale alla fase simmetrica precedente. Ovviamente, anche con questo modello, è assolutamente necessario che l’*output* di ogni fase sia adeguatamente e formalmente documentato.

### Le fasi del modello a V

- a) **Preanalisi/Studio di fattibilità.** Questa fase ha, come nel modello a cascata, l’obiettivo di stabilire l’opportunità economica del progetto, di esaminare le alternative possibili per la realizzazione dettagliando i costi e i benefici e sceglierne una; in questo caso, la fase è particolarmente delicata perché i requisiti degli utenti e/o la tecnologia non sono completamente note e quindi, le conseguenti decisioni non possono essere ritenute definitive. È anche importante, in questa fase e nelle tre successive, che le varie decisioni prese siano tra loro il più possibile indipendenti in modo tale che, quando si ricicla, si possano realizzare facilmente solo le correzioni conseguenti alle decisioni modificate.
- b) **Raccolta dei requisiti dell’utente e analisi.** Questa fase congloba le due corrispondenti del modello a cascata, dal momento che i presupposti su cui si basano sono relativamente instabili perché l’utente può non essere in grado di esplicitare in modo definitivo i requisiti. In questo caso, è assolutamente necessario che il documento di *output* sia modulare in maniera che gli eventuali ricicli riguardino solo porzioni localizzate delle funzionalità. Esso deve essere sufficientemente sintetico per risultare leggibile e quindi di approvazione significativa anche da parte di committenti e utenti che non abbiano chiare le aspettative sui risultati attesi dal progetto.
- c) **System design.** In questa fase (come nella corrispondente del modello a cascata) viene prodotto il documento a maggior contenuto tecnologico, nel quale viene descritta, in ogni dettaglio, l’architettura *hardware* e *software* dell’intero sistema da realizzare e quella del sistema di sviluppo. È di nuovo necessario che il documento sia modulare, specialmente per le componenti tecnologiche meno conosciute o sperimentate.
- d) **Program design.** In questa fase, tenuto conto delle funzionalità descritte nella fase b) e delle specifiche tecnologiche descritte nella fase c), sono documentati dettagliatamente, tutti i moduli *software* (transazioni, programmi, procedure, ecc.) da implementare; il suo *output* è, di solito, il documento di mole maggiore fra quelli prodotti dalle 9 fasi.
- e) **Programmazione/codifica.** In questa fase, l’*output* prodotto è rappresentato dai programmi e dalla documentazione che deve essere usata per la loro manutenzione. Non vengono ancora pro-

dotti (a differenza del modello a cascata) i manuali d'uso per l'utente, che saranno predisposti della fase successiva.

- f) **Program test.** Questa fase è simmetrica della fase d): vengono generati e costruiti *data base* di prova, “popolati” opportunamente per verificare le funzionalità e le prestazioni di ciascuno dei moduli realizzati. La documentazione prodotta documenta le prove effettivamente eseguite e il loro esito. In questa fase è molto importante il coinvolgimento degli utenti finali, perché, data l'incertezza evidenziata durante la definizione dei requisiti, questa è sostanzialmente la prima volta che essi si rendono conto delle funzionalità fornite dal sistema; in questa situazione sono possibili e frequenti, non solo interventi a livello di programmazione/codifica, ma anche ripensamenti nell'accorpamento in moduli delle varie funzionalità e quindi un riciclo alla fase d). Per ogni modulo accettato, viene predisposta la documentazione utente.
- g) **System test.** Questa fase è simmetrica della fase c). Terminato il *test* di tutti i programmi, viene eseguito un *test*, in genere da parte dei realizzatori con l'assistenza degli utenti di riferimento, per controllare che le prestazioni globali del sistema siano accettabili e le macro funzioni (soprattutto quelle di sistema come la regolamentazione degli accessi, la sicurezza,...) siano correttamente realizzate. Per le eventuali parti non accettate, si ritorna alla fase c) apportando le opportune modifiche e successivamente alle fasi successive per le parti interessate. Si vede chiaramente come la modularità sia essenziale per realizzare efficacemente questo modello. Terminati tutti gli eventuali ricicli, viene infine prodotta la documentazione per la installazione e la gestione (*backup*, *checkpoint*, ecc.).
- h) **Accettazione da parte dell'utente.** Questa fase è simmetrica della fase b); essa non è presente (esplicitamente) nel modello a cascata perché in tale modello le fasi dalla definizione dei requisiti a quella di test sono sufficienti ad assicurare l'aderenza dei prodotti costruiti alle esigenze reali. Nel caso in esame, invece, è necessario questo controllo finale per garantire che tutti i requisiti (impliciti ed espliciti) siano soddisfatti: ove ciò non accada, è necessario ritornare alla fase simmetrica. Questa fase ha termine con il controllo di tutta la documentazione necessaria.
- i) **Messa in esercizio e review del progetto.** In questa fase deve essere completata la installazione dell'*hardware* e del *software* previsti e, inoltre, vengono generati e “popolati” i *data base* di gestione; vengono addestrati e resi operativi gli utenti finali. In questo modello, per le stesse ragioni per cui è possibile un riciclo alla fase b), quasi sempre la messa in esercizio è lunga e delicata; al limite non sono rari i casi in cui l'intero progetto viene cancellato a questo stadio, perché non è di fatto possibile utilizzare i suoi prodotti. Anche in questo caso “negativo”, tutto il lavoro fatto e le attività svolte devono contribuire ad aumentare le conoscenze e le competenze dei partecipanti mediante una formale e documentata attività di *review* del progetto.

### Caratteristiche del modello a V

I vantaggi di questo modello discendono essenzialmente dal fatto che consente di gestire i ricicli, almeno quando questi sono “controllabili”, cioè quando ci si aspetta che siano più probabili i ricicli della parte bassa della V rispetto a quelli della parte alta; in questo caso, le previsioni sui costi e sui tempi diventano sempre più attendibili, a mano a mano che si procede con il progetto. Durante lo studio di fattibilità occorre una valutazione accurata dei rischi per garantire la “controllabilità” dei ricicli. Le risorse specialistiche coinvolte in ogni fase sono liberate alle fine della fase ma sono non più necessarie solo alla fine della fase simmetrica. Un caso tipico in cui è ragionevole applicare il modello a V è la cosiddetta “prima informatizzazione” che succede quando in una azienda si vuole informatizzare per la prima volta le attività di un certo settore e non esiste una vasta e utilizzabile esperienza accumulata da progetti simili terminati con successo.

### 4.3 Modello incrementale

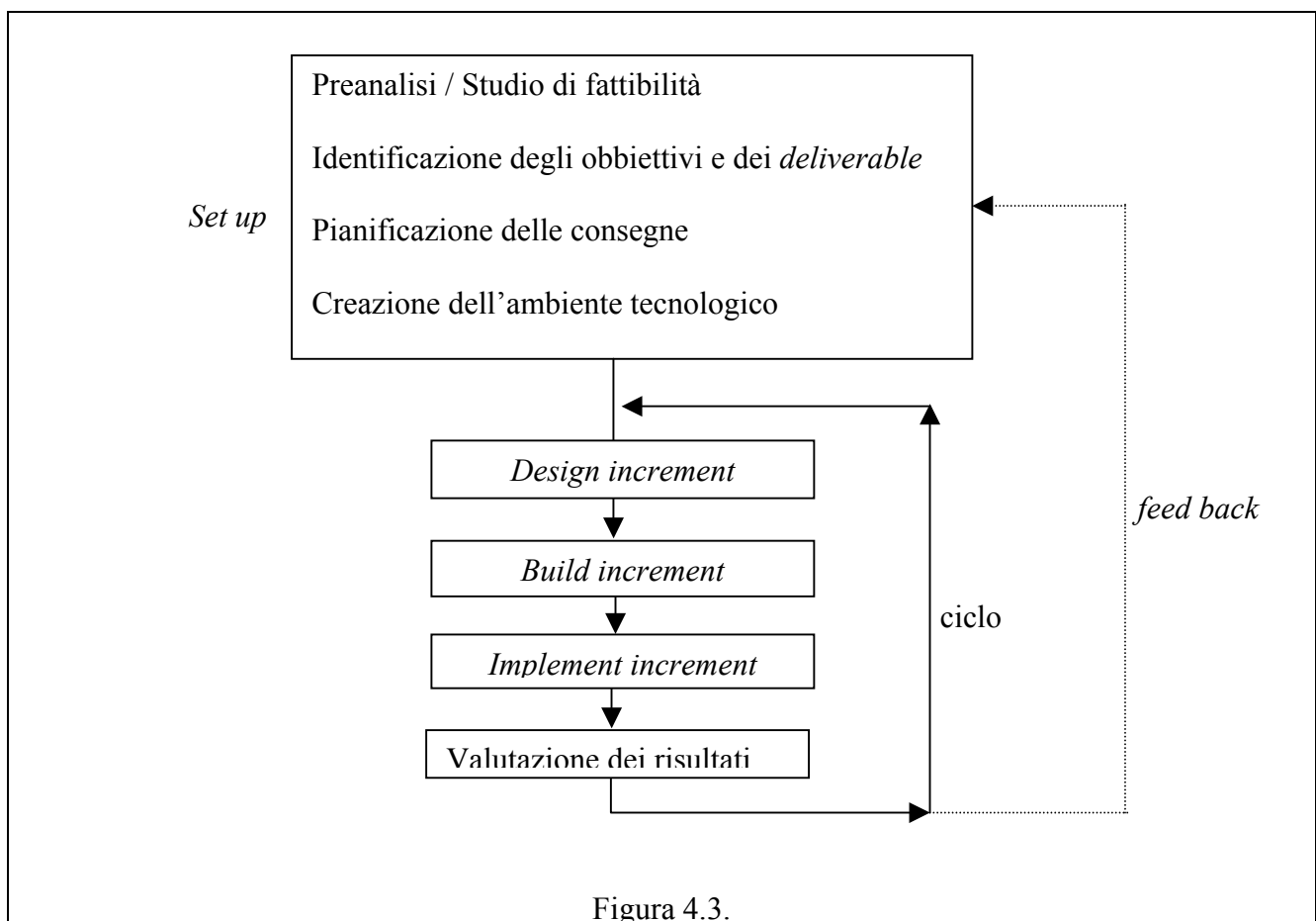
Come è stato ricordato, il modello a cascata si basa sui seguenti due presupposti fondamentali:

- il committente accetta che tutto l'*output* del progetto venga messo in esercizio in blocco nell'ultima fase temporale del progetto;
- i requisiti degli utenti e la tecnologia sono ben noti e accettati sia dal committente sia dallo sviluppatore.

Quando (solo) il secondo presupposto non è valido si adotta il modello a V. Quando, invece, non è valido (solo) il primo dei due presupposti, si adotta il modello "a consegna incrementale" (o semplicemente "incrementale"), che permette al committente di avere, entro tempi brevi, una prima consegna di risultati, anche se parziali, e consegne successive via via durante lo svolgersi del progetto.

Questa situazione si può presentare quando esigenze estranee al singolo progetto e alla sua economicità impongono che alcuni sottosistemi siano messi in esercizio rapidamente; rispettare questa condizione si paga necessariamente con un maggior costo complessivo del progetto, un allungamento del tempo totale "*elapsed*" e (eventualmente) con un aumento dei rischi.

In questo modello si distingue un blocco di fasi iniziali, dette di *setup*, e un ciclo iterativo di consegne che viene ripetuto più volte finché l'intero progetto non è terminato (con l'ultima consegna), come mostrato in Figura 4.3.



Il blocco di *setup* e il ciclo iterativo sono entrambi formati da quattro fasi. Alla fine di ogni ciclo, l'ultima fase permette di decidere se

- riciclare per produrre nuove consegne,
- ritornare (per un aggiustamento) a una delle fasi di *setup*,
- ritenere il progetto terminato.

Ovviamente, anche con questo modello, è assolutamente necessario che l'*output* di ogni fase sia adeguatamente e formalmente documentato.

### Le fasi del modello incrementale

Quando le condizioni generali del progetto richiedono l'applicazione di questo modello (cioè quando il committente esige alcuni risultati "immediati"), se ne deve tenere conto già nello studio di fattibilità e nella identificazione dei requisiti che risultano abbastanza diversi dagli analoghi previsti per i modelli precedenti. Infatti, dal punto di vista globale, la differenza principale deriva dal fatto che ora occorre determinare prima l'ambiente tecnologico e il piano delle consegne e solo successivamente fare l'analisi (dettagliata) dei problemi da risolvere. Anche per queste ragioni, questo modello si presta bene nei casi di seconda informatizzazione.

#### Setup

- a) **Preanalisi/Studio di fattibilità.** Questa fase ha, come nei precedenti modelli, l'obiettivo di stabilire l'opportunità economica del progetto, di esaminare le alternative possibili per la realizzazione, dettagliando i costi e i benefici, e sceglierne una. In questo caso, occorre anche prevedere i rischi conseguenti alla eventualità di dover modificare non solo parti del sistema appena realizzate (come nei modelli precedenti), ma anche parti del sistema già in esercizio.
- b) **Raccolta dei requisiti degli utenti e identificazione dei prodotti da consegnare (*deliverable*).** In questa fase viene preparato un documento che descrive in maniera dettagliata i requisiti degli utenti e fissa i gruppi di funzionalità associati a ciascuna delle singole consegne in cui si dovrà articolare il progetto.
- c) **Pianificazione delle consegne.** A differenza dai due precedenti modelli, una pianificazione anche di massima del progetto è possibile solo in questa fase, perché solo alla fine della fase precedente sono stati definiti il numero e la complessità dei cicli di consegna.
- d) **System design e macroanalisi.** In questa fase, l'obiettivo principale è definire l'ambiente tecnologico e l'architettura opportuni per il progetto; questa attività può richiedere l'approfondimento di alcune problematiche.

#### Ciclo di consegna

- e) **Increment design.** In questa fase si analizzano tutti i problemi connessi con le funzionalità associate alla consegna in esame (corrisponde alla fase di analisi degli altri modelli).
- f) **Increment building.** In questa fase si fa il *program design* e la codifica di tutte le funzionalità in esame; si prepara la documentazione per la manutenzione e per l'utente; si fanno i *test*.
- g) **Increment deploy.** In questa fase si fa l'installazione, l'addestramento e la messa in esercizio delle funzionalità in esame e si prepara la documentazione di gestione.
- h) **Valutazione dei risultati.** In questa fase occorre verificare se a seguito della consegna appena fatta occorre rivedere il contenuto e la pianificazione delle altre consegne, l'architettura dell'intero sistema o la macroanalisi.

### Caratteristiche del modello incrementale

I vantaggi di questo modello, a causa delle consegne distribuite nel tempo, discendono essenzialmente dal fatto che

- il *feedback* delle prime consegne influenza le successive e quindi gli errori strategici tendono ad evidenziarsi durante lo svolgimento del progetto (piuttosto che solo alla fine);
- gli eventuali errori di analisi tendono ad essere "piccoli" o poco rilevanti perché la consegna è temporalmente vicina alla fase di indagine dei problemi;
- gli utenti interagiscono meglio con il *team* di progetto perché vedono immediatamente il risultato del loro coinvolgimento;
- ciascuna consegna è relativamente "piccola" e quindi facile da gestire;
- il progetto può essere sospeso, interrotto o cancellato lasciando comunque dei risultati utili (le consegne già effettuate).

Gli svantaggi del modello sono

- il cosiddetto *breakage*, cioè una consegna può richiedere il rifacimento di (parte delle) consegne precedenti;
- la produttività (quantità di prodotto/la durata del progetto) può essere più bassa di quella del modello a cascata;
- fare un “buon piano” di consegne incrementale è molto più difficile che pianificare un modello a cascata.

A commento dell'ultimo punto si può affermare che una regola empirica suggerisce che ogni consegna dovrebbe riguardare tra il 10 e il 30 per cento del prodotto dell'intero progetto; quindi, in generale, si devono pianificare un numero di consegne compreso fra 3 e 10; un numero maggiore potrebbe risultare “fastidioso” per gli utenti coinvolti in più di una consegna. Infatti, il tempo di adattamento di un utente, cioè l'intervallo fra una sollecitazione significativa (per esempio il coinvolgimento nella installazione di un *deliverable*) e la successiva oscilla fra le 4 e le 12 settimane (talvolta è molto maggiore). Inoltre, lo *stress* “cognitivo” può essere aggravato dal fatto che questo modello, essendo spesso applicato nei casi di seconda informatizzazione, impone all'utente di gestire contemporaneamente la vecchia tecnologia e la nuova, finché non siano state effettuate tutte le consegne che lo riguardano.

#### **4.4 Modello a prototipi**

I modelli precedenti si caratterizzano per la presenza di entrambi o di uno solo dei seguenti due presupposti fondamentali:

- il committente accetta che tutto l'*output* del progetto venga messo in esercizio in blocco nell'ultima fase temporale del progetto;
- i requisiti degli utenti e la tecnologia sono ben noti e accettati sia dal committente sia dallo sviluppatore.

Ci sono dei casi, caratterizzati dalla incertezza tecnologica e dalla incertezza dei requisiti, in cui occorre necessariamente procedere per tentativi; vale a dire occorre procedere con piccole realizzazioni, coinvolgendo eventualmente un campione di utenti ridotto, per poi discutere su quanto realizzato per mettere a punto le funzionalità necessarie.

Si possono adottare due approcci:

- *usa e getta*: i prototipi che vengono costruiti e usati per ragionare con l'utente, anche in passi successivi, vengono eliminati e sostituiti dal sistema definitivo;
- *evolutivo*: si parte con un prototipo che viene via via modificato e sviluppato, per approssimazioni successive, fino allo stato di sistema definitivo (cioè accettato dal committente).

I principali vantaggi di questo modello sono riassunti dalla formula *learning by doing* e si possono elencare come segue:

- migliore comunicazione con gli utenti, che normalmente non leggono o non capiscono la documentazione tecnica mentre capiscono, in tutte le implicazioni, l'esempio funzionante costituito dal prototipo;
- viene valorizzato il ruolo degli utenti che si sentono maggiormente coinvolti nel progetto;
- è relativamente facile definire i *requirements* anche quando sono in partenza solo relativamente poco noti perché vengono costruiti e validati operativamente;
- si raggiunge facilmente la completezza e soprattutto la compatibilità e coerenza delle specifiche funzionali e tecnologiche;
- i cambiamenti sono quasi tutti evidenziati durante la fase di sviluppo e non quando il sistema è diventato operativo (con possibili risparmi);

- si migliora l'efficienza e l'efficacia del sistema: le prestazioni (tempi di risposta, tipi di interazione con l'utente, ecc.) sono immediatamente visibili e si può raggiungere quindi immediatamente il compromesso più opportuno (condiviso con l'utente) tra costo e prestazioni.

Gli svantaggi del modello sono

- mancanza di controllo sugli obiettivi del progetto: l'utente può essere tentato di chiedere "troppo";
- è difficile mantenere il controllo sui costi e sui tempi ipotizzati;
- richiede *hardware* più potente, sia perché il sistema di sviluppo deve avere le medesime caratteristiche di quello di gestione, sia perché gli ambienti di sviluppo utili in questo modello tendono ad essere poco efficienti, cioè a consumare più risorse di quelli usati in altri modelli;
- è richiesta la vicinanza fisica tra sviluppatori e utenti e quindi non sono possibili le economie caratteristiche delle "fabbriche di *software*".

Talvolta, invece che per l'intero progetto, la tecnica di prototipazione viene usata per sostituire le fasi e), f) e g) (*program design*, programmazione/codifica e *test*) del modello a cascata, che normalmente non prevede molta interazione fra sviluppatori e utenti, condensandole in una sola che prevede una collaborazione stretta; in questo modo si riesce ad anticipare le modifiche che, nel modello a cascata, gli utenti possono richiedere solo nella fase g).

Il modello a cascata, con questa variante, è usato abbastanza spesso e quindi esistono esperienze comparative con quello più tradizionale; in particolare si può affermare che nell'interazione con l'utente il 35% di modifiche viene introdotto per ragioni "cosmetiche", cioè per migliorare in maniera non essenziale la qualità dell'interazione, il 60% circa viene introdotto per migliorare il sistema e correggere errori locali di analisi e il 5% circa per rimediare a errori globali di analisi.

#### **4.5 Modello a spirale**

I modelli visti in precedenza sono tipici dell'ambiente standard di riferimento dei progetti trattati in questo contesto, cioè dei sistemi informativi delle aziende medio-grandi. In questi ambienti, tipicamente è possibile realizzare progetti "sequenziali", cioè progetti i cui prodotti (studio di fattibilità, requisiti degli utenti, analisi funzionale, analisi di dettaglio, ecc.) sono costruiti, almeno in linea di principio, sequenzialmente; i vari modelli finora illustrati differiscono di fatto solo per il grado di violazione "locale" di questo ordine (che è importante mantenere in qualche modo perché consente di abbassare i costi in quanto gli specialisti di ogni fase sono coinvolti per tempi limitati e, in linea di principio, prevedibili). Ambienti in cui sono di frequente sviluppati progetti *software* che non sempre consentono uno sviluppo sequenziale sono:

- applicazioni multimediali (come giochi, enciclopedie, ambienti virtuali);
- sistemi per il controllo di processi, per esempio in stabilimenti di produzione, raffinerie, impianti per la produzione di energia;
- sistemi *embedded*, cioè insiemi di processori e relativo *software* integrati per il controllo e la gestione di apparecchiature di varia complessità come per esempio: insiemi di elettrodomestici collegati (e "intelligenti"), grosse macchine utensili e soprattutto sistemi militari,
- sistemi per la simulazione di fenomeni complessi (previsioni meteorologiche, idrodinamica di bacini complessi, sistemi cosmologici, ecc.),
- sistemi operativi, *software* di base, pacchetti applicativi di larga diffusione.

Anche se spesso i progetti di produzione di *software* in questi ambiti possono superare anche di due ordini di grandezza i limiti economici del nostro contesto, tuttavia vengono qui presi in considerazione perché esistono comunque esempi (rari) che rientrano nei limiti di questa trattazione.

L'estrema innovatività che caratterizza la produzione di *software* in questi ambienti non si riesce ad affrontare neppure con il modello a prototipi sopra visto. Il modello a spirale è stato elaborato agli inizi degli anni 90 da Barry Boehm alla Carnegie Mellon University/Software Engineering Institute<sup>1</sup>; esso è una evoluzione del modello a prototipi che viene disciplinato e articolato in quattro fasi che si ripetono ciclicamente:

- a) (ri)definizione degli obiettivi, delle alternative e dei vincoli;
- b) analisi dei rischi e costruzione di prototipi;
- c) sviluppo e test;
- d) pianificazione del ciclo successivo.

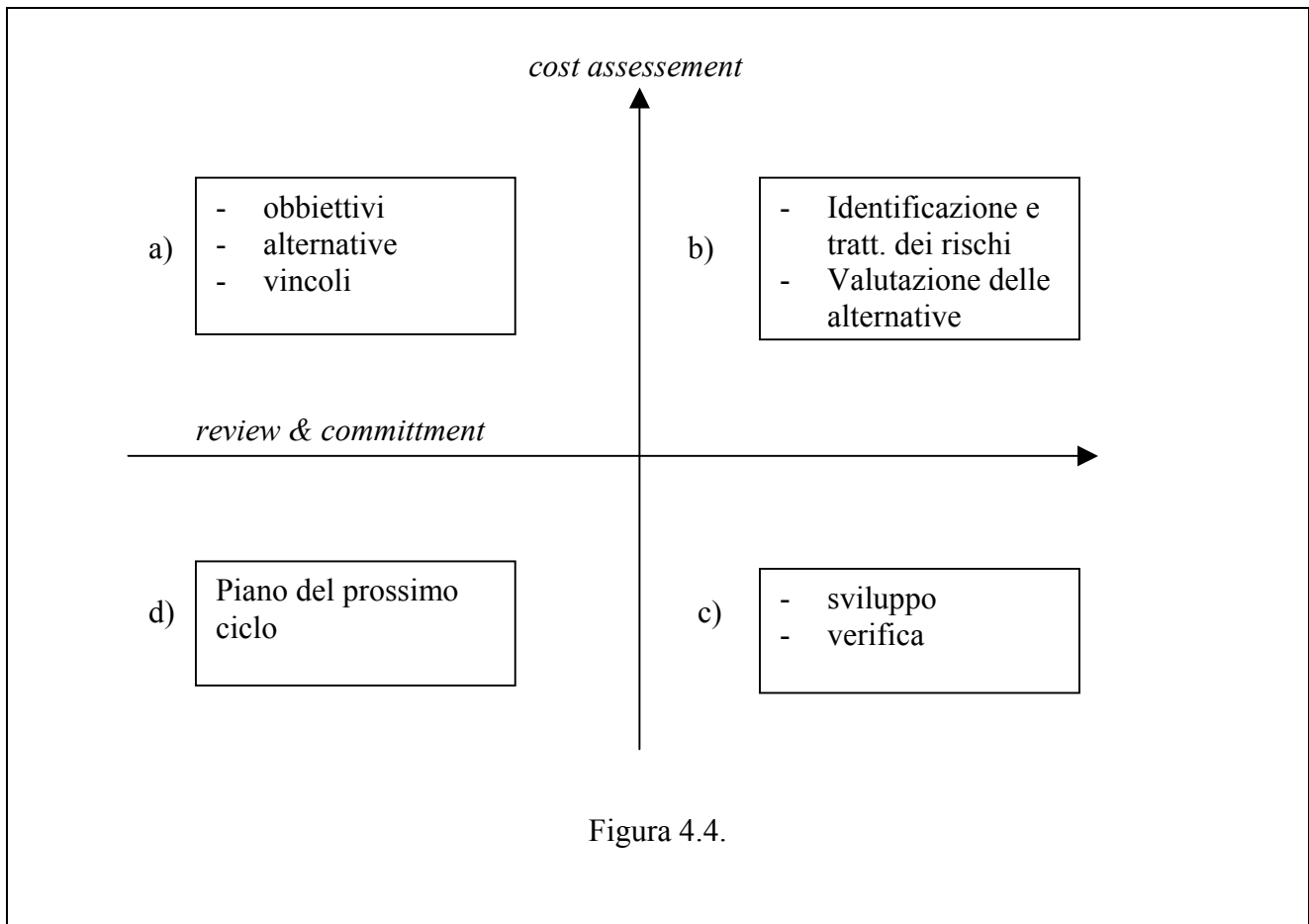


Figura 4.4.

<sup>1</sup> Si veda per esempio il *report* CMU/SEI-2000-SR-008, da cui sono tratte molte delle considerazioni qui espresse, e la bibliografia ivi citata (il *report* è disponibile sul sito Internet della SEI).

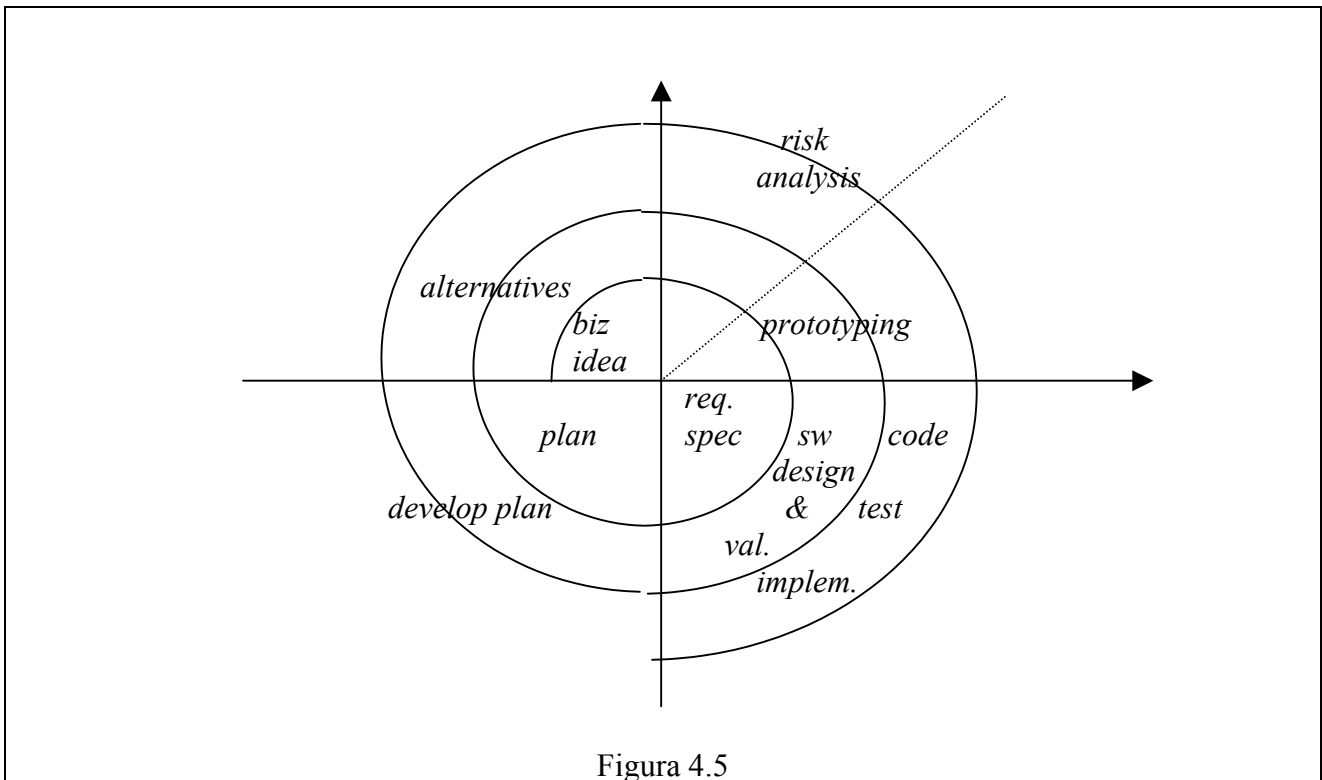


Figura 4.5

Il modello è illustrato dagli schemi delle figure 4.4 e 4.5

Il modello a spirale è in realtà un generatore di modelli di progetto *risk driven* che si adattano progressivamente all'ambiente in cui si opera con due fondamentali caratteristiche:

- approccio ciclico, per aumentare progressivamente il grado di definizione e di implementazione dei prodotti e diminuire il grado di rischio;
- *milestone*, per assicurare il coinvolgimento degli attori in soluzioni fattibili e soddisfacenti.

Occorre tenere presente che questo modello non è un modello a cascata acciambellato (cioè portato avanti per ripetizioni cicliche), ma incorpora l'idea che tutti gli oggetti e i prodotti importanti del progetto sono considerati in parallelo (e non sequenzialmente) e ad ogni giro aumentano via via la chiarezza, la definizione e la parte realizzata di essi.

Valgono le seguenti considerazioni (interpretabili come invarianti del ciclo).

- **Spostare le decisioni avanti nel tempo, il più possibile (compatibilmente con i vincoli del progetto).** Poiché gli oggetti chiave (l'idea di *business*, i *requirements*, i piani, l'architettura e il *design*, gli algoritmi, *COTS component*, ecc.) sono presi in considerazione in parallelo, si evita un *commitment* prematuro, cioè una decisione impegnativa troppo anticipata, su argomenti fondamentali come per esempio la piattaforma *hardware*, i tempi di risposta, ecc. Un esempio famoso in letteratura, di sistema che realizzato con questo modello avrebbe fatto risparmiare in tempo e costi, è quello cui fa riferimento la figura 4.6.

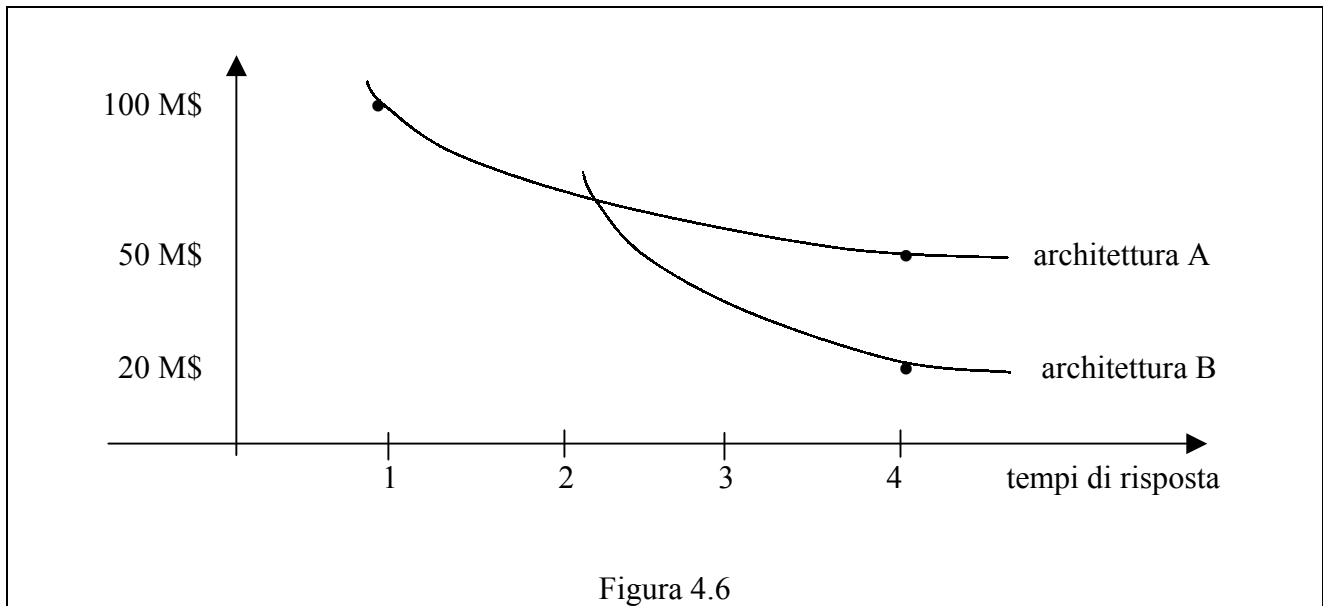


Figura 4.6

Una organizzazione governativa americana doveva sviluppare un sistema *on-line* con oltre un migliaio di utenti di cui parecchie centinaia potevano essere attivi contemporaneamente e che richiedeva una pesante elaborazione di dati *on-line*. Sul mercato, adatte all'esercizio di quel sistema, erano presenti due architetture (vedi fig 4.6); solo l'architettura A era capace anche di tempi di risposta inferiore a due secondi. L'ipotesi iniziale del progetto era che i tempi di risposta fossero compresi fra uno e due secondi, per cui fu necessariamente scelta l'architettura A, relativamente costosa ma l'unica che, con una opportuna configurazione, consentiva di rispettare i vincoli sui tempi di risposta; fu iniziata la costruzione del sistema e verso la metà dei lavori (dopo la produzione di vari prototipi esaminati e discussi con gli utenti) ci si accorse che erano più che accettabili tempi di risposta di tre-quattro secondi: questo consentì un considerevole abbassamento dei costi nell'ambito dell'architettura scelta, acquistando un sistema hardware in una configurazione ridotta. Come si vede chiaramente dalla figura, uno spostamento della scelta architeturale in avanti (quando si era constatato che erano sufficienti tempi di risposta più lunghi) avrebbe consentito di adottare l'architettura B meno costosa, ma comunque adeguata per i nuovi vincoli, riducendo i costi di un ulteriore 60% senza influire in modo essenziale sui tempi di realizzazione del progetto.

- **Tutte le competenze e gli interessi devono essere coinvolte continuamente.** È stato osservato che nei progetti sequenziali sono necessarie in ogni fase solo determinate competenze. In realtà questo a rigore non è completamente vero perché, in ogni fase, è comunque sempre necessaria la partecipazione di tutte le "competenze"; alcune sono presenti in modo determinante e diretto, altre in maniera solo marginale: queste ultime di norma sono surrogate dalla cultura degli effettivi partecipanti. Nei progetti in cui è necessario adottare il modello a spirale, la presenza delle competenze in maniera indiretta non è accettabile. A titolo di esempio, si consideri la seguente tabella.

Fasi tradizionali di un progetto software	Attori previsti nei progetti sequenziali	Altri attori necessari nei progetti a spirale
Requisiti degli utenti	Committente, utenti, analisti	Sviluppatori
Sviluppo e test	Committente e sviluppatori	Utenti e manutentori
Messa in esercizio	Sviluppatori e utenti	Committente e manutentori

Nello stendere i requisiti degli utenti, che in questo tipo di progetti non hanno esperienza pregressa, la mancanza degli sviluppatori porta ad assunzioni non realistiche, con costi eccessivi o al di fuori della possibilità tecnologica. Durante lo sviluppo e il *test*, la mancanza degli utenti, di cui gli sviluppatori non hanno un modello attendibile, può portare alla costruzione di prodotti con cattiva usa-

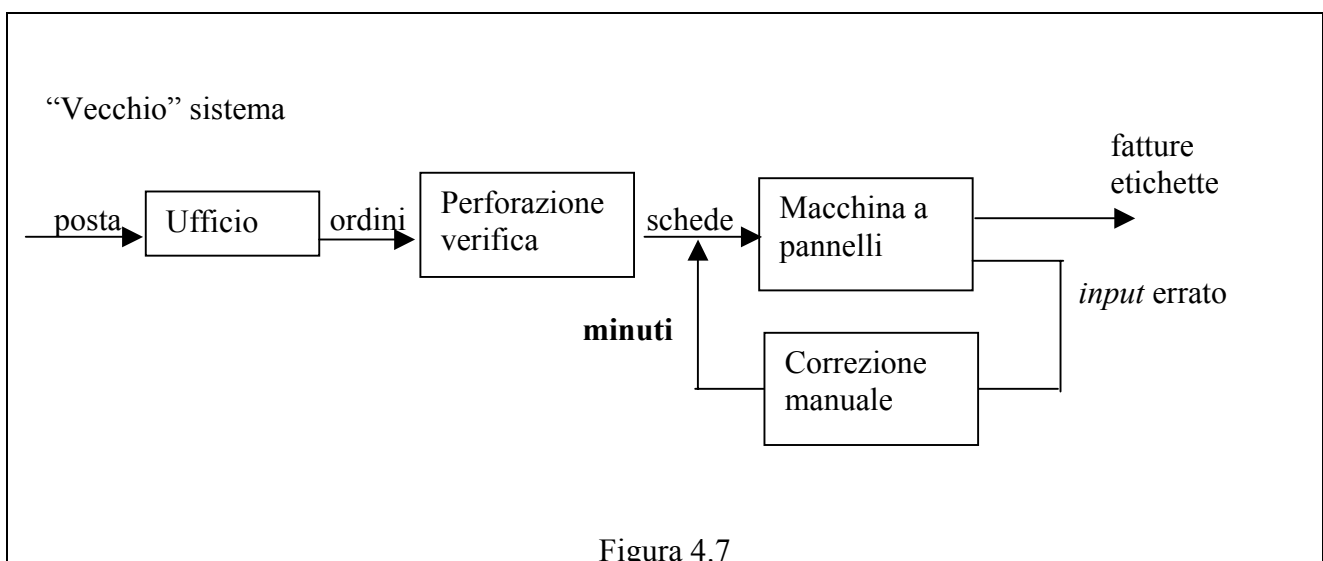
bilità; in questo tipo di progetti inoltre, molto spesso la manutenzione è fatta in ambienti completamente diversi (almeno dal punto di vista della logistica) da quelli dello sviluppo e quindi è essenziale che anche il gruppo incaricato della manutenzione sia presente in questa fase. Analogamente, è necessaria la presenza del committente e dei manutentori durante l'attività di messa in esercizio perché le funzionalità complessive del sistema e la verifica che è stato raggiunto lo scopo per cui è stato approvato il progetto sono complessivamente visibili solo in questa fase.

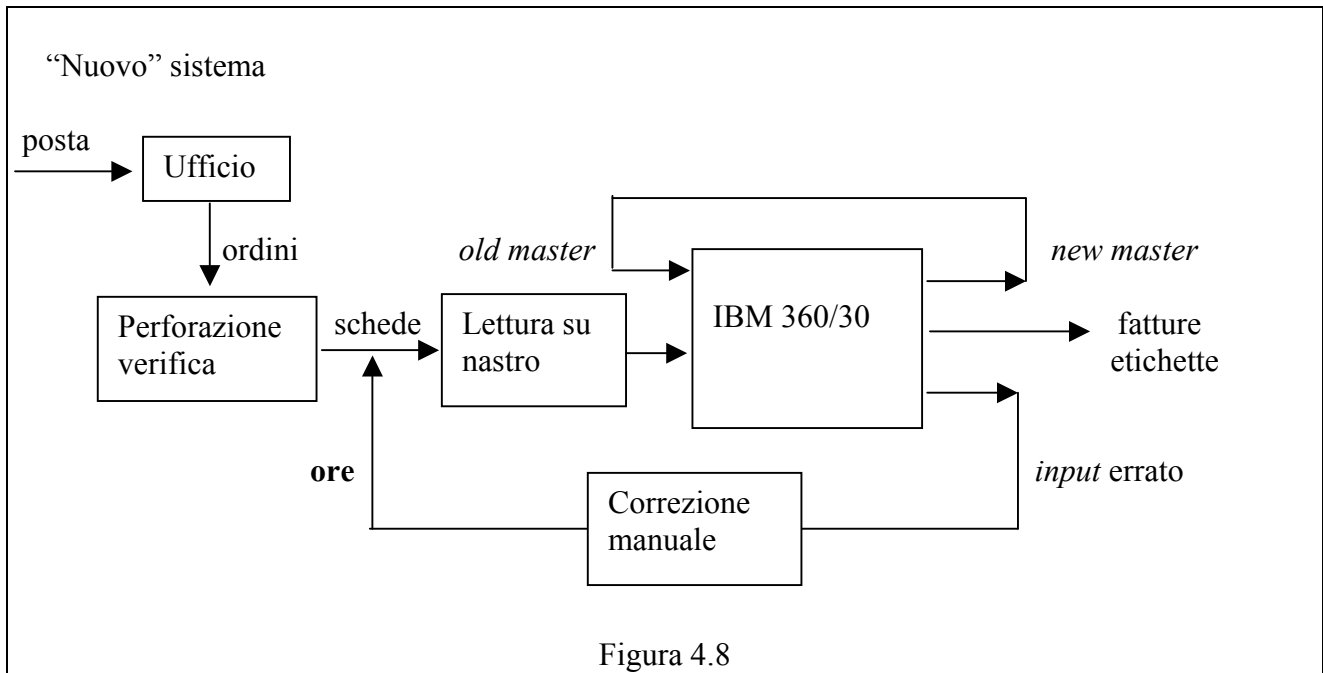
- **La logistica e l'operatività della gestione sono importanti.** In un progetto *software*, in realtà, il *software* e la sottostante architettura *hardware* non sono tutto; occorre prendere in esame anche la logistica, cioè le caratteristiche operative specifiche della gestione. Questo, nel modello in esame, consiste nel considerare (ed attuare) ad ogni ciclo la messa in gestione. A differenza dagli altri modelli (nei quali la messa in gestione viene effettuata solo alla fine), in questo caso è determinante occuparsene fin dal primo ciclo per evitare decisivi e costosissimi errori di architettura. Un famoso esempio che illustra questa problematica è il sistema per gestire gli ordini e gli abbonamenti della rivista *Scientific American*. Alla fine degli anni 60, ordini e abbonamenti venivano ricevuti per posta; le informazioni venivano perforate su scheda e una macchina a pannelli da una parte produceva etichette per la spedizione, fatture per la contabilità ed elenchi per l'amministrazione, e dall'altra produceva delle schede scartate (perché contenenti errori) che venivano verificate e corrette manualmente rispetto agli originali cartacei per essere rimesse nella macchina a pannelli dopo qualche minuto. Questo processo è illustrato in figura 4.7.

Si decise di adottare un calcolatore (*general purpose*, di ultima generazione per quel tempo) per rendere più moderno ed efficiente questo processo. L'architettura è mostrata in figura 4.8.

Il progetto di automazione (gestito essenzialmente con un modello a cascata, di fatto l'unico conosciuto a quei tempi) prevedeva che gli ordini e gli abbonamenti, ricevuti e perforati come in precedenza, fossero letti dal calcolatore e registrati su un *file* (detto di variazioni) residente su nastro magnetico; questo contribuiva con un "*master*" (*file* su nastro contenente tutti gli abbonamenti in vita) a costruire un nuovo *master*, a produrre etichette e fatture (come in precedenza) e un elenco degli errori che venivano confrontati con la documentazione cartacea per produrre le correzioni.

La differenza fondamentale fra la situazione vecchia e la nuova è risultata la seguente: prima il ciclo di correzione durava qualche minuto, con la nuova architettura durava qualche ora! Un progetto analogo, condotto con il modello a spirale, dovendo prendere in considerazione la messa in gestione, e quindi la logistica, fin dall'inizio avrebbe messo in evidenza tempestivamente i difetti della soluzione proposta.





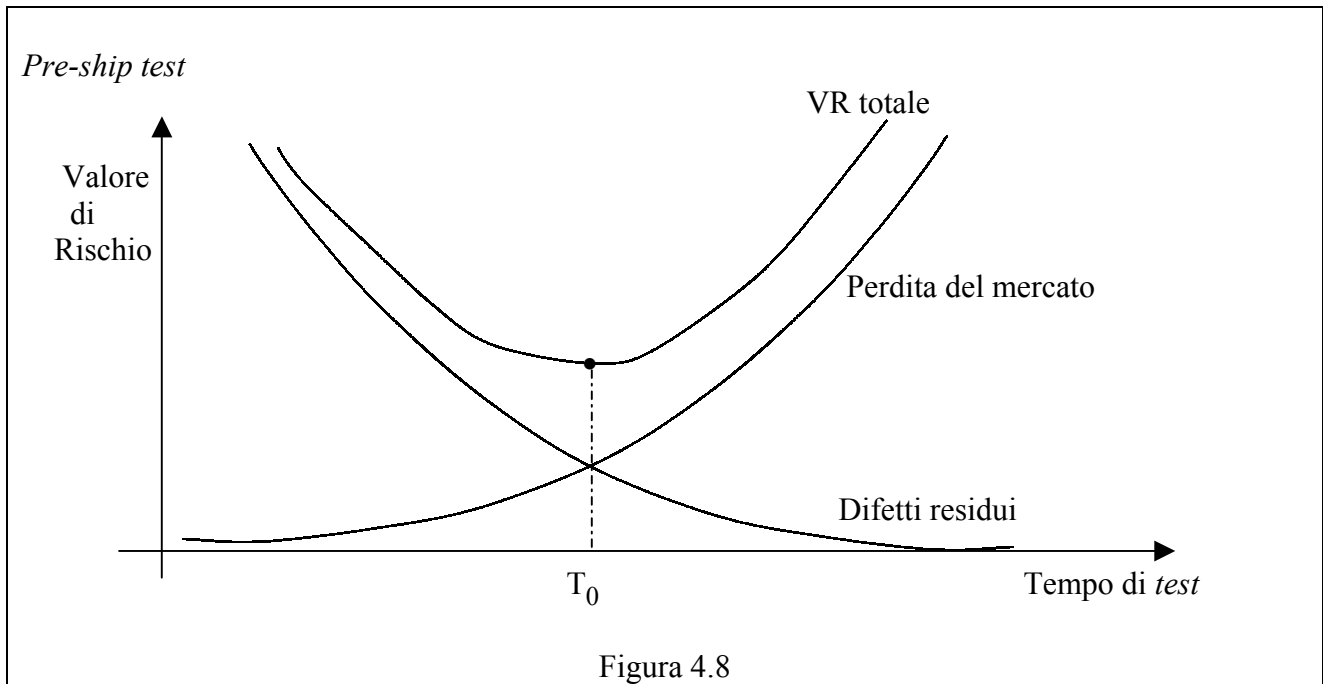
- **I tre milestone fondamentali.** Nei modelli precedenti (si pensi ad esempio al modello a cascata), ogni fase ha termine con l'accettazione esplicita del documento relativo. Nel modello a spirale, in cui tutti i documenti vengono elaborati in parallelo, occorre introdurre una nuova attività da svolgere ad ogni ciclo al fine di garantire la coerenza complessiva delle attività svolte in parallelo rispetto alle esigenze di tutti gli attori. A tal fine vengono considerati, ad ogni ciclo, tre *milestone* consistenti nell'accettazione esplicita, anche se non formale, da parte di tutti gli attori dello stato di avanzamento dei tre seguenti argomenti:

- gli obiettivi, detti LCO (*Life Cycle Objectives*);
- l'architettura della soluzione, detta LCA (*Life Cycle Architecture*);
- le caratteristiche operative, dette IOC (*Initial Operation Capabilities*).

È ovvio che ad ogni ciclo le specifiche dei tre *milestone* diventano sempre più precise.

- **Criterio d'arresto locale e valutazione dei rischi.** Ad ogni ciclo si deve decidere se una certa attività si può considerare di fatto conclusa oppure no. In questo caso il criterio da usare è essenzialmente economico. Per esempio, dovendo descrivere un'interfaccia GUI (*Graphic User Interface*), è inutile richiedere specifiche troppo stringenti perché tipicamente sarà realizzata mediante componenti COTS (*Commercial Off The Shelves*) e quindi occorre attenersi a specifiche standard. Si ricorda che l'impiego di componenti standard anche nella produzione di sistemi *software* sta diventando di uso comune ed è prevedibile una sua diffusione sempre maggiore.

- **Criterio d'arresto globale e valutazione dei rischi.** A ogni ciclo, durante il quale viene sempre svolta una parte di tutte le attività del progetto, occorre usare l'analisi dei rischi per stabilire quando è necessario o opportuno interrompere il ciclo. Potrebbe, infatti, sembrare naturale interrompere il ciclo quando non esiste più alcuna attività da svolgere perché i prodotti non sono più migliorabili. Questo criterio è chiaramente inadeguato come si vede se si considera il modello a spirale applicato alla realizzazione di un prodotto che deve essere messo sul mercato; si devono contemperare due esigenze "opposte": ottenere un buon prodotto e arrivare subito sul mercato (per battere la concorrenza e per recuperare i costi). In questo caso, l'analisi dei rischi, che produce andamenti come quelli descritti in figura 4.8, suggerisce quando è opportuno interrompere il ciclo e considerare terminato il processo.



L'idea di base che sottende al modello a spirale è la stessa che caratterizza i cicli di controllo aziendale (di gestione e strategico): l'articolazione in quattro fasi che si ripetono ciclicamente. Queste fasi per il controllo sono:

- ideazione e pianificazione,
- attuazione,
- rilevazione ed elaborazione dei dati,
- valutazione dei risultati e di possibili alternative.

#### **4.6 I modelli "leggeri"**

#### **4.7 Esercizi**

E.4.1 Con riferimento all'esercizio E2.3, si discuta quale modello di sviluppo del progetto è più adatto a quel contesto.

E.4.1 Con riferimento all'esercizio E2.4, si discuta quale modello di sviluppo del progetto è più adatto a quel contesto.