1 Introduzione

1.1 Sistemi software

Un *sistema software* è un insieme di componenti software che funzionano in modo coordinato allo scopo di informatizzare una certa attività. La realizzazione di un sistema software richiede l'impiego di un gruppo di lavoro, nel quale ogni persona ricopre un ruolo ben preciso e le attività dei vari gruppi vanno coordinate, e tempo da dedicare alle varie fasi di sviluppo.

Esistono due categorie di sistemi software: i *sistemi generici*, definiti in base alle tendenze di mercato, e i *sistemi customizzati*, richiesti da uno specifico cliente (il committente).

1.2 Il processo software

Con *ingegneria del software* si intende l'applicazione del processo dell'Ingegneria alla produzione di sistemi software. Il processo è suddiviso in:

- specifica: definizione dei requisiti funzionali e non funzionali
- progettazione: si definiscono architettura, controllo, comportamento dei componenti, strutture dati, algoritmi, struttura del codice, interfaccia utente
- implementazione: scrittura del codice e integrazione dei moduli
- collaudo: si controlla se il sistema ha difetti di funzionamento e se soddisfa i requisiti
- manutenzione: modifiche del sistema dopo la consegna

1.3 Gestione del processo

L'ingegneria del software si occupa anche della gestione del progetto che si svolge in parallelo al processo software. Le principali attività di gestione sono l'*assegnazione* di risorse (umane, finanziarie...), la *stima del tempo* necessario per ogni attività, la *stima dei costi* e la *stima dei rischi*.

2 Specifica

La *specifica* è l'insieme di attività necessarie per generare il documento dei requisiti che descrive i *requisiti funzionali* e i *requisiti non funzionali*: descrive il "cosa" il sistema deve fare, non il "come". I requisiti servono per una proposta di contratto e modellare fasi successive del processo software.

2.1 Requisiti funzionali

I requisiti funzionali sono i servizi che il cliente richiede al sistema. Per ogni servizio si descrive:

- cosa accade nell'interazione tra utente e sistema
- cosa accade in seguito ad un certo input o stimolo
- cosa accade in particolari sistuazioni, ad esempio in caso di eccezioni

Non viene descritto come funziona internamente il sistema, in quanto è oggetto della successiva fase di progettazione.

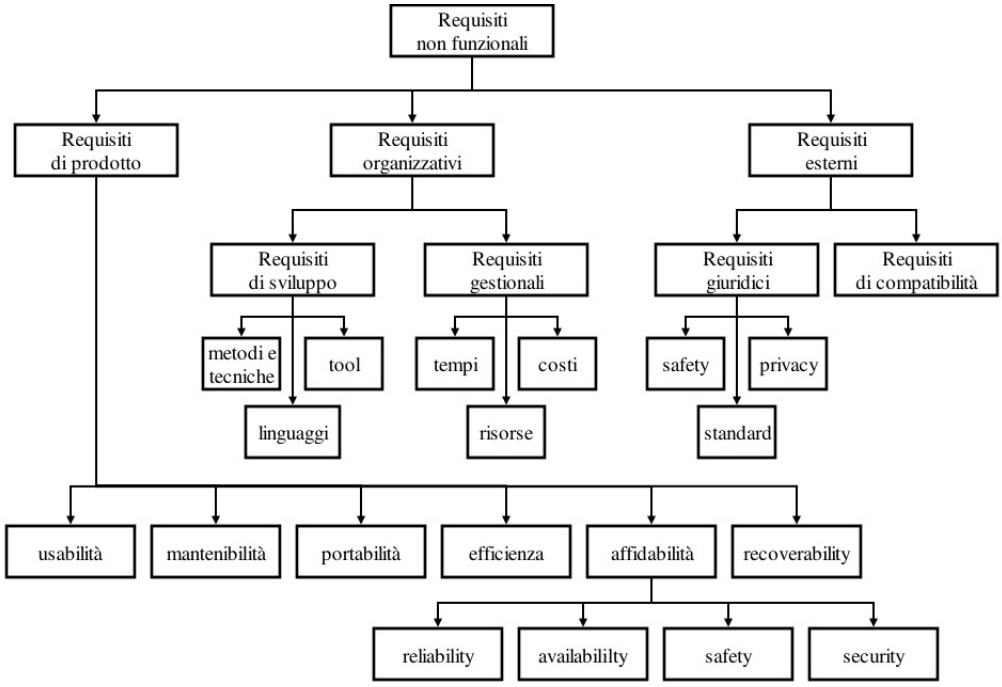
2.2 Requisiti non funzionali

I requisiti non funzionali sono divisi in tre categorie: *requisiti di prodotto*, *requisiti organizzativi* e *requisiti esterni*.

I requisiti di prodotto sono attributi che definiscono la qualità del sistema. Una *proprietà complessiva* riguarda il sistema nel suo complesso; una *proprietà emergente* è una proprietà che "emerge" dal funzionamento del sistema, dopo che è stato implementato.

I requisiti organizzativi sono caratteristiche riguardanti le fasi del processo software o la gestione del progetto. I *requisiti di sviluppo* sono i metodi e le tecniche di sviluppo utilizzati; i *requisiti gestionali* sono le risorse utilizzate.

I requisiti esterni derivano da fattori esterni al sistema e al processo software. Essi sono i requisiti di compatibilità con altri sistemi e aspetti giuridici.



Usabilità

L'*usabilità* è il grafo di facilità con cui l'utente riesce a comprendere l'uso del software. Il sistema deve avere un'interfaccia utente intuitiva ed curata, in quanto è fattore critico per il successo di un prodotto. L'uso del sistema deve essere ben documentato, per permettere all'utente di apprendere velocemente l'uso del prodotto. Il *training* degli utenti può migliorare l'usabilità del prodotto.

Mantenibilità

la *mantenibilità* è il grado di facilità di manutenzione. Le cause della manutenzione sono molteplici, e deve essere possibile l'evoluzione del software per soddisfare i requisiti nel tempo.

Portabilità

La *portabilità* è la capacità di migrazione da un ambiente ad un altro.

Recoverability

La *recoverability* è la capacità di ripristinare lo stato e i dati del sistema dopo che si è verificato un fallimento.

Efficienza

L'*efficienza* è il livello di prestazioni del sistema, e può essere misurato in vari modi: tempo di risposta, numero medio di richieste...

Affidabilità

L'*affidabilità* è il grado di fiducia con cui si ritiene che il sistema svolga in modo corretto la propria funzione. Ci sono varie misure di affidabilità:

- *reliability*: capacità di fornire i servizi in modo continuativo per una certa durata di tempo
- *availability*: capacità di fornire i servizi nel momento richiesto
- *safety*: capacità di operare senza causare danni materiali
- *security*: capacità di proteggersi da intrusioni e attacchi

Un sistema è definito *critico* quando il suo non corretto funzionamento può provocare conseguenze "disastrose" a persone e ambiente (*safety critical system*) o perdite economiche (*business critical system*). Il costo cresce in modo esponenziale rispetto al grado di affidabilità richiesto.

2.3 Processo di specifica

Il *processo di specifica* è il processo per generare il documento dei requisiti, ed è diviso in più fasi. Lo stesso requisito viene definito con due gradi di dettaglio diversi. Il *requisito utente* è descritto ad alto livello, in linguaggio naturale, ed è il risultato della deduzione dei requisiti. Il *requisito di sistema* è descritto dettagliatamente, fornendo tutti i dettagli necessari per la fase di progettazione, ed è il risultato dell'analisi dei requisiti.

Studio di fattibilità

Lo *studio di fattibilità* è la valutazione della possibilità di sviluppare il sistema e dei suoi vantaggi per il committente. Si decide se la costruzione del sistema è fattibile date le risorse disponibili e se il sistema è effettivamente utile al cliente. Per svolgere lo studio si raccolgono informazioni e si prepara un rapporto di fattibilità, che contiene la valutazione della possibilità di costruire un sistema e dei vantaggi che possono derivare dalla sua introduzione.

Deduzione dei requisiti

La *deduzione dei requisiti* è la raccolta di informazioni da cui dedurre quali sono i requisiti. Le informazioni si possono raccogliere mediante uno studio del dominio applicativo del sistema richiesto, il dialogo con stakeholder, studio di sistemi simili già realizzati e studio di sistemi con cui dovrà interagire quello da sviluppare.

Il *dominio applicativo* è l'insieme di entità reali su cui il sistema software ha effetto.

Uno *stakeholder* è, in ambito economico, il soggetto che può influenzare il successo di un'impresa o che ha interessi nelle decisioni dell'impresa; in ambito del processo software sono persone che possono influenzare il processo o che hanno interesse nelle decisioni assunte in esso.

È possibile dialogare con gli stakeholder tramite *interviste*, nelle quali viene chiesto di raccontare attraverso degli esempi reali come l'attività lavorativa funziona realmente, e tramite *etnografia*, l'osservazione dei potenziali utenti nello svolgimento delle loro mansioni.

Il dialogo con gli stakeholder presenta vari problemi, in quanto non sono in grado di indicare chiaramente cosa vogliono dal sistema, omettendo informazioni ritenute ovvie ed utilizzando terminologia non adatta. Inoltre, lo stesso requisito può essere espresso da più stakeholder in maniera differente, ed addirittura essere in conflitto.

Molti problemi scaturiscono dal *linguaggio naturale*, in quanto una descrizione ad alto livello di un requisito può generare confusione. La soluzione è quella di utilizzare il linguaggio in modo coerente, evitando gergo tecnico ed illustrando i requisiti tramite semplici diagrammi.

Analisi dei requisiti

La *analisi dei requisiti* è l'organizzazione, negoziazione e modellazione dei requisiti. Comprende:

- *classificazione e organizzazione dei requisiti*
- *assegnazione di priorità ai requisiti*: si stabilisce il grado di rilevanza di ogni requisito
- *negoziazione dei requisiti*
- *modellazione analitica dei requisiti*: produzione di modelli che rappresentano o descrivono nel dettaglio i requisiti

I *requisiti di sistema* sono l'espansione dei requisiti utente, e formano la base per la progettazione. Il linguaggio naturale non è adatto alla definizione di un requisito di sistema, quindi è necessario usare template, modelli grafici o notazione matematica.

Il *modello data-flow*, detto anche pipe & filter, permette di modelalre il flusso e l'elaborazione dei dati, ma non prevede la gestione degli errori. L'elaborazione è di tipo batch: input $\rightarrow$ elaborazione $\rightarrow$ output.

I requisiti non funzionali si possono specificare definendone delle misure quantitative:

- *efficienza*: tempo di elaborazione delle richieste, occupazione di memoria
- *affidabilità*: probabilità di malfunzionamento, disponibilità
- *usabilità*: tempo di addestramento, aiuto contestuale

Il *documento dei requisiti* contiene il risultato della deduzione e dell'analisi, ed è la dichiarazione ufficiale di ciò che si deve sviluppare. Il documento contiene una breve introduzione che descrive le funzionalità del sistema, un glossario contenente le definizione di termini tecnici, i requisiti utente e i requisiti di sistema, correlati con modelli UML. Il documento è letto da tutte le figure coinvolte nella realizzazione del progetto.

Validazione dei requisiti

La *validazione dei requisiti* è la verifica del rispetto di alcune proprietà da parte del documento dei requisiti, serve ad evitare la scoperta di *errori di specifica* durante le fasi successive del processo software. Sono da verificare le segueni proprietà:

- *completezza*: tutti i requisiti richiesti dal committente devono essere documentati
- *coerenza*: la specifica dei requisiti non deve contenere definizioni tra loro contraddittorie
- *precisione*: l'interpretazione di una definizione di requisito deve essere unica

- *realismo*: i requisiti devono essere implementati date le risorse disponibili
- *tracciabilità*

Quando si modifica un requisito bisogna valutarne l'impatto sul resto della specifica: è quindi necessario tracciarlo. Vari tipi di *tracciabilità*:

- *tracciabilità della sorgente*: reperire la fonte d'informazione relativa al requisito
- *tracciabilità dei requisiti*: individuare i requisiti dipendenti
- *tracciabilità del progetto*: individuare i componenti del sistema che realizzano il requisito
- *tracciabilità dei test*: individuare i test-case usati per collaudare il requisito

Per validare i requisiti si può impegare un gruppo di *revisori* che ricontrolli i requisiti e *costruire dei prototipi*.

3 Progettazione

3.1 Attività di progettazione

Durante la fase di *progettazione architetturale* viene definita la struttura del sistema, come questo verrà distribuito e come il sistema si dovrà comportare. Sono inoltre progettate le strutture dati, gli algoritmi e la GUI.

3.2 Progettazione architetturale

Strutturazione

Il sistema può essere strutturato in vari sottosistemi (strati), tipicamente tre. Ogni strato interagisce con gli strati adiacenti. Gli strati sono:

- *presentazione*: l'interfaccia utente, raccoglie i dati dall'utente
- *elaborazione*: elabora i dati in input e produce dati in output
- *gestione dei dati*: il database

Un *sottosistema* è la parte del sistema dedicata a svolgere una certa attività, mentre un *modulo* è la parte del sottosistema dedicata a svolgere particolari funzioni legate all'attività del sottosistema.

Deployment

Con *deployment* si intende la distribuzione dei componenti in vari dispositivi hardware. Nel *deployment a 1-tier* i tre stati del sistema sono concentrati su un dispositivo, in quello a *2-tiers* su due e in quello a *3-tiers* su tre.

Con deployment a 2-tiers si possono avere due soluzioni: *fat client* e *thin client*. Nel modello *thin client*, il server si occupa dell'elaborazione e della gestione dei dati, mentre il client si occupa della presentazione. Nel modello *fat client*, il server si occupa della gestione dei dati, mentre il client si occupa della presentazione e dell'elaborazione.

Metodo di controllo

Un componente fornisce servizi ad altri componenti. Un'*interfaccia* è un insieme di operazioni che il componente mette a disposizione di altri componenti ed è condivisa con i componenti che lo invocano. Un *corpo* è la parte interna del componente e non è conosciuto agli altri componenti. La separazione tra interfaccia e corpo è detta *information hiding*.

Esistono diversi stili di controllo (attivazione) tra componenti:

- controllo *centralizzato*: è presente un componente detto *controllore*, che controlla l'attivazione e il coordinamento degli altri componenenti
- controllo *basato su eventi*: è basato su eventi esterni (ad esempio un segnale) ed ogni componente si occupa di gestire determinati eventi; il gestore degli eventi è detto *broker*, che rileva l'evento e lo notifica tramite broadcast (selettivo o non selettivo)
- controllo *call-return*: il controllo passa dall'alto verso il basso
- controllo *client-server*: un componente client chiede un servizio ad un componente server attraverso una chiamata di proceduta e il componente server risponde

Modellazione del comportamenti ad oggetti

I componenti del sistema sono considerati come oggetti che interagiscono. Un *oggetto* è definito da *attributi* e *operazioni*. Gli oggetti comunicano tra di loro attraverso lo scambio di messaggi.

4 Collaudo

La fase di collaudo avviene alla fine dell'implementazione del sistema. Si ricercano e correggono difetti, si controlla che il prodotto realizzi ogni servizio senza malfunzionamenti (fase di *verifica*) e che soddisfi i requisiti del committente (fase di *validazione*).

4.1 Ispezione

Motivazioni

L'*ispezione* è una tecnica statica di analisi del codice, basata sulla lettura di questo e della documentazione. L'ispezione è meno costosa del testing e può essere eseguita su una versione incompleta del sistema, ma alcuni requisiti non funzionali non si possono collaudare solo con l'ispezione (efficienza, affidabilità...).

Un team analizza il codice e segnala i possibili difetti. Viene seguita una checklist che indica i possibili difetti da investigare:

- errori nei dati
- errori di controllo
- errori di I/O
- errori di interfaccia
- errori nella gestione della memoria
- errori di gestione delle eccezioni

Ruoli nel team di ispezione

Il team di ispezione è composto da *autori* del codice, che correggono i difetti rilevati durante l'ispezione, *ispettori*, che trovano difetti, e *moderatore*, che gestisce il processo di ispezione.

Processo di ispezione

Durante la fase di *pianificazione*, il moderatore seleziona gli ispettori e controlla che il materiale sia completo.

Durante la fase di *introduzione*, il moderatore organizza una riunione preliminare con autori e ispettori, nella quale è discusso lo scopo del codice e la checklist da seguire.

Durante la fase di *preparazione individuale*, gli ispettori studiano il materiale e cercano difetti nel codice in base alla checklist ed all'esperienza personale.

Durante la fase di *riunione di ispezione*, gli ispettori indicano i difetti individuati.

Durante la fase di *rielaborazione*, il programma è modificato dall'autore per correggere i difetti.

Durante la fase di *prosecuzione*, il moderatore decide se è necessario un ulteriore processo di ispezione.

Analisi statica del codice

Gli strumenti CASE supportano l'ispezione del codice eseguendo su di esso l'*analisi del flusso di controllo*, che assicura l'assenza di cicli con uscire multiple, salti incondizionati e codice non raggiungibile; l'*analisi dell'uso dei dati*, che assicura l'assenza di problemi legati alle variabili; l'*analisi delle interfacce* e l'*analisi della gestione della memoria*. Due strumenti per l'analisi statica sono il compilatore gcc e SonarQube.

L'analisi statica precede l'ispezione del codice fornendo informazioni utili all'individuazione dei difetti, ma non è sufficiente.

4.2 Testing

Processo di testing

Durante la fase di *testing*, sono preparati ed eseguiti i test case, ed i loro risultati in output sono confrontati con quelli attesi.

Nella fase di *debugging* viene controllata l'esecuzione del programma e il valore delle variabili, in modo da individuare l'errore. L'errore viene corretto e viene eseguito un *testing di regressione*, nel quale si ripete l'ultimo test-case e tutti quelli precedenti.

4.2.1 Test-case

Un *test-case* è composto da dati in input e dati in output attesi.

Esistono due approcci per scegliere test-case. Nel testing *black box*, la scelta dei test-case è basata sulla conoscenza di quali sono i dati in input e quelli in output.

Dato l'insieme dei dati in input e l'insieme dei dati in output, una *partizione di equivalenza* è un sottoinsieme dei dati in input per cui il sistema produce sempre lo stesso dato in output (oppure simili). Si può fare black box testing usando delle partizioni di equivalenza. Per scrivere test-case, si individuano le partizioni di equivalenza e si scelgono un numero finito di test-case: in particolare si scelgono test-case con dati I/O al confine delle partizioni di equivalenza (*Boundaty Value Analysis*).

Nel testing *white box*, la scelta dei test-case è basata sulla struttura del codice. L'obiettivo è testare ogni parte del codice. Il codice viene rappresentato con un *flow graph*, un grafo che rappresenta i possibili cammini nel codice.

4.2.2 Complessità ciclomatica

Nel codice, un cammino si dice *indipendente* se introduce almeno una nuova sequenza di istruzioni o una nuova condizione. Il numero di cammini indipendenti equivale alla *complessità ciclomatica* (CC) del flow graph. CC è il numero minimo di test-case richiesti per eseguire almeno una volta ogni parte del codice. I *dynamic program analyser* sono strumenti CASE che, dato un test-case, indicano quali parti del codice sono state interessate da un test-case e quali sono ancora da testare.

Testing d'integrazione

I componenti possono essere implementati gradualmente oppure contemporaneamente. Il testing riguarda ciascun componente e la sua integrazione nel sistema. Prima vengono testati i componenti in maniera isolata, poi viene testato il sistema con le componenti integrate fino ad allora. Il testing compiuto sul sistema con tutte le componenti è detto *release testing*.

Lo scopo del *testing d'integrazione* è verificare l'interfacciamento corretto dei componenti. Qui si applicano i test-case applicati nei test d'integrazione precedenti e si applicano nuovi test. Se i componenti integrati sono pochi, si usa white box testing.

Nel *top-down integration testing* si costruiscono prima i moduli primari, poi quelli secondari. I moduli secondari non ancora disponibili sono sostituiti da STUB, dei moduli "dummy" che forniscono servizi pre-impostati.

Nel *bottom-up integration testing* si costruiscono prima i moduli secondari, poi quelli primari. I moduli primari non ancora disponibili sono sostituiti da DRIVER, dei test program che invocano i moduli secondo i dati selezionati per test-case.

Release testing

Il *release testing* riguarda il sistema completo. Riguarda la validazione ed è di tipo black-box.

Per i sistemi customizzati si ha il *test di accettazione*, nel quale il committente controlla il soddisfacimento dei requisiti e si effettuano correzioni finchè il cliente non è soddisfatto.

Per i sistemi generici si hanno le fasi di *alpha e beta testing*. La versione "alpha" del sistema viene resa disponibile ad un gruppo di sviluppatori per il collaudo finchè non si raggiunge ad un sistema soffisfacente; la versione "beta" del sistema viene resa disponibile ad un gruppo di clienti ed eventualmente corretta, dopodichè il prodotto viene messo sul mercato.

Stress testing

Lo *stress testing* serve per verificare l'efficienza e l'affidabilità del sistema. Viene svolto quando il sistema è completamente integrato. Vengono eseguiti test in cui il carico di lavoro è molto superiore a quello previsto normalmente.

Altri tipi di testing

- *test di usabilità*: si valuta la facilità con cui gli utenti riescono ad usare il sistema
- *recovery testing*
- *security testing*: simulazione di attacchi dall'esterno
- *deployment testing*: verifica dell'installazione sui dispositivi
- *back-to-back testing*: si usa quando varie versioni del sistema sono disponibili, si effettua lo stesso test su tutte le versiono e si confrontano gli output delle varie versioni

5 Manutenzione

La manutenzione riguarda tutte le modifiche fatte al sistema dopo la consegna. La manutenzione è un processo ciclico, che permette al sistema di "evolversi".

5.1 Tipi di manutenzione e costo

Manutenzione correttiva

La *manutenzione correttiva* corregge difetti non emersi in fase di collaudo. I difetti possono essere *di implementazione*, i meno costosi da correggere, *di progettazione* e *di specifica*, i più costosi in quanto potrebbe essere necessario riprogettare il sistema.

Manutenzione adattiva

Con *manutenzione adattativa* s'intende l'adattamento del sistema a cambiamenti di piattaforma.

Manutenzione migliorativa

Con *manutenzione migliorativa* s'intende l'aggiunta, cambiamento o miglioramento di requisiti funzionali e non funzionali, secondo le richieste del committente.

Costi di manutenzione

Molti sono i fattori che influenzano il costo di manutenzione:

- *dipendenza dei componenti*: la modifica dell componenti potrebbe avere ripercussioni sugli altri componenti
- *linguaggio di programmazione*: i programmi scritti con linguaggi ad alto livello sono più facili da capire e da mantenere
- *struttura del codice*: codice ben strutturato e documentato rende più facile la manutenzione
- *collaudo*: una fase di collaudo approfondita riduce il numero di difetti
- *qualità della documentazione*: una documentazione chiara e completa facilita la comprensione del sistema da mantenere
- *stabilità dello staff*
- *età del sistema*
- *stabilità del dominio dell'applicazione*: se il dominio subisce variazioni, il sistema deve essere aggiornato
- *stabilità della piattaforma*

5.2 Processo di manutenzione

Nel processo di manutenzione, bisogna innanzitutto *identificare l'intervento* da svolgere e analizzarne l'impatto sul sistema, identificando quali componenti e tests saranno impattati dalla modifica. In seguito sono *realizzate le modifiche*, aggiornando, se necessario, specifica, progettazione e/o implementazione. Dopo il collaudo, la nuova versione viene rilasciata.

CRF

Il *Change Request Form* (CRF) è un documento formale che descrive una modifica. È compilato dal proponente della modifica, gli sviluppatori e dalla Change Control Board.

5.3 Patch, versioni e release

Patch

Nel caso si presentino problemi che devono essere risolti in fretta, si richiede d'urgenza una *manutenzione d'emergenza*. Viene modificato direttamente il codice, applicando la soluzione più immediata, e si rilascai una versione aggiornata, detta *patch*.

Versioni

Una *versione* è un'istanza del sistema che differisce per qualche aspetto dalle altre istanze. Una versione è identificabile non solo da numeri di versione, ma anche dagli attributi

Release

Una *release* è una particolare verisone che viene distribuita a committenti/clienti. Una release comprende tutte le componenti software necessarie per far funzionare il sistema e la documentazione.

Configurazione software

La *configurazione software* è l'insieme di informazioni prodotto da un processo software. Include documentazione, codice e dati. Il *database delle configurazioni* contiene varie configurazioni software corrispondenti alle release di un sistema.

5.4 Sistemi ereditati e re-engineering

Un *sistema ereditato* è un vecchio sistema che deve essere mantenuto nel tempo. Sfruttano tecnologie obsolete e sono costosi e difficili da mantenere. È possibile *reimplementarli*, *re-ingegnerizzarli* ed infine *dismetterli*.

Il *forward engineering* è il processo software nel quale si crea un nuovo progetto partendo dalla specifica dei requisiti.

Il *re-engineering* è il processo mediante il quale un nuovo sistema nasce dalla trasformazione di uno vecchio, allo scopo di rinnovarlo e aumentare la mantenibilità. Le attività di re-engineering sono: traduzione del codice, ristrutturazione del codice, ristrutturazione dei dati.

Traduzione del codice

Con *traduzione del codice* s'intende il passaggio da un liguaggio di programmazione ad un altro, oppure ad una versione più recente di quel linguaggio.

Refactoring

Le modifiche fatte al sistema nel corso del tempo tendono a rendere il codice sempre meno ordinato. La *ristrutturazione* cerca di rendere il codice più semplice, leggibile, comprensibile e modulare; vengono aggiunti commenti, eliminate le ridondanze, uniformato lo stile di programmazione. Non modifica il comportamento del codice, ma ne aumenta la leggibilità.

Ristrutturazione dei dati

È il processo di riorganizzazione delle strutture dati.

Reverse engineering

Per sistemi ereditati, la documentazione può essere scarsa o assente, complicando le attività di re-engineering. Il *reverse engineering* è la generazione di documentazione partendo da codice e dati.

Reimplementazione del sistema

Quando il sistema è troppo mal strutturato per il re-engineering, è inevitabile la reimplementazione del sistema.

6 Gestione del progetto

Il processo software e la gestione si svolgono in