Indice

2 – Processi software

Un processo software è un insieme di attività che porta alla creazione di un prodotto software. Come è stato ampiamente spiegato nel primo capitolo, non esiste un processo software universalmente applicabile. Il processo utilizzato in aziende differenti dipende dal tipo di software che si sta sviluppando, dalle richieste del cliente e dalle capacità delle persone che scrivono il software.

2.1 – Modelli dei processi software

Un modello di processo software, chiamato anche **modello SDLC** (*Software Development Life Cycle*), è una **rappresentazione semplificata di un processo software**.

Ogni modello rappresenta un processo da una particolare prospettiva, per cui fornisce solo delle informazioni parziale sul processo.

Questi **modelli generici** sono **descrizioni astratte di alto livello dei processi software**, che possono essere utilizzate per spiegare i diversi approcci allo sviluppo del software. Possono essere considerate come strutture di processo da estendere e adattare per creare processi di ingegneria del software più specifici.

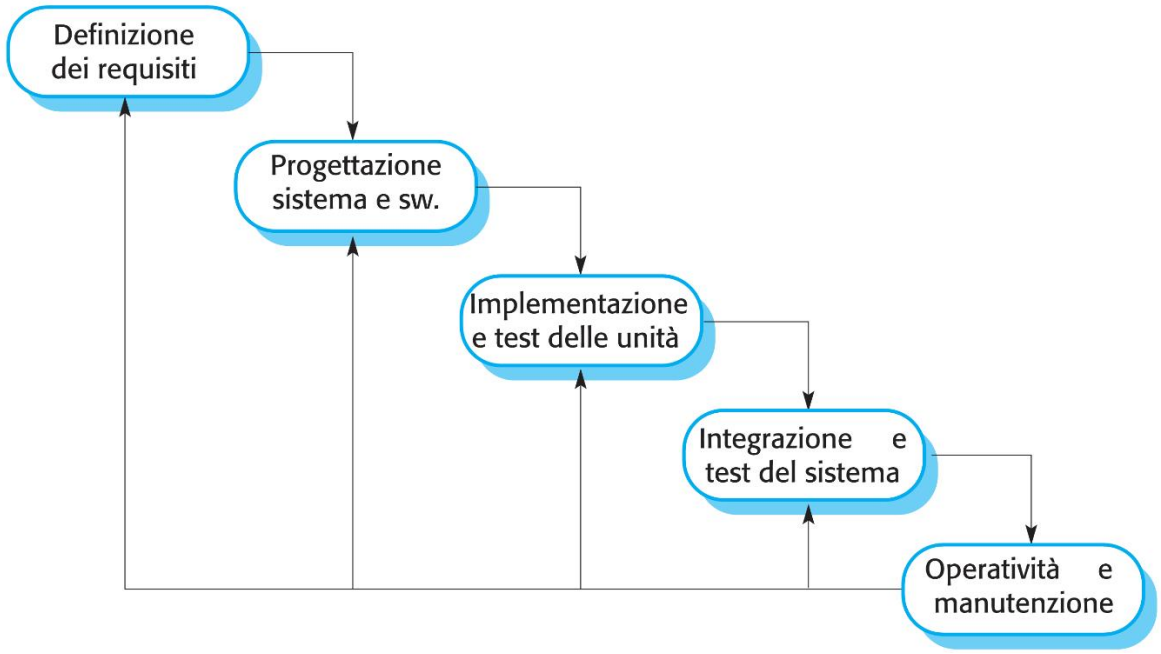
I **modelli di processo generici sono**:

1. **Modello a cascata**. Le attività di processo fondamentali (specifica, sviluppo, convalida ed evoluzione) sono rappresentate come fasi distinte del processo, per esempio la specifica dei requisiti, la progettazione del software, l'implementazione, il test e così via.
2. **Sviluppo incrementale**. Le attività di specifica, sviluppo e convalida sono intrecciate. Il sistema viene sviluppato come una serie di versioni, ciascuna delle quali aggiunge nuove funzionalità alla versione precedente.
3. **Integrazione e configurazione**. Approccio che necessita di una grande disponibilità di componenti o sistemi riutilizzabili. Il processo di sviluppo si basa sulla configurazione di questi componenti per utilizzarli in una nuova disposizione e integrarli in un sistema.

Per esempio, il software a sicurezza critica di solito è sviluppato utilizzando un processo a cascata in quanto sono richieste molte analisi e documentazione prima di iniziare l'implementazione. I prodotti software adesso sono sempre sviluppati utilizzando un modello di processo incrementale. I sistemi aziendali vengono sempre più sviluppati configurando e integrando i sistemi esistenti per creare un nuovo sistema con le funzionalità richieste.

2.1.1 – Modello a cascata

A causa del susseguirsi di una fase dopo l'altra, il modello è conosciuto come modello a cascata o come ciclo di vita del software. Il modello a cascata è un esempio di processo guidato da un piano. Almeno in linea di principio, occorre pianificare tutte le attività di processo prima di iniziare lo sviluppo del software.



I **principali stadi**, come si può vedere dall'immagine, sono 5:

1. **Analisi e definizione dei requisiti**. Vengono determinati i servizi del sistema, i vincoli e gli obbiettivi, attraverso degli incontri con gli utenti del sistema. I requisiti si definiscono in dettaglio e servono come specifica del sistema.
2. **Progettazione del sistema e del software**. Il processo di progettazione del sistema suddivide i requisiti sul sistema hardware o sul software e stabilisce l'architettura generale del sistema.
3. **Implementazione e test delle unità**. Il progetto del software è realizzato come un insieme di programmi o di unità del programma. Il test delle unità verifica che ognuna soddisfi le proprie specifiche.
4. **Integrazione e test del sistema**. Le singole unità di programma o i programmi sono integrati e testati come un sistema completo per accertare che i requisiti del software siano soddisfatti. Dopo il test finale, il sistema viene consegnato al cliente.
5. **Operatività e manutenzione**. Questa è la fase più lunga dell'intero ciclo di vita. Il sistema è installato e messo in opera, si correggono gli errori non scoperti nei primi stadi, si migliora l'implementazione delle unità di sistema e si incrementano i servizi del sistema.

In linea teorica, qualsiasi face successiva non dovrebbe mai partire prima che sia finita quella precedente. Effettivamente, questa regola viene applicata nello **sviluppo hardware** poiché i costi di fabbricazione sono elevati.

Al contrario, nello **sviluppo software** questa definizione non viene rispettata poiché questi stadi si sovrappongono e si scambiano informazioni tra di loro. Durante la progettazione vengono

identificati problemi e requisiti, durante la codifica vengono individuati i problemi di progettazione e così via.

Data la difficoltà di applicare questo modello allo sviluppo software, esistono solo alcuni tipi di **sistemi che possono avere tale modello**:

- I **sistemi integrati**, in cui il software deve interfacciarsi con sistemi hardware. Per cui, a causa della *inflessibilità* dell'hardware, non è possibile rinviare le decisioni sulle funzionalità del software, finché esso è in fase di implementazione.
- I **sistemi critici**, in cui è presente un'analisi approfondita della sicurezza e della protezione del software. Ma per effettuare tali analisi, i documenti della specifica e del progetto devono essere completi.
- I **grandi sistemi software** che fanno parte di sistemi più complessi sviluppati da più società. In questo caso, l'hardware può essere sviluppato con un modello comune a quello software poiché le società lo trovano più semplice.

Il modello a cascata **non è appropriato** in quei casi in cui è necessaria una comunicazione informale nei team e quando i requisiti del software cambiano rapidamente.

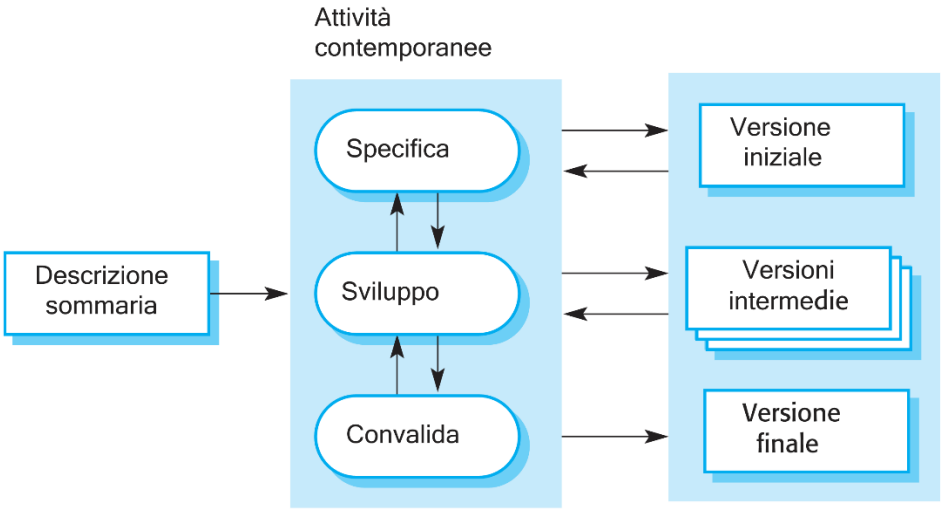
È importante citare una **variante del modello a cascata**, ovvero lo sviluppo di sistemi formali dove viene creato un modello matematico della specifica del sistema. Successivamente, il modello viene perfezionato in un codice eseguibile, utilizzando trasformazioni matematiche che preservano la coerenza. Questa variante viene **utilizzata** nello sviluppo di sistemi software che hanno requisiti severi di sicurezza, affidabilità e protezione.

Tuttavia, nonostante la doverosa menzione, a **causa** degli **alti costi di sviluppo di una specifica formale**, questo modello di sviluppo viene raramente utilizzato, tranne nell'ingegnerizzazione dei sistemi critici.

2.1.2 – Sviluppo incrementale

Lo sviluppo incrementale consiste in:

1. **Sviluppare un'implementazione iniziale**;
2. **Esporla agli utenti**;
3. **Perfezionarla attraverso molte versioni finché non si ottiene il sistema richiesto**.



A differenza del modello a cascata, nello sviluppo incrementale le attività di specifica, sviluppo e convalida sono intrecciate con feedback veloci tra le varie attività.

Lo **sviluppo incrementale** è l'approccio **più comune** per **sviluppare sistemi di applicazioni e prodotti software**.

L'approccio può essere di tre **tipi**:

- *plan-driven*, gli incrementi del sistema sono stabili in anticipo.
- *agile*, vengono identificati gli incrementi iniziali, ma lo sviluppo dei successivi incrementi dipende dall'avanzamento del lavoro e dalle priorità del cliente.
- *ibrido*, ovvero una combinazione dei due tipi precedenti.

Lo **sviluppo incrementale** del software è **migliore dell**'approccio a **cascata per** quei **sistemi** i cui requisiti è probabile **che cambino durante il processo di sviluppo**. Per esempio, molti sistemi aziendali e prodotti software.

Un altro **vantaggio** dello sviluppo software in modo incrementale è la facilità di apportare modifiche al software durante lo sviluppo e l'economicità.

Ogni incremento apporta nuove funzionalità richieste dal cliente. Nelle fasi iniziali gli incrementi sono le richieste più importanti e urgenti del cliente. Grazie a questo approccio, il cliente può fornire una valutazione del sistema fin da subito così da richiedere modifiche che verranno apportate negli incrementi successivi.

Quindi, in conclusione i **pregi** dello **sviluppo incrementale** sono 3:

1. Il **costo di implementazione** delle **modifiche** dei requisiti **è ridotto**. L'analisi e la documentazione che devono essere ripetute ad ogni modifica sono molto minori rispetto al modello a cascata.
2. È **più facile ottenere valutazioni del cliente** sul lavoro di sviluppo che è stato fatto. In un modello a cascata è difficile poiché un cliente deve giudicare l'avanzamento del lavoro dai documenti della progettazione del software.
3. È possibile **consegnare in anticipo una versione utilizzabile** del software, nonostante la mancanza di alcune funzionalità. I clienti sono in grado di utilizzare e valutare il software prima di quanto sia possibile con un processo a cascata.

Mentre i **difetti** o **problemi** dell'approccio incrementale sono principalmente 2:

1. Il **processo non è visibile**. Inoltre, i manager devono avere **consegne regolari** per constatare i progressi. Nel caso in cui il sistema venga **sviluppato velocemente**, lo sviluppo incrementale **non** è più **economico** data l'elevata produzione di documentazione per ogni versione prodotta.
2. La **struttura** tende a **degradarsi ogni** qualvolta vengono aggiunti **nuovi incrementi**. Ogni volta che viene aggiunta una nuova funzionalità, il **codice** diventa **sempre più complicato** e ne consegue una **difficoltà** (e un **costo**) **maggiore** nell'aggiungere nuove funzionalità al sistema. Tuttavia, questo **problema** viene **parzialmente arginato** tramite un **approccio** di tipo ***agile*** poiché viene effettuato un costante "$refactoring$", ovvero un miglioramento della struttura, una sorta di ristrutturazione del software.

Un'ultima osservazione da fare sono i **problemi** relativi allo **sviluppo di grandi sistemi** tramite lo sviluppo incrementale. In questo caso, tali sistemi richiedono contesti o architetture stabili e i vari team che lavorano su parti differenti del sistema devono avere obbiettivi definiti in modo chiaro. Tutto ciò deve essere pianificato in anticipo, quindi il contrario dello sviluppo incrementale.

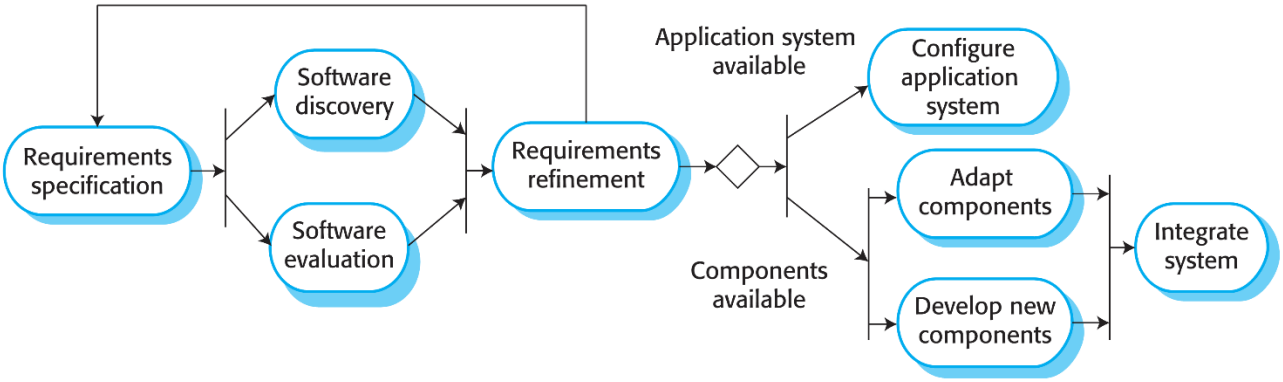
2.1.3 – Integrazione e configurazione

Nella maggior parte dei progetti software si è soliti riutilizzare il software. L'approccio orientato al riutilizzo si fonda su una larga base di componenti software riutilizzabili e su un framework di integrazione per comporre questi componenti.

Esistono **tre tipi di componenti** frequentemente utilizzati:

1. **Sistemi di applicazioni indipendenti** che sono configurati per essere utilizzati in un particolare ambiente.
2. **Collezioni di oggetti** che sono sviluppate come un componente o un pacchetto da integrare tramite un framework di integrazione dei componenti, come Java Spring.
3. **Servizi web** che sono sviluppati in conformità agli standard e che sono disponibili per essere utilizzati da siti remoti su Internet.

La figura mostra un modello di processo generico per lo sviluppo basato sul riutilizzo tramite l'integrazione e la configurazione dei componenti.



Le **fasi** del processo sono:

1. **Specifica dei requisiti** (*Requirements specification*). Vengono definiti i requisiti iniziali del sistema tramite una breve descrizione dei requisiti e delle funzionalità essenziali del sistema.
2. **Ricerca e valutazione del software** (*Software discovery and evaluation*). Vengono ricercati i componenti e i sistemi che possono fornire le funzionalità richieste.
3. **Perfezionamento dei requisiti** (*Requirements refinement*). I requisiti vengono perfezionati e modificati in funzione dei componenti disponibili, mentre la specifica del sistema viene ridefinita. Se le modifiche non sono disponibili, l'attività di analisi dei componenti (punto 1) può essere ripetuta per cercare soluzioni alternative.
4. **Configurazione del sistema delle applicazioni** (*Configure application system*). Se disponibile, viene riutilizzato un sistema di applicazioni pronto all'uso che soddisfa i requisiti, ovviamente può essere configurato per creare il nuovo sistema.
5. **Adattamento e integrazione dei componenti** (*Adapt components*, *Develop new components*, *Integrate system*). Se non disponibile un sistema di applicazioni (punto 4), i singoli componenti riutilizzabili possono essere modificati per sviluppare i nuovi componenti, che vengono poi integrati per creare il sistema.

I **vantaggi**, legati tra di loro, di questo sistema sono molteplici:

- **Riduzione significativa** della quantità di software da sviluppare (vantaggio principale);
- **Riduzione** dei costi;
- **Riduzione** dei rischi;
- Consegne **più veloci**.

Purtroppo, i problemi (**svantaggi**), anch'essi legati tra di loro, che crea l'adozione di questo sistema non sono da sottovalutare:

- **Compromessi** nei requisiti;
- Il sistema **non soddisfa** le reali **necessità** degli utenti;
- **Perdita** del **controllo** sull'evoluzione del sistema poiché le nuove versioni dei componenti riutilizzati non sono sotto il controllo dell'organizzazione che li usa.

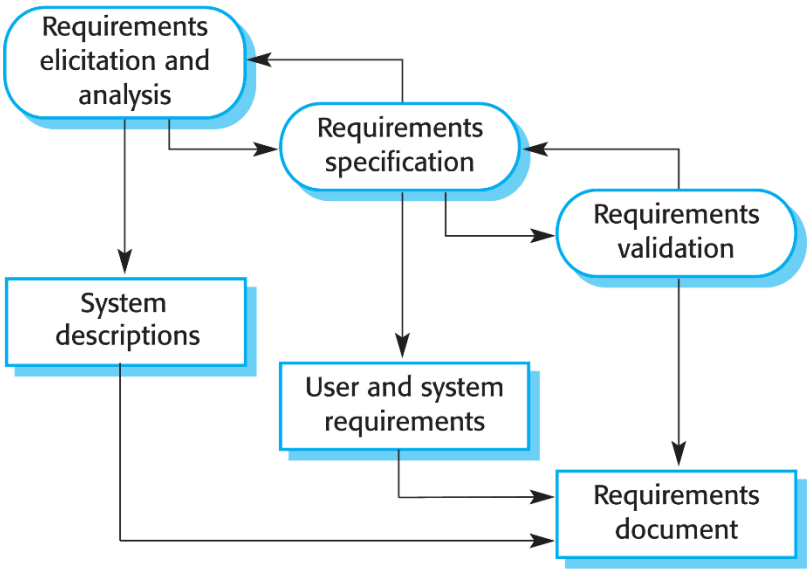
2.2.1 – Specifica del software

La creazione delle **specifiche del software**, chiamata ingegneria dei requisiti, è il processo per capire e definire quali servizi sono richiesti dal sistema, e per identificare i vincoli all'operatività e allo sviluppo del sistema.

Questo stadio è particolarmente **critico** nel processo software, poiché gli errori in questa fase portano inevitabilmente a problemi successivi nella progettazione e implementazione del sistema.

Prima dell'avvio di questo processo, talvolta le società effettuano uno studio di fattibilità o di mercato per stabilire se esiste un mercato per il software e se sia tecnicamente ed economicamente realistico sviluppare il software richiesto. Uno studio di fattibilità dovrebbe essere **rapido** e **poco costoso**, inoltre il risultato dovrebbe permettere di decidere se continuare o meno con un'analisi più dettagliata.

Le **fasi dell'ingegneria dei requisiti** sono mostrate in figura. Questo processo permette di produrre un documento di caratteristiche concordate che specifica un sistema che soddisfa i requisiti degli stakeholder. I requisiti vengono presentati in modo diverso: agli utenti finali e clienti, vengono presentati i requisiti di alto livello, mentre agli sviluppatori i requisiti con una specifica più dettagliata.



Le **fasi** principali del **processo** di ingegneria dei requisiti:

1. **Deduzione e analisi dei requisiti** (*Requirements elicitation and analysis*). Viene fatta la deduzione dei requisiti grazie all'osservazione di sistemi già esistenti, discussione con possibili utenti e acquirenti, l'analisi dei task e via dicendo. Può coinvolgere lo sviluppo di uno o più modelli e prototipi, che aiutano gli analisti a capire il sistema da specificare.

2. **Specifica dei requisiti** (*Requirements specification*). Vengono tradotte le informazioni raccolte durante la fase di analisi in un documento che definisce un insieme di requisiti. Il documento può includere due tipi di requisiti: i requisiti utente che sono proposizioni astratte e i requisiti di sistema che sono una descrizione più dettagliata.

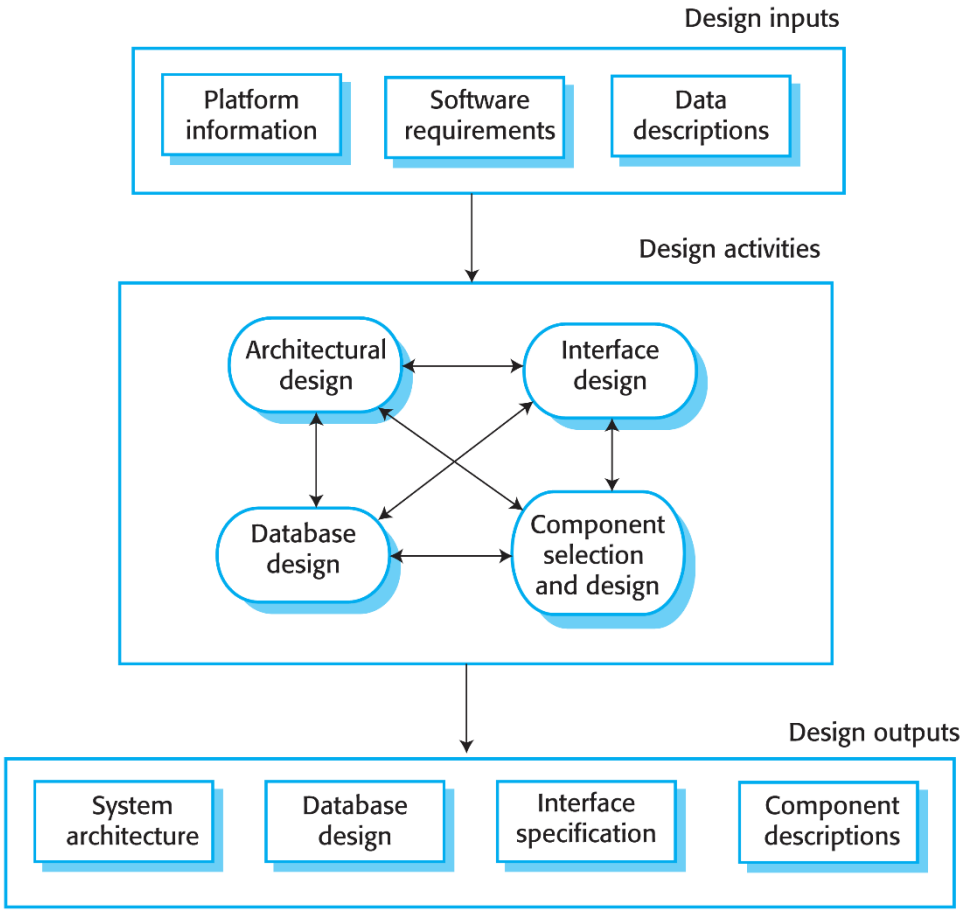
3. **Convalida dei requisiti** (*Requirements validation*). Vengono controllati i requisiti stabilendo se sono realistici, coerenti e completi.

Nonostante l'esistenza della terza fase, l'analisi dei requisiti è presente anche durante le altre attività di processo quali la "specifica software" e lo "sviluppo software". Per questo motivo le attività di analisi, definizione e specifica sono intrecciate.

2.2.2 – Progettazione e implementazione del software

L'**implementazione** dello sviluppo software è il processo di conversione delle specifiche in un sistema eseguibile da consegnare al cliente.

Il progetto del software è una descrizione della struttura del software che si deve implementare, dei modelli e delle strutture di dati usati dal sistema, delle interfacce tra i componenti del sistema e, a volte, tra gli algoritmi usati.



La figura mostra un modello astratto del processo di progettazione e mostra gli input, le attività e gli output del processo. Le attività sono sia intrecciate sia interdipendenti. Nuove informazioni vengono costantemente generate, e questo influisce sulle precedenti scelte; quindi, è inevitabile la revisione del progetto.

Molte interfacce software si interfacciano con altri sistemi software. Quest'ultimi includono il sistema operativo, il database, il middleware e altri sistemi di applicazioni. Tutto questo costituisce la "**piattaforma software**", l'ambiente in cui il software sarà eseguito. Le **informazioni su questa piattaforma sono l'input essenziale**, poiché i progettisti devono decidere come integrare meglio i loro prodotti con l'ambiente operativo.

Le **attività nel processo di progettazione** variano, in funzione del tipo di sistema che si sta sviluppando. Tuttavia, ci sono quattro attività che fanno parte del processo di progettazione:

1. **Progettazione dell'architettura** (*Architectural design*). Identificazione della struttura complessiva del sistema, dei componenti principali, delle loro relazioni e della loro distribuzione.
2. **Progettazione del database** (*Database design*). Progettazione delle strutture dati del sistema e la loro rappresentazione in un database.
3. **Progettazione dell'interfaccia** (*Interface design*). Definita l'interfaccia tra i componenti del sistema. Non deve essere ambigua, può essere usato da altri componenti senza conoscerne la sua implementazione. Una volta approvata la specifica dell'interfaccia, i componenti possono essere progettati e sviluppati separatamente.
4. **Progettazione e scelta dei componenti** (*Component selection and design*). Ricercati i componenti riutilizzabili e, se non sono disponibili componenti idonei, vengono progettati nuovi componenti.

Queste attività portano agli **output di progettazione**. Per i sistemi critici, gli output del processo di progettazione sono documenti dettagliati che descrivono in modo accurato il sistema.

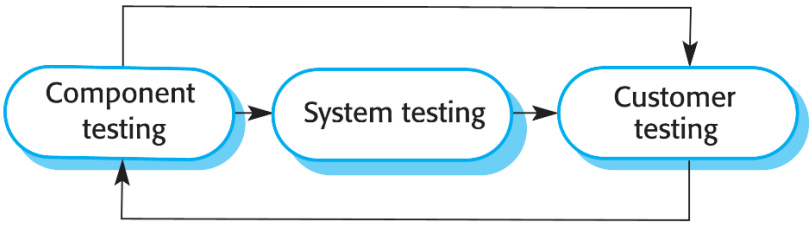
La programmazione è un'attività individuale e non c'è un processo generico che viene seguito abitualmente.

Solitamente i programmatori testano il codice che hanno sviluppato così da mettere in luce difetti che devono essere rimossi; questa procedura è chiamata **debug**. I processi di **test** e di **debug** sono diversi: il primo dimostra l'**esistenza dei difetti**, il secondo si occupa di **localizzarli** e successivamente **correggerli**.

2.2.3 – Convalida del software

La **convalida del software** o "la verifica e la convalida" (dall'inglese V&V, *Verification and Validation*) è intesa a mostrare che un sistema è conforme alle sue specifiche e soddisfa le aspettative del cliente. Il **test dei programmi**, ovvero una simulazione con dei dati di prova, è la **tecnica principale** di convalida del software.

La figura mostra un processo di test a **tre stadi**, dove sono testati prima i singoli componenti del sistema e poi il sistema integrato. Per il software di un cliente, il terzo stadio (*customer testing*) richiede che il sistema sia testato con i dati reali del cliente. Invece, per i prodotti venduti come applicazioni, il terzo stadio, chiamato anche "alpha test", prevede che alcuni utenti selezionati provino e commentino il software.

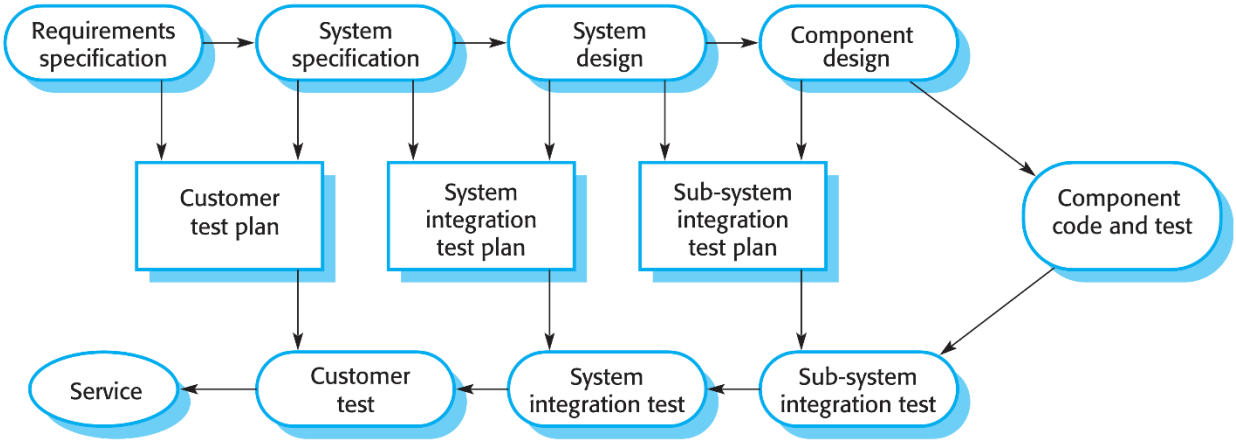


Gli **stadi** del processo sono tre:

1. **Test dei componenti** o delle unità (*Component testing*). I componenti che formano il sistema vengono testati dalle persone che sviluppano il sistema. Ciascun componente è stato testato separatamente.
2. **Test del sistema** (*System testing*). Si integrano i componenti per formare il sistema completo. Questo processo si occupa di trovare gli errori causati da interazioni impreviste tra i componenti e i problemi con le relative interfacce. Per sistemi più grandi, può essere necessario un processo articolato in più stadi.
3. **Test del cliente** (*Customer testing*). È lo stadio finale. Il sistema viene testato dal cliente con i suoi dati, anziché con dati di simulazione. Questo test può rivelare errori e omissioni nella definizione dei requisiti del sistema.

Ovviamente, i difetti scoperti richiedono un debug del programma, e ciò può comportare la ripetizione di altri stadi del processo di test.

Esiste anche un altro processo chiamato: **processo guidato da piani**. In questo caso, il test è guidato da una serie di piani di prova in cui un team indipendente di tester lavora seguendo questi piani di prova.



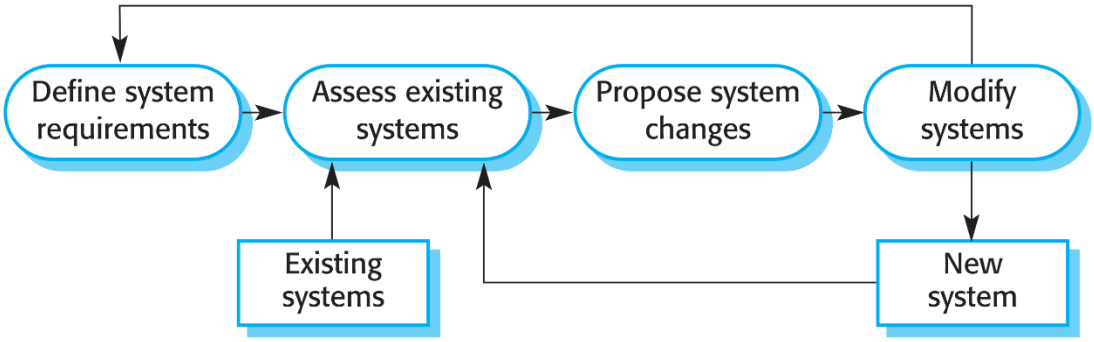
In figura viene mostrato come questi piani sono collegati alle attività di sviluppo e di test. Questo è anche detto modello-V di sviluppo (la lettera V si vede ruotando lateralmente lo schema). Il modello-V mostra le attività di convalida del software che corrispondono a ciascuna fase del modello a cascata.

Quando un sistema viene venduto come prodotto software, si utilizza un processo chiamato "beta test", che comporta la consegna del sistema a una serie di potenziali clienti che accettano di usarlo per un certo periodo e riferiscono i problemi agli sviluppatori.

2.2.4 – Evoluzione del software

Storicamente c'è sempre stata una divisione tra i processi di sviluppo e di evoluzione (o manutenzione) del software.

Questa distinzione sta diventando sempre più irrilevante. Sono pochi i sistemi software completamente nuovi ed è quindi più sensato vedere lo **sviluppo e la manutenzione come un tutt'uno**. È più realistico considerare l'ingegneria del software come un unico processo evolutivo (come in figura), in cui il software viene modificato continuamente nel corso della sua vita per adeguarlo ai cambiamenti dei requisiti e delle necessità dei clienti.



2.3 – Far fronte ai cambiamenti

I cambiamenti sono inevitabili in tutti i grandi progetti software. Cambiano i progetti e le implementazioni con l'arrivo di nuove tecnologie.

I **cambiamenti aggiungono costi allo sviluppo del software**, perché richiedono che il lavoro svolto deve essere rifatti; tutto questo è chiamato **rilavorazione del progetto**.

La **riduzione** dei **costi** è possibile tramite l'utilizzo di due approcci:

1. **Anticipazione dei cambiamenti**, quando il processo software include attività che possono **anticipare** o **predire** possibili **variazioni**, prima che sia richiesta una significativa rilavorazione.
   Per esempio, si potrebbe sviluppare un prototipo per mostrare alcune caratteristiche chiave del sistema ai clienti, i quali possono provare il prototipo e perfezionare i loro requisiti.

2. **Tolleranza ai cambiamenti**, quando il processo e il software sono progettati in modo che le modifiche possano essere facilmente apportate al sistema.

Si introducono **due metodi** per **far fronte** ai **cambiamenti** dei requisiti del sistema. È solo una breve introduzione, le specifiche più dettagliate verranno fornite nei paragrafi successivi:

1. **Prototipazione del sistema**. Una versione del sistema o una sua parte vengono sviluppate rapidamente per verificare i requisiti del cliente e la fattibilità delle scelte di progettazione. Questo è un metodo di anticipazione dei cambiamenti, perché consente agli utenti di provare il sistema prima della consegna e di perfezionare i loro requisiti. Il **numero** delle **proposte di modifica dei requisiti** fatte dopo la consegna è molto **probabile** che **si riduca**.

2. **Consegna incrementale**. Gli incrementi del sistema vengono consegnati al cliente per essere commentati e provati. Quindi, viene evitato l'approvazione prematura dei requisiti per l'intero sistema e consente alle **modifiche** di essere incluse in successivi incrementi a **costi relativamente bassi**.

Il refactoring è un altro importante meccanismo che supporta la tolleranza ai cambiamenti.

2.3.1 – Prototipazione

Un **prototipo** è la versione iniziale di un sistema software che viene utilizzata per dimostrare i concetti, provare opzioni di progettazione e scoprire più informazioni sui problemi e sulle possibili soluzioni.

Lo sviluppo in modo **rapido** e **iterativo** è **essenziale** per tenere sotto controllo i costi e permettere agli stakeholder del sistema di sperimentarlo nelle fasi iniziali del processo software.

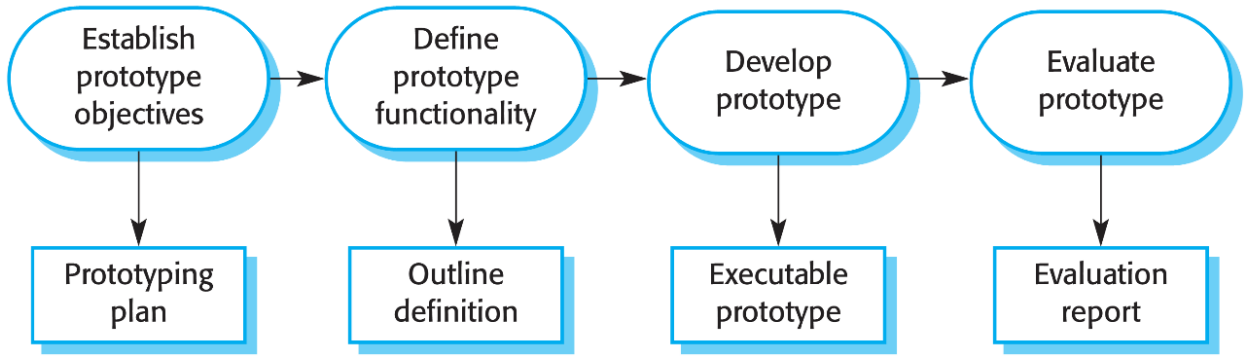
Un prototipo viene utilizzato come aiuto per anticipare le modifiche che potrebbero essere richieste:

1. Nell'ingegneria dei requisiti, un prototipo può aiutare a identificare e convalidare i requisiti del sistema.
2. Nel processo di progettazione del sistema, un prototipo può essere utilizzato per esplorare particolari soluzioni software e durante lo sviluppo di un'interfaccia utente per il sistema.

I prototipi permettono agli **utenti** di **suggerire** nuove **idee** per i requisiti e **facilitare** la **scoperta** dei **punti di forza** e di **debolezza** del software. Inoltre, possono emergere gli errori e le omissioni nei requisiti proposti.

Un prototipo può essere **utilizzato** mentre si **sta sviluppando** il sistema per **eseguire esperimenti** di progettazione e per **verificare** la **fattibilità** di un progetto proposto. Per questo motivo, la prototipazione rapida, con il coinvolgimento degli utenti finali, è l'unico modo sicuro per sviluppare interfacce grafiche utente.

Un **modello di processo per lo sviluppo di prototipi** è mostrato in figura.



Gli **obbiettivi** della prototipazione dovrebbero essere resi espliciti all'inizio del processo, ovvero il **primo passo** del modello (*Establish prototype objectives*).

Il **secondo passo** (*Define prototype functionality*) è decidere cosa inserire e cosa lasciar fuori dal prototipo del sistema. Per ridurre i costi di prototipazione e accelerare i tempi di consegna, si possono trascurare alcune funzionalità, per esempio i requisiti non funzionali, come i tempi di risposta e l'utilizzazione della memoria. Possono essere ridotti gli standard di affidabilità e qualità del programma a beneficio della velocità di produzione.

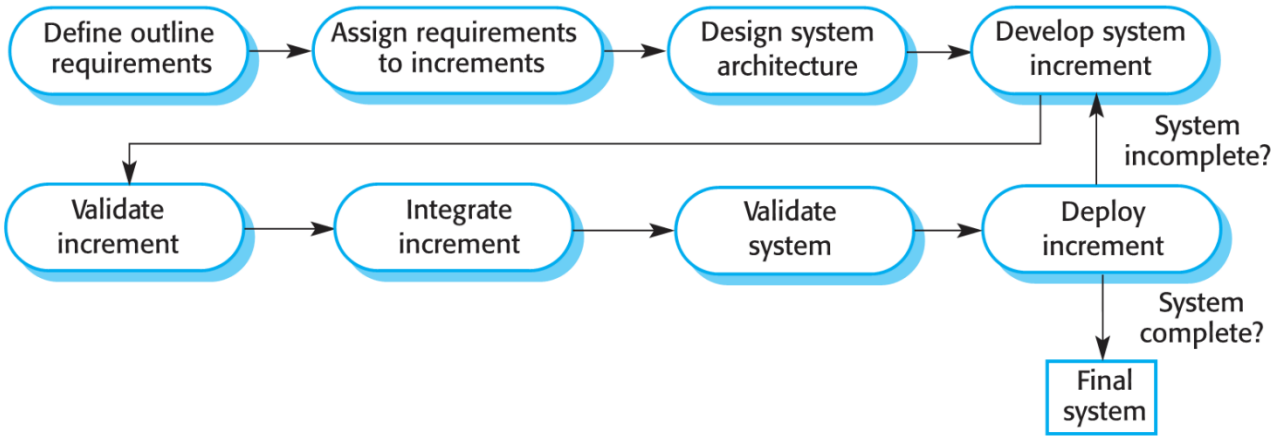
Il **terzo passo** (*Develop prototype*) riguarda lo sviluppo del prototipo che è pura programmazione.

Il **quarto** e ultimo passo (*Evaluate prototype*) è la **valutazione** del prototipo. I potenziali utenti hanno bisogno di tempo per trovarsi a loro agio con il nuovo sistema e per abituarsi a un normale schema di utilizzo. Una volta che utilizzano il sistema normalmente, gli utenti possono scoprire errori e omissioni nei requisiti.

Un **problema generale** della prototipazione è che gli utenti non possono usare il prototipo nello stesso modo in cui usano il sistema finale. Chi prova il prototipo potrebbe non essere un tipico utente del sistema. Il periodo di formazione degli utenti durante la valutazione del prototipo potrebbe essere insufficiente. Se il prototipo fosse lento, gli utenti potrebbero correggere il loro modo di lavorare ed evitare quelle funzionalità del sistema che hanno tempi di risposta lenti.

2.3.2 – Consegna incrementale

La **consegna incrementale** è un approccio allo sviluppo del software dove alcuni degli incrementi sviluppati sono consegnati al cliente e installati per essere usati nel loro ambiente di lavoro. Quindi, il cliente definisce i servizi più e meno importanti che il sistema deve fornire. I servizi richiesti vengono implementati tramite incrementi che hanno diverse priorità.



Dopo aver identificato gli incrementi di sistema, vengono definiti i requisiti del primo incremento, il quale viene subito sviluppato.

Durante lo sviluppo, viene effettuata l'analisi dei requisiti per gli incrementi successivi e non vengono accettati cambiamenti per i requisiti dell'incremento in corso di sviluppo.

Una volta effettuato e consegnato l'incremento, può essere installato nell'ambiente usuale di lavoro del cliente; così facendo, il cliente può provarlo nel sistema e definire i requisiti per i successivi incrementi. Quando i nuovi incrementi sono completati vengono integrati con quelli esistenti in modo da migliorare le funzionalità del sistema ad ogni consegna.

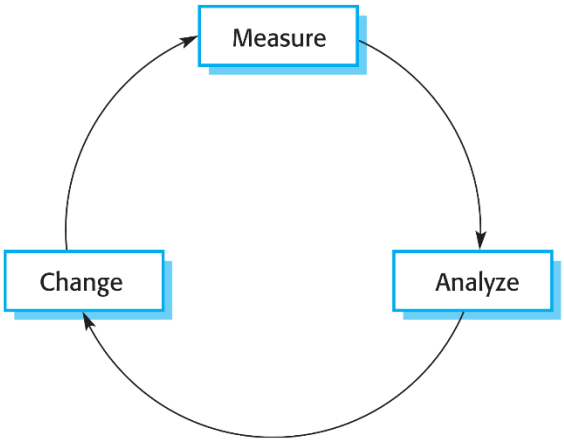
2.4 – Miglioramento dei processi

Miglioramento dei processi significa comprendere i processi esistenti e modificarli per aumentare la qualità dei prodotti e/o diminuire i costi e i tempi di sviluppo.

Per cambiare e migliorare i processi si usano due differenti approcci:

1. **Approccio di maturità del processo**. Si tratta di migliorare la gestione dei processi e dei progetti, e di introdurre buone pratiche di ingegneria del software nelle organizzazioni. Il livello di maturità dei processi evidenzia le buone pratiche tecniche e gestionali che sono state adottate nei processi organizzativi di sviluppo del software. Gli **obbiettivi** principali di questo approccio sono migliorare la qualità del prodotto e la prevedibilità dei processi.
2. **Approccio agile**. Si tratta di uno sviluppo iterativo e di una riduzione degli overhead nei processi software. Le caratteristiche principali sono una rapida consegna di funzionalità e una prontezza di risposta ai cambiamenti dei requisiti del cliente. La filosofia di questo metodo considera i migliori processi quelli con i più bassi overhead.

Il **processo** generale **di miglioramento** è **un'attività ciclica**:

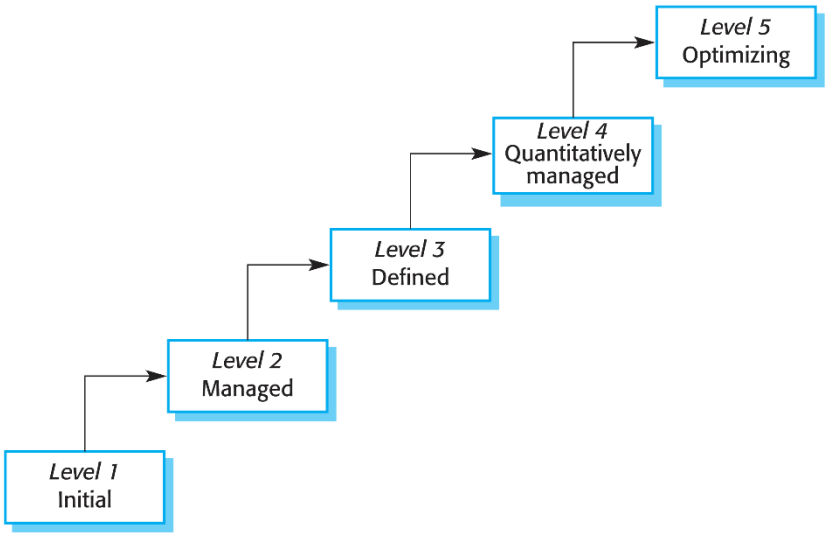


Gli stadi sono:

1. **Misurazione del processo**: si misurano uno o più attributi del prodotto o processo software per decidere se i miglioramenti sono stati efficaci. Ogni volta che si introducono dei miglioramenti, si misurano di nuovo gli stessi attributi, che dovrebbero essere migliori dei precedenti;
2. **Analisi del processo**: si valuta il processo corrente e si identificano i punti deboli e colli di bottiglia. L'analisi si focalizza anche su determinate caratteristiche dei processi come la rapidità e la robustezza;
3. **Modifica del processo**: vengono proposte alcune modifiche del processo per eliminare i punti deboli identificati durante l'analisi. Una volta apportate le modifiche, il ciclo ricomincia, e si raccolgono i dati per valutare l'efficacia delle modifiche.

Come si può osservare, il miglioramento dei processi è un'**attività a lungo termine**; quindi, ogni stadio del processo di miglioramento potrebbe anche durare parecchi mesi. È anche un'attività continua poiché l'ambiente operativo cambia e anche i nuovi processi dovranno evolversi per tenere conto di questi cambiamenti.

I livelli di maturità dei processi, come viene mostrato in figura, sono principalmente cinque:



1. **Iniziale**: gli obbiettivi associati all'area dei processi sono soddisfatti, e per tutti i processi lo scopo del lavoro da svolgere è definito e comunicato ai membri del team.
2. **Gestito**: gli obbiettivi associati all'area dei processi sono soddisfatti, e le politiche organizzative definiscono quando ogni processo deve essere utilizzato. Devono quindi esistere piani di progettazione documentati che definiscono gli obbiettivi dei progetti.
3. **Definito**: avviene la standardizzazione e l'installazione dei processi organizzativi. I giudizi e le misurazioni dei processi devono essere raccolti e utilizzati per i miglioramenti futuri dei processi.
4. **Quantitativamente gestito**: vengono effettuate misurazioni su processi e prodotti per poi essere utilizzate nella gestione di processi.
5. **Ottimizzazione**: è il livello massimo, l'organizzazione deve utilizzare le misurazioni su processi e prodotti per guidare il miglioramento dei processi. Si analizzano i trend, e i processi vengono adatti alle mutate esigenze delle aziende.

Purtroppo, ci sono **troppi overhead** nel miglioramento formale dei processi **per le piccole società** di software, ed è difficile stimare la maturità con i processi agili. Per cui, soltanto le grandi società di software usano questo approccio focalizzato sulla maturità per migliorare i processi software.

Per concludere, i **vantaggi della consegna incrementale** sono:

1. I **clienti** possono **utilizzare** gli **incrementi come prototipi** e **acquisire esperienza** per perfezionare i requisiti degli incrementi successivi. Diversamente dai prototipi, gli incrementi sono parti del sistema reale, quindi non occorre addestrare nuovamente l'utente.
2. I **clienti** traggono **vantaggio** da una **consegna rapida** senza dover aspettare il sistema completo. Il primo incremento soddisfa i loro requisiti più critici, quindi possono utilizzare il software immediatamente.
3. Il **processo** conserva i **benefici dello sviluppo incrementale**, dato che è facile incorporare le modifiche nel sistema.
4. Poiché i servizi con priorità più alta sono consegnati prima e gli incrementi successivi sono incrementati con questi, i **servizi di sistema più importanti** sono **testati più intensamente** degli altri.

Mentre gli **svantaggi della consegna incrementale** sono:

1. La **consegna iterativa è problematica** quando un nuovo sistema deve rimpiazzare un sistema esistente. Gli utenti hanno bisogno di tutte le funzionalità del vecchio sistema e di solito sono riluttanti a sperimentare un sistema completamente nuovo.
2. Spesso i sistemi richiedono funzionalità utilizzate da parti differenti del sistema. Poiché i requisiti non sono definiti nei dettagli finché c'è un incremento da implementare, è **difficile identificare** le **funzionalità comuni** che sono richieste da tutti gli incrementi.
3. L'essenza dei processi iterativi è che la specifica viene sviluppata insieme al software. Tuttavia, questo è in **conflitto con** il **modello di acquisizione** di molte organizzazioni, dove la specifica del sistema completo è parte del contratto. Nell'approccio incrementale, non c'è la specifica del sistema completo finché non viene definito l'ultimo incremento.

Per alcuni tipi di sistemi, lo sviluppo e la consegna incrementali non sono l'approccio migliore. Per esempio, i sistemi molto grandi dove lo sviluppo può richiedere team che lavorano in posti diversi.