

Ingegneria del Software

UniVR - Dipartimento di Informatica

Fabio Irimie

2° Semestre 2024/2025

Indice

1	Introduzione	2
1.1	Cos'è l'ingegneria del software?	2
1.1.1	Economia	2
1.1.2	Fallimento dei progetti	2
1.2	Prodotti software	2
1.2.1	Prodotti generici	2
1.2.2	Prodotti personalizzati	2
1.3	Etica dell'ingegneria del software	2
1.3.1	Codice etico ACM/IEEE	3
1.3.2	Esempi	3
1.3.3	Casi di studio	4
1.4	Caratteristiche di un buon software	6
1.5	Problemi che influenzano il software	6
1.6	Tipi di applicazione	6
1.7	Principi fondamentali	7
2	Processi del software	7
2.1	Modelli di processo	8
2.1.1	Modello a cascata	8
2.1.2	Modello incrementale	9
2.1.3	Modello a integrazione e configurazione	9
2.2	Attività dei processi	10
2.2.1	Specificazione	10
2.2.2	Design	11
2.2.3	Implementazione	12
2.2.4	Validazione	12
2.2.5	Evoluzione	12
2.3	Gestione dei cambiamenti	13
2.3.1	Prototipi	13
2.3.2	Consegna incrementale	14
2.4	Miglioramento dei processi	15
2.4.1	Misurazione dei processi	15

1 Introduzione

1.1 Cos'è l'ingegneria del software?

L'ingegneria del software è un insieme di metodologie, teorie, metodi e strumenti, che guidano nello sviluppo di software **professionale** in modo che esso fornisca le funzionalità richieste, sia performante e mantenibile, affidabile e usabile. Il **software** non è solo il programma e l'eseguibile, ma anche **la documentazione associata**. Le attività principale dell'ingegneria del software sono:

- **Specifica del software:** Il cliente e l'ingegnere del software definiscono le funzionalità e i vincoli del software da produrre.
- **Sviluppo:** Il software viene progettato e implementato.
- **Validazione:** Il software viene verificato per assicurarsi che soddisfi i requisiti forniti dal cliente.
- **Evoluzione:** Il software viene modificato per adattarlo a nuovi requisiti del cliente o del mercato.

1.1.1 Economia

Il software spesso costa più dell'hardware, e il costo è più legato alla manutenzione che allo sviluppo. Quindi l'ingegneria del software è importante per ridurre i costi di manutenzione.

1.1.2 Fallimento dei progetti

I progetti spesso falliscono per la **crescente complessità del sistema**: nuove tecniche di sviluppo e nuove tecnologie rendono i sistemi sempre più complessi, e quindi più difficili da mantenere.

1.2 Prodotti software

1.2.1 Prodotti generici

Sono prodotti che vengono pubblicizzati e venduti a qualsiasi cliente che ne faccia richiesta. La specifica è di proprietà del produttore e le decisioni sulle modifiche sono prese dai produttori.

1.2.2 Prodotti personalizzati

Sono prodotti che vengono commissionati da clienti specifici per soddisfare le loro necessità. La specifica è di proprietà del cliente dopo una contrattazione con il produttore e le decisioni sulle modifiche sono prese dal cliente.

1.3 Etica dell'ingegneria del software

Progettare software richiede ulteriori responsabilità rispetto alla semplice applicazione delle abilità tecniche. Gli ingegneri si devono comportare in modo etico e responsabile e ciò implica più del semplice seguire la legge, bisogna infatti seguire un insieme di principi moralmente corretti.

I problemi principali della responsabilità morale sono:

- **Confidenzialità:** gli ingegneri dovrebbero rispettare la confidenzialità delle informazioni fornite dai clienti o da altri a prescindere dal fatto che sia stato firmato un contratto di confidenzialità
- **Competenza:** gli ingegneri non dovrebbero rappresentare in modo negativo il loro livello di competenza e non dovrebbero accettare volutamente incarichi che vanno oltre le loro competenze
- **Diritti di proprietà intellettuale:** gli ingegneri dovrebbero essere a conoscenza delle leggi che governano l'utilizzo delle proprietà intellettuali, come brevetti, copyright, ecc...e dovrebbero assicurarsi che la proprietà intellettuale dei datori di lavoro e dei clienti sia protetta
- **Utilizzo sbagliato dei computer:** gli ingegneri non dovrebbero utilizzare i computer per attività immorali, illegali o malevoli

1.3.1 Codice etico ACM/IEEE

Società professionali hanno creato un codice etico da rispettare per tutti i membri di queste società. Questo codice contiene 8 principi relativi al comportamento e alle decisioni degli ingegneri:

- **Pubblico:** Gli ingegneri del software devono agire in modo coerente con l'interesse pubblico.
- **Cliente e datore di lavoro:** Gli ingegneri del software devono agire in modo da tutelare al meglio gli interessi del loro cliente e datore di lavoro, in coerenza con l'interesse pubblico.
- **Prodotto:** Gli ingegneri del software devono garantire che i loro prodotti e le relative modifiche soddisfino i più alti standard professionali possibili.
- **Giudizio:** Gli ingegneri del software devono mantenere integrità e indipendenza nel loro giudizio professionale.
- **Gestione:** I manager e i leader nell'ingegneria del software devono aderire e promuovere un approccio etico alla gestione dello sviluppo e della manutenzione del software.
- **Professione:** Gli ingegneri del software devono promuovere l'integrità e la reputazione della professione in coerenza con l'interesse pubblico.
- **Colleghi:** Gli ingegneri del software devono essere equi e solidali con i loro colleghi.
- **Se stessi:** Gli ingegneri del software devono partecipare a un apprendimento continuo riguardante la pratica della loro professione e promuovere un approccio etico alla pratica professionale.

1.3.2 Esempi

Alcuni esempi di dilemmi etici sono:

- Disaccordo di principio con le politiche imposte dai superiori

- Il datore di lavoro si comporta in modo non etico e rilascia un sistema che può essere dannoso per la sicurezza senza finire la fase di test del sistema
- Partecipazione nello sviluppo di armi militari o di sistemi nucleari

1.3.3 Casi di studio

Prendiamo ad esempio i seguenti casi da studiare:

- *Pompa di insulina personale*: un sistema embedded in una pompa di insulina utilizzata dai diabetici per controllare l'insulina nel sangue

In base ad un sensore del sangue il sistema calcola la quantità di insulina da iniettare. Il calcolo è basato sul cambiamento dei livelli di glucosio nel sangue. Questi dati vengono inviati ad una micro-pompa che invia la dose corretta di insulina.

Questo è un sistema **safety-critical** perché bassi livelli di glucosio nel sangue possono portare a effetti negativi della salute.

I requisiti del sistema sono:

- Iniettare insulina quando richiesto
- Deve, in modo affidabile e corretto, iniettare la quantità corretta di insulina
- Il sistema deve essere progettato e implementato per far sì che il sistema soddisfi **sempre** i requisiti
- *Un sistema di gestione di cartelle cliniche per pazienti con problemi di salute mentale*: Mentcare, un sistema utilizzato per mantenere cartelle cliniche per pazienti con problemi di salute mentale

La maggior parte dei pazienti non ha bisogno di essere ricoverata, ma ha bisogno di specialisti che li seguono. Questo sistema permette ai pazienti di avere un accesso diretto ai loro dati in locale e di poterli condividere con i loro medici.

Gli obiettivi sono:

- Supportare le attività di trattamento dei pazienti
- Generare informazioni che permettono ai medici di valutare il trattamento e di fare diagnosi
- Monitorare i progressi dei pazienti
- Amministrare le cartelle cliniche

I problemi principali sono:

- Privacy dei dati confidenziali
- Sicurezza dei pazienti, che potrebbero avere comportamenti pericolosi verso se stessi o verso altri.

- *Stazione del meteo*: un sistema che colleziona dati sul meteo in aree remote

Le stazioni raccolgono dati da un insieme di sensori che periodicamente misurano temperatura, pressione, quantità di radiazione solare, quantità di pioggia, velocità e direzione del vento.

Queste stazioni sono responsabili di raccogliere e trasmettere i dati a un centro di controllo dopo un'iniziale processamento. Queste stazioni possono comunicare tramite i satelliti tra di loro per creare dei report sui principali problemi.

Il centro di controllo raccoglie i dati da tutte le stazioni, li analizza e li processa e li archivia.

I requisiti del sistema sono:

- Monitorare i sensori, l'hardware dell'energia e della comunicazione e riportare i problemi al centro di controllo
- Monitorare il sistema di energia assicurando che le batterie siano ricaricate quando le condizioni ambientali lo permettono e che i generatori siano spenti durante condizioni pericolose
- Supportare una reconfigurazione dinamica che permette di rimpiazzare parti del software con delle nuove versioni e di sostituire i sensori danneggiati

- *iLearn*: un sistema che agevola lo studio

iLearn è un sistema che contiene strumenti e applicazioni general-purpose progettati per supportare l'apprendimento. Gli strumenti presenti nel sistema sono scelti dai docenti e dagli studenti. Alcuni esempi di applicazioni sono:

- Fogli di calcolo
- Strumenti di presentazione
- Strumenti di consegna e gestione dei compiti ed esami

Questo sistema è di tipo service-oriented, cioè è composto da un insieme di servizi che possono essere combinati per creare un sistema unico e configurabile, questo permette di aggiornare il sistema in modo incrementale ogni volta che sono necessari nuovi servizi.

I requisiti del sistema sono:

- Servizi di utilità generale che forniscono funzionalità di base indipendenti dalle applicazioni e che potrebbero essere utilizzati da altri servizi nel sistema
- Servizi applicativi che mettono a disposizione applicazioni specifiche e contenuto educativo
- Servizi di configurazione che servono ad adattare un ambiente con un insieme specifico di applicazioni e definiscono in che modo i servizi sono condivisi tra professori, studenti e genitori

1.4 Caratteristiche di un buon software

- **Mantenibilità:** la facilità con cui il software può essere modificato per correggere difetti, migliorare le prestazioni o adattarlo a cambiamenti.
- **Affidabilità e sicurezza:** la capacità del software di svolgere le sue funzioni in modo corretto. I malfunzionamenti non devono causare danni fisici o economici. Utenti malintenzionati non devono poter violare la sicurezza del sistema.
- **Efficienza:** il software non deve sprecare risorse (CPU, memoria, ecc).
- **Accettabilità:** il software deve essere accettato dagli utenti per i quali è stato progettato.

1.5 Problemi che influenzano il software

- **Eterogeneità:** I sistemi devono sempre di più operare in modo distribuito, e quindi devono essere in grado di comunicare con sistemi diversi. Oppure bisogna garantire che il software funzioni su piattaforme diverse.
- **Cambiamento sociale o del business:** Il software deve essere in grado di adattarsi a cambiamenti sociali o organizzativi nelle aziende.
- **Sicurezza e fiducia:** Siccome il software fa parte della vita di tutti i giorni è essenziale che ci sia fiducia nel software.
- **Scalabilità:** Il software deve essere sviluppato a più scale, cioè soluzione che funziona in piccolo deve adattarsi anche a grandi scale senza rischiare di fallire.

1.6 Tipi di applicazione

Ci sono diversi tipi di sistemi software e non c'è un insieme universale di tecniche applicabili a tutti i sistemi. Quindi i metodi e gli strumenti utilizzati dipendono dal tipo di applicazione che si deve sviluppare, dalle richieste dei clienti e dalle competenze degli sviluppatori. I principali tipi di applicazione sono:

- **Applicazioni stand-alone:** Sono applicazioni che si eseguono su un singolo computer locale e includono le funzionalità necessarie per l'utente.
- **Applicazioni interattive transaction-based:** Sono applicazioni che sono eseguite su un computer in remoto e sono accessibili dagli utenti dai propri computer. Queste includono le applicazioni web come gli e-commerce.
- **Sistemi embedded:** Sono applicazioni che controllano e gestiscono dispositivi hardware. Questi sistemi sono i più numerosi.
- **Sistemi batch:** Questi sistemi elaborano grandi quantità di dati per produrre un output.
- **Sistemi di intrattenimento:** Questi sistemi sono per uso personale e servono ad intrattenere l'utente.

- **Sistemi di modellazione e simulazione:** Questi sistemi sono sviluppati da scienziati per modellare processi fisici o situazioni che includono tanti oggetti separati che interagiscono tra di loro.
- **Sistemi di collezione di dati (o IOT):** Questi sistemi raccolgono dati da sensori e li inviano ad un sistema centrale per l'elaborazione.
- **Sistemi di sistemi:** Questi sistemi sono composti da più sistemi software che collaborano tra di loro.

1.7 Principi fondamentali

- I sistemi devono essere sviluppati utilizzando un processo strutturato e ben pensato.
- L'affidabilità e la performance sono importanti per ogni tipo di software.
- Capire e gestire i requisiti del software è essenziale.
- Dove appropriato si dovrebbe riutilizzare software che è già stato sviluppato al posto di svilupparne uno da zero.

2 Processi del software

I **processi** del software sono un insieme di attività strutturate che sono necessarie per sviluppare un sistema software. Ci sono diversi tipi di processi, ma tutti coinvolgono le seguenti attività:

- **Specificazione**
- **Sviluppo**
- **Design e implementazione**
- **Validazione**
- **Evoluzione**

Un modello di processo software è una rappresentazione astratta di un processo da un punto di vista particolare.

Per descrivere e discutere i processi di solito si parla delle attività all'interno di questi processi e dell'ordine in cui queste attività vengono svolte. Alcune descrizioni possono anche includere:

- **Prodotti:** cioè i risultati di un'attività.
- **Ruoli:** cioè le responsabilità di un individuo o di un gruppo.
- **Pre e post condizioni:** cioè le condizioni che devono essere soddisfatte prima e dopo un'attività.

Alcuni esempi di processi sono:

- **Processi plan-driven:** sono processi dove tutte le attività sono pianificate in anticipo e tutti i progressi sono misurati rispetto al piano.

- **Processi agili:** sono processi in cui la pianificazione è incrementale ed è più facile adattarsi ai cambiamenti.

Nel pratico si utilizza un approccio ibrido tra i due.

2.1 Modelli di processo

- **Modello a cascata:** è un modello plan-driven con fasi di specificazione e di sviluppo distinte e separate
- **Modello incrementale:** la specifica, lo sviluppo e la validazione sono intercalate. Può essere plan-driven o agile.
- **Integrazione e configurazione:** i sistemi sono sviluppati da componenti già esistenti e configurabili.

Nel pratico, sistemi molto grandi sono sviluppati utilizzando un processo che incorpora elementi di tutti e tre i modelli.

2.1.1 Modello a cascata

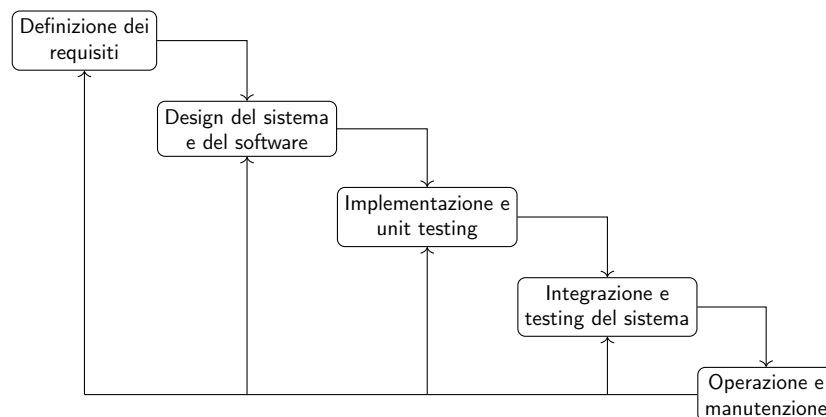


Figura 1: Modello a cascata

Le diverse fasi del modello a cascata sono:

- Analisi e definizione dei requisiti
- Design del sistema e del software
- Implementazione e unit testing
- Integrazione e testing del sistema
- Operazione e manutenzione

Il vantaggio del modello a cascata è la partizione inflessibile del progetto in fasi distinte rende difficile rispondere ai cambiamenti dei requisiti, quindi questo modello è adatto solo per progetti dove i requisiti sono ben definiti e i cambiamenti rimangono limitati. Però pochi progetti soddisfano queste condizioni.

Questo modello è più utilizzato in sistemi molto grandi e sviluppati da molte persone. In quelle circostanze la natura plan-driven di questo modello è utile per coordinare il lavoro.

2.1.2 Modello incrementale

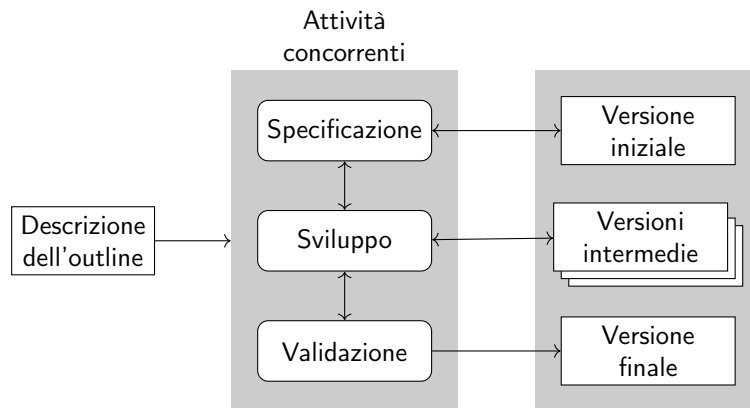


Figura 2: Modello incrementale

I vantaggi del modello incrementale sono:

- Il costo del cambiamento dei requisiti è ridotto e la quantità di analisi e documentazione da rifare è molto meno rispetto al modello a cascata.
- È più facile ottenere un feedback dagli utenti sul lavoro svolto.
- È possibile consegnare software utile più velocemente al cliente.

Gli svantaggi del modello incrementale sono:

- Il processo non è visibile, cioè è difficile valutare il progresso. I manager hanno bisogno di costanti "deliverables", cioè risultati intermedi, per misurare il progresso e non sempre è poco costoso produrre documenti che riflettono lo stato di ogni versione del software.
- La struttura del sistema tende a decadere con l'aggiunta di nuovi incrementi. A meno che il tempo e i soldi non vengano spesi per il refactor e il miglioramento del software, i cambiamenti regolari tenderanno a corrompere la struttura del sistema e quindi incorporare nuovi cambiamenti diventerà sempre più difficile e costoso.

2.1.3 Modello a integrazione e configurazione

Questo modello si basa sulla realizzazione di sistemi integrati tramite componenti software già esistenti e configurabili. Il riutilizzo è l'approccio standard per creare molti tipi di sistemi.

I tipi di software riutilizzabile sono:

- Applicazioni o sistemi stand-alone (a volte chiamati COTS, Commercial Off-The-Shelf) che sono configurati per essere utilizzati in un certo ambiente
- Collezioni di oggetti o librerie che sono sviluppati come pacchetti per l'integrazione con vari framework
- Servizi web sviluppati secondo standard che permettono l'interazione tramite internet

Le attività principali di questo modello sono:

- Specificazione dei requisiti
- Ricerca e analisi del software riutilizzabile
- Adattamento dei requisiti
- Configurazione del sistema
- Adattamento e integrazione dei componenti

I vantaggi di questo modello sono:

- Riduzione dei costi e del rischio siccome meno software è sviluppato da zero
- Deployment più veloce

Gli svantaggi sono:

- È inevitabile fare compromessi per i requisiti, quindi i sistemi potrebbero non essere più ciò che aveva richiesto il cliente
- Perdita di controllo sull'evoluzione dei componenti riutilizzati

2.2 Attività dei processi

I processi software reali sono sequenze interconnesse di attività tecniche, collaborative e manageriali con l'obiettivo di specificare, progettare, implementare e testare un sistema software. Questi quattro processi base sono organizzati in modo differente in processi di sviluppo diversi.

2.2.1 Specificazione

Il processo di stabilire che processi sono richiesti e quali sono i vincoli sull'operazione e lo sviluppo del sistema è chiamato **specificazione del software** e consiste in:

- **Ricavo e analisi dei requisiti:** cosa vogliono o cosa si aspettano gli utenti interessati dal sistema
- **Specifica dei requisiti:** definizione formale dei requisiti del sistema
- **Validazione dei requisiti:** verifica che i requisiti siano completi e consistenti

2.2.2 Design

Questo processo consiste nella conversione delle specifiche del sistema in un sistema eseguibile e consiste in:

- **Design del software:** progettazione di una struttura software che realizzi i requisiti del sistema
- **Implementazione:** traduzione del design in un programma eseguibile

Queste attività sono interconnesse e spesso si intrecciano.

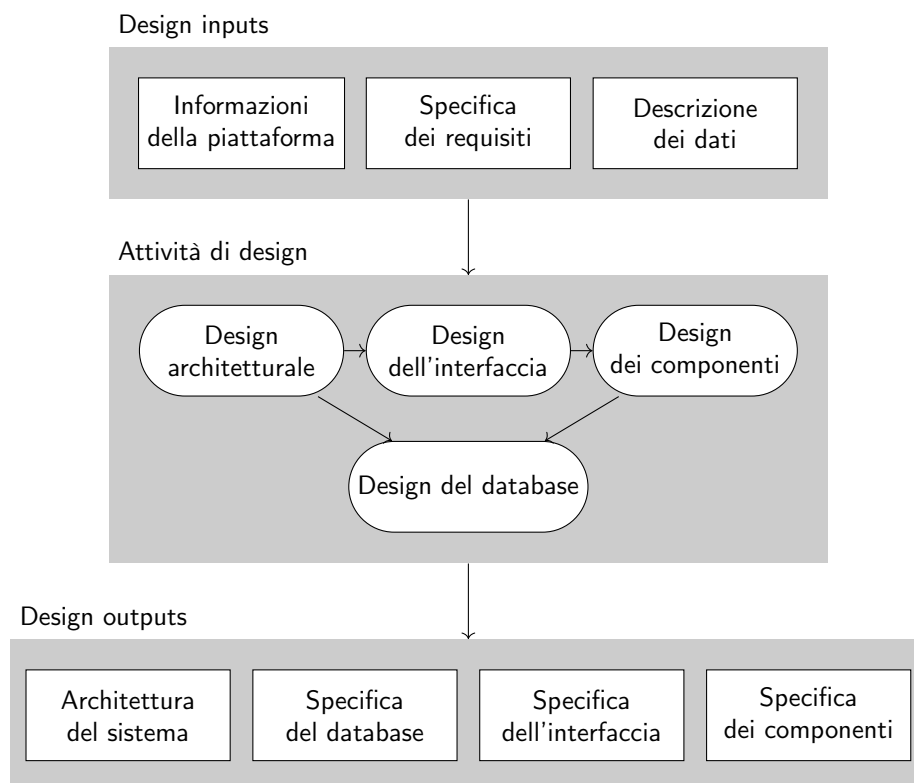


Figura 3: Modello generale del processo di design

Le principali attività di design sono:

- **Design architetturale** dove si identifica la struttura complessiva del sistema, i componenti principali, le loro relazioni e come sono distribuiti
- **Design del database** dove si progettano le strutture dati del sistema e come esse vengono rappresentate all'interno del database
- **Design dell'interfaccia** dove si definiscono le interfacce tra i componenti del sistema
- **Design e selezione dei componenti** dove si cercano componenti riutilizzabili. Se non sono disponibili si progettano nuovi componenti

2.2.3 Implementazione

Il software è implementato sviluppando dei programmi, oppure configurando un'applicazione. Il design e l'implementazione sono attività interconnesse per la maggior parte dei sistemi software.

La programmazione è un'attività individuale senza alcun processo standard, mentre invece il debugging consiste nel trovare e correggere errori nei programmi.

2.2.4 Validazione

La verifica e la validazione serve a dimostrare che un sistema è conforme alle specifiche e che soddisfa i requisiti del cliente. La validazione consiste nel controllare e verificare i processi e anche nel fare **testing**.

Il testing consiste nell'esecuzione di un sistema con dei casi di test derivati dalla specificazione dei dati reali che il software deve processare. Il testing è l'attività di validazione più utilizzata.

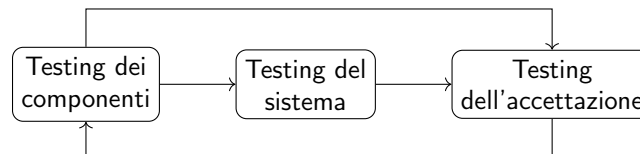


Figura 4: Fasi del testing

Le principali fasi del testing sono:

- **Testing dei componenti:** I singoli componenti vengono testati individualmente. I componenti possono essere funzioni, oggetti o gruppi coesi di tali entità.
- **Testing del sistema:** I componenti vengono integrati e il sistema viene testato come un'entità unica. È importante testare eventuali proprietà emergenti
- **Testing dell'accettazione:** Il sistema viene testato con i dati reali del cliente per verificare che soddisfi i requisiti del cliente

2.2.5 Evoluzione

Il software è flessibile e può cambiare col cambiamento dei requisiti. Anche se la linea tra sviluppo e mantenimento è spesso sfumata, questo diventa irrilevante siccome sempre meno sistemi software vengono sviluppati da zero.

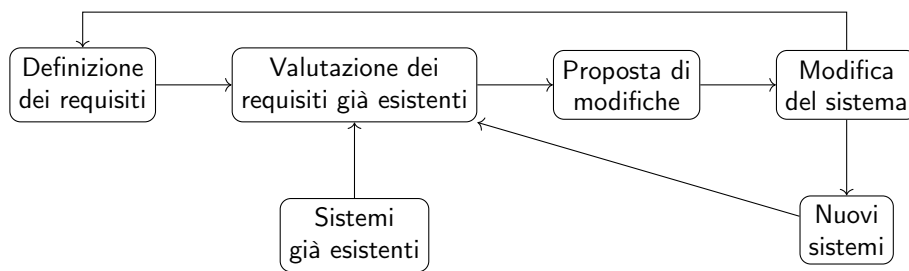


Figura 5: Evoluzione di un sistema

2.3 Gestione dei cambiamenti

Il cambiamento è inevitabile in tutti i grandi progetti software ed esso è dovuto da:

- Cambiamenti del mercato
- Nuove tecnologie offrono nuove possibilità
- Cambiamento delle piattaforme

Il cambiamento porta a costi aggiuntivi dovuti alla necessità di rifare il lavoro precedente più il lavoro aggiuntivo per implementare le nuove funzionalità. Per ridurre questi costi si può **anticipare il cambiamento**, cioè includere delle attività nel processo di sviluppo che permettano di anticipare i possibili cambiamenti prima che avvenga il rework, ad esempio si possono sviluppare dei prototipi da mostrare al cliente prima del prodotto finito. Oppure si può aumentare la tolleranza al cambiamento, quindi sviluppare il software in modo che sia più facile cambiare il software in futuro.

2.3.1 Prototipi

Un prototipo è un'implementazione parziale del sistema che viene utilizzata per dimostrare concetti e provare nuove opzioni di design. Un prototipo può essere utilizzato in:

- Processi di specificazione dei requisiti per aiutare a scoprire i requisiti del sistema
- Processi di design per esplorare alternative di design
- Processi di testing per provare parti del sistema

I vantaggi della prototipazione sono:

- Miglioramento nell'usabilità del sistema
- Un prodotto più vicino alle aspettative e alle esigenze del cliente
- Miglioramento della qualità del design
- Miglioramento della manutenibilità
- Riduzione dell'impegno di sviluppo

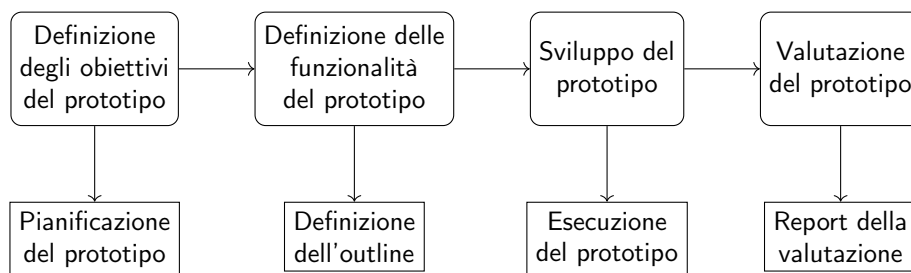


Figura 6: Processo di sviluppo del prototipo

Lo sviluppo di un prototipo si può basare su linguaggi e strumenti fatti apposta per la prototipazione. La prototipazione consiste nel creare una versione parziale funzionante del sistema e ciò comporta tralasciare alcune funzionalità. Il prototipo si deve concentrare su:

- Aree del prodotto che non sono ben definite o capite
- Tralasciare il controllo degli errori per concentrarsi sui requisiti funzionali

I prototipi dovrebbero essere scartati siccome non sono una buona base per il sistema finale.

2.3.2 Consegna incrementale

Al posto di consegnare il sistema in un'unica volta già completo, si può separare lo sviluppo e la consegna in incrementi che implementano parte delle funzionalità richieste seguendo un ordine di priorità. Le due fasi si compongono di:

- **Sviluppo incrementale**
 - Sviluppare il sistema in incrementi e validare ogni incremento prima di procedere con il successivo
 - Utilizzare l'approccio utilizzato nei metodi agili per lo sviluppo
 - La valutazione di ogni incremento è fatta dal cliente/utente
- **Consegna incrementale**
 - Consegnare un incremento per permettere al cliente di utilizzarlo
 - Il cliente può valutare l'incremento e fornire feedback
 - Il feedback può essere utilizzato per modificare i requisiti per gli incrementi successivi

I vantaggi della consegna incrementale sono:

- Il feedback del cliente può essere ottenuto ad ogni incremento per migliorare il sistema
- I primi incrementi possono funzionare da prototipo
- Rischio più basso di fallimento del progetto
- I servizi con priorità più alta tendono ad essere testati di più

I problemi della consegna incrementale sono:

- Molti sistemi richiedono un insieme di funzionalità comuni utilizzate da parti diverse del sistema, ma nella consegna incrementale la definizione dei requisiti avviene soltanto quando è necessario sviluppare un nuovo incremento e quindi è difficile identificare queste funzionalità comuni in anticipo
- La specifica è definita in congruenza con lo sviluppo, ma questo va contro il modello di molte organizzazioni

2.4 Miglioramento dei processi

Il miglioramento dei processi consiste nel capire processi esistenti e trovare modi per migliorare la qualità, ridurre il costo o accelerare lo sviluppo.

Ci sono due principali approcci per il miglioramento dei processi:

- **Approccio di maturità:** si concentra sul miglioramento dei processi e della gestione del progetto e introdurre delle buone pratiche di sviluppo software.
- **Approccio agile:** si concentra sullo sviluppo iterativo e sulla riduzione di overhead dei processi

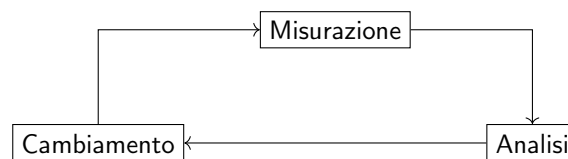


Figura 7: Ciclo di miglioramento dei processi

Le fasi del ciclo di miglioramento dei processi sono:

- **Misurazione dei processi:** si misurano i processi esistenti per capire come funzionano e dove ci sono problemi e questo farà da base per il miglioramento
- **Analisi dei processi:** si analizza il processo corrente e tutti gli svantaggi vengono identificati e si stila un modello che descrive il processo (chiamato process map)
- **Cambiamento dei processi:** si identificano i cambiamenti che possono essere fatti per migliorare i problemi individuati. Quando questi cambiamenti vengono introdotti, il ciclo ricomincia

2.4.1 Misurazione dei processi

Quando possibile bisognerebbe raccogliere dati quantitativi sui processi per utilizzarli a monitorare il miglioramento dei processi. I dati però non devono guidare il processo di miglioramento. Le metriche più comuni sono:

- Il tempo richiesto per completare un'attività
- Le risorse richieste per i processi o le attività

- Il numero di occorrenze di un particolare evento

Il modello di maturità SEI (Software Engineering Institute) è un modello per valutare la maturità dei processi di sviluppo software:

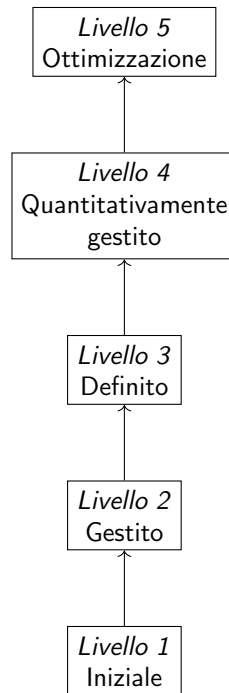


Figura 8: Livelli di maturità

- **Iniziale:** essenzialmente incontrollato
- **Ripetibile:** Procedure di gestione del prodotto definite e utilizzate
- **Definito:** Processi di sviluppo definiti e utilizzati
- **Gestito:** Strategie di gestione di qualità definiti e utilizzati
- **Ottimizzazione:** Strategie di miglioramento dei processi definite e utilizzate