UNIVERSITÀ DI PADOVA  FACOLTÀ DI INGEGNERIA

Relazione di Tirocinio Breve

LA REINGEGNERIZZAZIONE DI APPLICAZIONI ACCESSIBILI VIA WEB: TRE CASI DI STUDIO

Laureando: Fabrizio Greggio

Relatore: prof. Maristella Agosti

Co-relatore: Ing. Simone Friso

Corso di laurea Triennale in Ingegneria Informatica

27/09/2010

Anno Accademico 2009/2010

ABSTRACT

Un Legacy system è un sistema informativo di valore ereditato dal passato. La sua importanza nell'organizzione da cui è utilizzato è elevata, ma a causa di vari fattori, i principali dei quali la sua età, la documentazione disponibile e il metodo con cui è stato progettato rendono la sua manutenzione difficile e onerosa.

Il Software Reengineering è il processo di manutenzione preventiva, che ha lo scopo di migliorare la futura manuntenibilità senza stravolgere le sue funzionalità e prevede le attività di traduzione del codice sorgente, reverse engineering, miglioramento della struttura del programma e la sua modularizzazione, e reingegnerizzazione dei dati. Un processo di software reengineering può essere costoso e addirittura inutile se prima non viene fatta una attenta analisi, poiché il costo di ogni fase va a sommarsi e a influenzare il costo delle altre.

Nello sviluppo di nuovo codice è necessario prevedere e organizzare una buona fase di test in quanto oltre a rendere migliore l'applicazione ne riduce i costi di realizzazione e manutenzione.

I casi di studio riguarderanno La Scuola di Dottorato del Dei, Moodle e Giga: Tre applicazioni accessibili via web nel dominio http://dei.unipd.it/ , alle quali sono stati applicati dei semplici processi di reingegnerizzazione.

Indice generale

1.INTRODUZIONE

Nell'ambito dello svolgimento dell'attività di servizio civile, trascorso nell'università degli studi di Padova al DEI (Dipartimento di Ingegneria dell'Informazione), dal gennaio 2009 a gennaio 2010 presso i Servizi di Laboratorio, sotto la responsabilità dell' Ing. Francesca Bettini e seguito dall'Ing. Simone Friso ho avuto modo di svolgere diverse mansioni. Il mio lavoro di tesi riguarda in particolar modo tre progetti perché hanno avuto una rilevante importanza nel corso delle mie attività e come risvolto nel dipartimento.

1.1 Organizzazione della tesi

Nel secondo capitolo della tesi si analizzeranno le problematiche che spingono un'organizzazione a reingegnerizzare un software Legacy, si capirà quando questa operazione è necessaria ed economicamente vantaggiosa prima definendo e elencando le caratteristiche di un software Legacy e poi descrivendo il processo di Reengineering nelle sue fasi principali .

Detto ciò si analizzerà la fase di Test nel ciclo di vita di una applicazione, motivando perchè è una operazione importante e descrivendo e commentando i vari tipi di test.

Infine si spiegherà quale metodo di lavoro è stato adottato.

Nel terzo capitolo si parlerà dei tre casi di studio di reengineering descrivendo per ogni applicazione software analizzata gli scopi per cui era stata progettata e le attività effettuate.

Nel quarto capitolo si trarranno le conclusioni sul lavoro svolto.

2.PROBLEMATICHE DEL REENGINEERING

In questo capitolo prima di tutto si spiegherà cos'è un Legacy system, poi si suddivideranno i vari tipi di Legacy in base al trattamento possibile a cui si possono sottoporre.

Detto ciò si definirà cos'è la manutenzione del software e si spiegherà perchè sia necessaria e importante all'interno del ciclo di vita del software.

Si prosegue definendo cos'è il software reengineering spiegando se e quando conviene, di quali fasi è composto un processo di reingegnerizzazione di un sistema/applicazione legacy e motivando la necessità di una attenta analisi a monte di esso.

Il capitolo procede poi trattando la fase di test nel ciclo di vita di una applicazione, spiegandone i motivi e gli scopi per cui viene impiegata.

Il capitolo termina descrivendo il metodo di lavoro adottato nel svolgere i casi di studio presentati nel terzo capitolo.

2.1 I Legacy systems

Si definisce Legacy system un sistema informativo di valore ereditato dal passato.

I concetti fondamentali sono appunto “di valore” (cioè critico per il business dell’organizzazione) e “ereditato dal passato” (generalmente 5 anni o più, ma comunque già operativo nel momento in cui lo si prende in considerazione).

Altre definizioni:

“Grandi sistemi software con cui non si vorrebbe avere a che fare ma che sono vitali per l’organizzazione” [Bennet],

“Ogni applicazione in produzione” [Schick]

“Ogni sistema informativo che resiste alle modifiche ed evoluzioni necessarie per tener dietro ai nuovi e mutevoli requisisti di business dell’organizzazione” [Brodie].

Le caratteristiche fondamentali di un Legacy System sono:

- È un sistema fondamentale per l’operatività dell’organizzazione; inoltre spesso è pesantemente utilizzato essendo mission-critical e dovendo quindi rimanere generalmente operativo al 100% e 24 ore su 24;
- Su di esso l’organizzazione ha pesantemente investito nel corso degli anni, e quindi non può essere semplicemente accantonato così com’è;
- E’ composto da moltissime righe di codice (anche centinaia di migliaia).
- Spesso il suo primo nucleo risale ad oltre un decennio fa ed è quindi progettato secondo vecchie concezioni;
- È scritto in linguaggio di vecchia generazione ;

- E' supportato da un DBMS obsoleto (ad es. database IMS - Information Management System di IBM), sempre che esista un DBMS e non si faccia ricorso al file system (file VSAM - Virtual Sequential Access Method);
- Le interfacce utente sono quelle a caratteri (terminali 3270 o 5250) e non grafiche,
- Le applicazioni che lo compongono sono prevalentemente monolitiche, ognuna è stato progettata e realizzata indipendentemente dal resto e quindi il sistema presenta un'integrazione prevalentemente verticale. Strutturalmente tale software applicativo è tipicamente suddiviso in transazioni (una funzione utente scritta in COBOL) usabili contemporaneamente da più utenti;
- Non è ben documentato ed è difficile da comprendere, in quanto, quasi sempre, la documentazione non è aggiornata con le modifiche che sono state via via apportate al software;
- Il sistema può essere considerato come un repository di anni di esperienza e pratiche aziendali non esplicitamente documentate, ma presenti e "immerse" nel codice stesso del legacy;

Dal punto di vista del trattamento, le applicazioni Legacy possono essere classificate come:

1. Altamente decomponibili, sono ben strutturati e presentano alcune caratteristiche fondamentali:

- I componenti applicativi sono separabili in logica di presentazione, logica applicativa e logica d'accesso ai dati, cioè il software è decomposto in tre livelli logici.
- I moduli applicativi sono indipendenti tra di loro (non ci sono interdipendenze gerarchiche).
- I moduli applicativi hanno interfacce ben definite con i servizi di database, quelli di presentazione e le altre applicazioni.

2. Data decomponibili, sono sistemi cosiddetti "semistrutturati" con le seguenti caratteristiche fondamentali:

- I componenti applicativi sono separabili in due livelli: i servizi d'accesso ai dati e quelli di presentazione e logica applicativa (fusi in un unico blocco).
- I moduli applicativi hanno interfacce ben definite verso le altre applicazioni. In questi sistemi è possibile accedere direttamente ai dati, ma non alla logica applicativa.

3. **Program decomponibili,** sono anch'essi "semistrutturati" con le seguenti caratteristiche:
   - I componenti applicativi sono separabili in due livelli: i servizi di presentazione e quelli d'accesso ai dati e logica applicativa (fusi in un unico blocco).
   - I moduli applicativi hanno interfacce ben definite verso le altre applicazioni.

   In questi sistemi non è possibile accedere direttamente ai dati, ma è necessario invocare delle funzioni predefinite (tipicamente una transazione). In questa categoria rientrano la maggior parte delle applicazioni legacy.
4. **Monolitici** (non strutturati), sono sistemi in cui tutti i componenti appaiono come un unico blocco in cui tutti i tre livelli logici sono fusi insieme. Generalmente a questi sistemi si può accedere solo attraverso l'invocazione da terminale.

Molte applicazioni in realtà hanno un'architettura che è una combinazione di queste quattro. Dal punto di vista della facilità di trattamento, i Legacy System possono essere distinti in:

- **Ostili:** sono quelli che non permettono la possibilità di interfacciamento con l'esterno.
- **Trattabili:** l'interfacciamento con altri sistemi risulta possibile con un certo sforzo di programmazione e tecnologie ad hoc.
- **Amichevoli:** l'interfacciamento con l'esterno è facilmente attuabile.

È evidente che i sistemi del primo tipo sono amichevoli, quelli Data/Program decomponibili risultano trattabili, mentre quelli dell'ultimo tipo rimangono ostili.

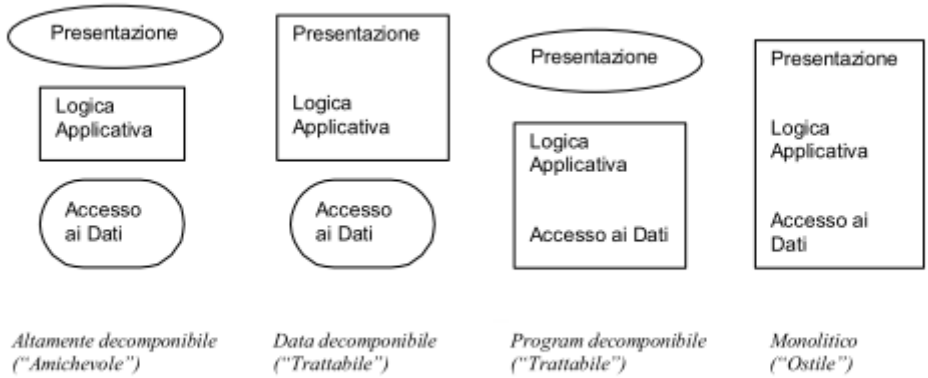

Figura 1: Le diverse categorie di Legacy System

2.2 La manutenzione del software

La manutenzione del software viene definita come " the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment" [ANSI/IEEE Std. 729-1983]

Per manutenzione si intende quindi la fase del ciclo di vita del software che comprende il complesso delle operazioni necessarie a conservare la conveniente funzionalità, efficienza e modificabilità del sistema dopo il rilascio del prodotto software.

In generale si può dire che la manutenzione è la base di tutti gli interventi che si possono operare su un sistema, sia perché la velocità con cui esso tende a diventare legacy dipende molto dall'adeguatezza e qualità del processo di manutenzione, sia perché spesso diventa poi difficile dare una chiara separazione tra intervento di manutenzione e di altro tipo. In linea di massima si può assumere che si è in presenza di manutenzione quando, nonostante le modifiche, il sistema rimane strutturalmente costante e continua a giocare lo stesso ruolo nell'ambito del contesto d'uso.

In base allo scopo, i tipi di manutenzione possono essere:

- **Manutenzione correttiva** (25%), che ha lo scopo di eliminare i difetti (fault) che producono guasti(failure) del sistema software, sfuggiti ai test finali prima della consegna al cliente..
- **Manutenzione adeguativa** (21%), che ha lo scopo di adattare il sistema software ad eventuali cambiamenti nel caso in cui le specifiche siano modificate in seguito a variazioni delle condizioni al contorno.
- **Manutenzione perfettiva** (50%), che ha lo scopo di estendere il software con funzionalità aggiuntive, migliorare quelle già in utilizzo ed eliminare quelle più obsolete..
- **Manutenzione preventiva o software reengineering**, che consiste nell'effettuare modifiche che rendano più semplici le future correzioni, e gli adattamenti per cambiare la struttura dell'applicazione mantenendone però intatta la funzionalità.

La manutenzione è un elemento strategico per il software, e ancora oggi, nonostante il miglioramento dei processi produttivi, rappresenta oltre il 70% della spesa per i sistemi informatici; inoltre una manutenzione non strutturata e caotica porta al rapido degrado di un sistema, e quindi alla sua totale ingestibilità (accelera il diventare legacy di un sistema).

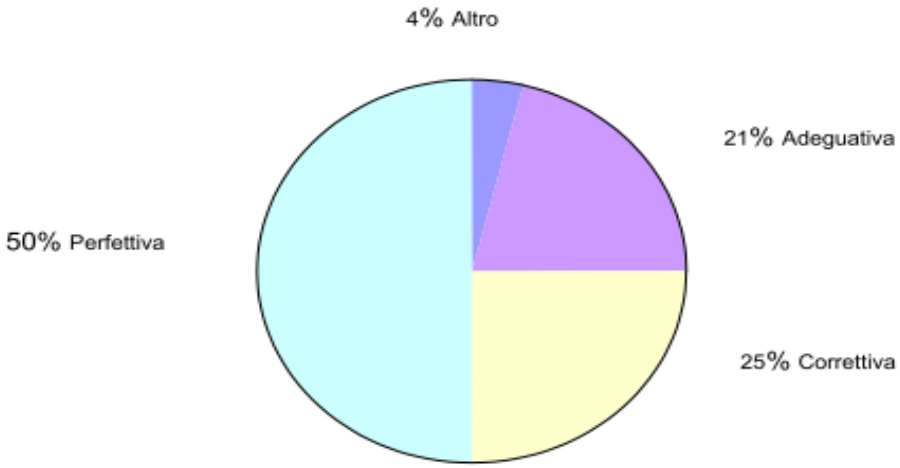

Figura 2: Incidenza sui costi dei vari tipi di manutenzione

La manutenzione ha tutta una serie di problematiche, dovute principalmente al fatto che è estremamente complicato capire il software realizzato da altri (in funzione inversa alla quantità di documentazione) e che l'autore del software in genere non è disponibile per collaborare alla risoluzione del problema. Inoltre, soprattutto nel caso di sistemi molto vecchi, la documentazione spesso non esiste o è fortemente inadeguata**.**

I motivi per cui i costi possono essere così alti sono i seguenti:

- Il personale addetto alla manutenzione non è formato o non ha familiarità nel campo dell'applicazione. Gli sviluppatori originari dell'applicazione legacy non sono facilmente rintracciabili.
- L'applicazione non essendo programmata con tecniche moderne può essere mal-strutturata e difficilmente comprensibile.
- I cambiamenti apportati al software possono non essere adeguatamente documentati e causare nuovi difetti software.

2.3 Software Reengineering

Come spiegato nel paragrafo precedente, il Software Reengineering è il processo di modifica di un software Legacy importante per l'organizzione, atto a migliorare le prestazioni e la struttura di esso, e a produrre nuova documentazione del software per rendere il codice più comprensibile, semplificando così le future manutenzioni.

Il costo della manutenzione del software Legacy aumenta proporzionalmente con l'avanzare del tempo, e la creazione di nuovo software ad hoc che vada a sostituire il Legacy molto spesso risulta avere costi proibitivi per l'organizzazione.

Il Software Reengineering dunque è senz'altro utile per aumentare la qualità, l'utilità, e il periodo di vita del software Legacy.

2.3.1 Software Reengineering vs Buisness Process Reengineering

Il termine Reengineering viene anche associato al termine Buisness Process Reengineering, ovvero il processo di ridimensionare e riorganizzare i processi produttivi di una azienda per aumentarne l'efficenza, migliorare la qualità dei beni e dei servizi offerti, e diminuire i costi dei fattori produttivi. In analogia con quanto detto in precedenza, la situazione dei processi aziendali da sottoporre a Reengineering corrisponde al software Legacy, ma a differenza del software, i nuovi processi possono essere modificati drasticamente, inglobando attività non pertinenti alla situazione precedente o cessando addirittura di esistere.

Molto spesso in una organizzazione dove l'ausilio del software è indispensabile al buon funzionamento dei processi aziendali, si evince che il Reengineering dei processi deve essere necessariamente affiancato a quello del software.

2.3.2 Software Reengineering vs Forward Engineering

Con il termine Forward Engineering si intende la creazione di una nuova applicazione che sostituisca quella Legacy esistente.

La grande differenza tra i due processi risulta nelle specifiche iniziali del sistema. Invece di partire con nuove specifiche, nel Software Reengineering è il sistema Legacy a fungere come specifica per il processo. L'attività fondamentale in questo caso consisterà principalmente nel comprendere il funzionamento del codice del Legacy e a modificarlo. Viceversa nel Forward Engineering il punto di partenza del processo sono le specifiche richieste: il lavoro poi consisterà nella progettazione e implementazione di nuovo codice.

Il Software Reengineering ha due grandi vantaggi rispetto alla radicale decisione di sviluppare una nuova applicazione:

1.Riduzione dei Rischi. Esiste un alto rischio associato allo sviluppo di nuovo codice per una grande applicazione importante ed essenziale per l'organizzazione. Possono essere fatti errori nelle specifiche della nuova applicazione, errori nello sviluppo, bug difficilmente rintracciabili ecc.

2.Riduzione dei Costi. Come già detto, dal lato economico risulta molto conveniente il Reengineering dell' applicazione rispetto alla creazione di una nuova. Dalle statistiche fatte, i costi di Reengineering risultano essere stimati inferiori a un quarto di quelli sostenuti per il Forward Engineering.

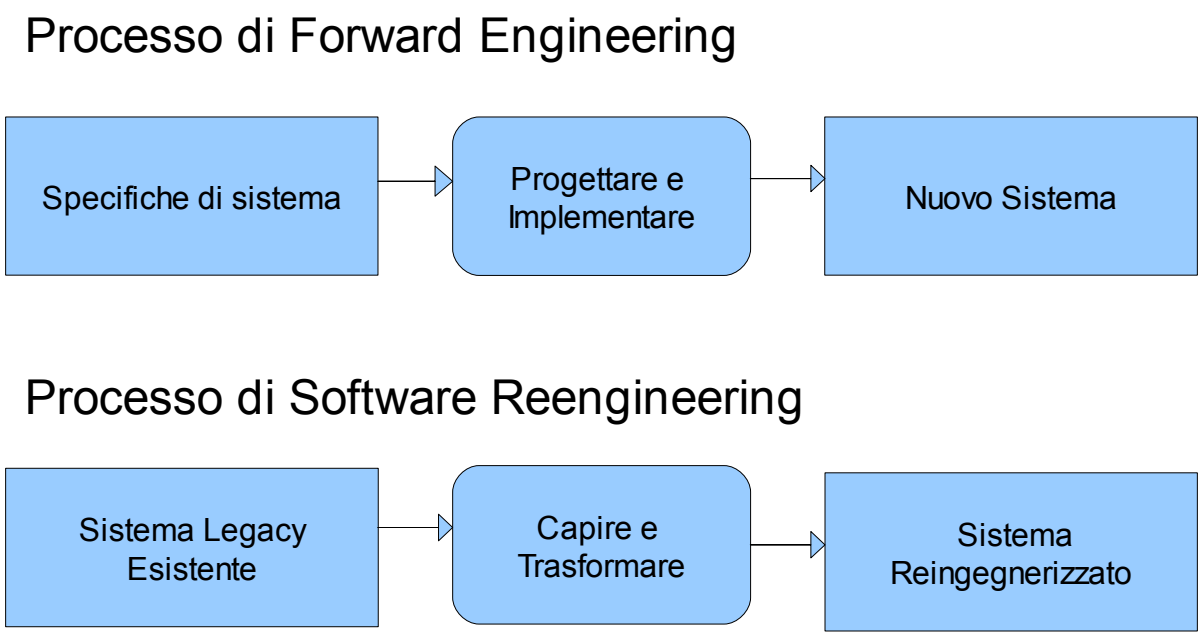

Figura 3: Forward Engineering vs Software Reengineering

Le fasi principali del processo di Reengineering schematizzato in precedenza sono le seguenti e verranno successivamente trattate:

1. Traduzione del codice: Il codice sorgente del programma viene tradotto da un vecchio linguaggio di programmazione a una nuova versione dello stesso linguaggio, oppure ad un linguaggio di programmazione più moderno.
2. Reverse Engineering: Il programma viene analizzato in profondità per ottenere utili informazioni finalizzate alla redazione di una documentazione della sua organizzazione e del suo funzionamento.
3. Miglioramento della struttura del programma: Le strutture di controllo del programma vengo migliorate per aumentarne la leggibilità e facilitarne il riuso.
4. Modularizzazione del programma: Le parti di codice relative alla stessa funzionalità vengono raggruppate e riorganizzate in modo da eliminare ridondanze dove possibile.
5. Reingegnerizzaione dei dati: I dati già acquisiti dall'applicazione vengono reingegnerizzati e trasformati per essere elaborati rispecchiando le nuove strutture dati del software reingegnerizzato.

Queste fasi elencate non sono ovviamente tutte necessarie per il processo di Reengineering. Per esempio se si fa uso di tool ad hoc, il Reverse Engineering viene spesso svolto automaticamente, oppure la Reingegnerizzazione dei dati deve essere svolta solo quando le strutture dati del programma vengono cambiate.

Tuttavia la Ristrutturazione del programma è sempre fondamentale.

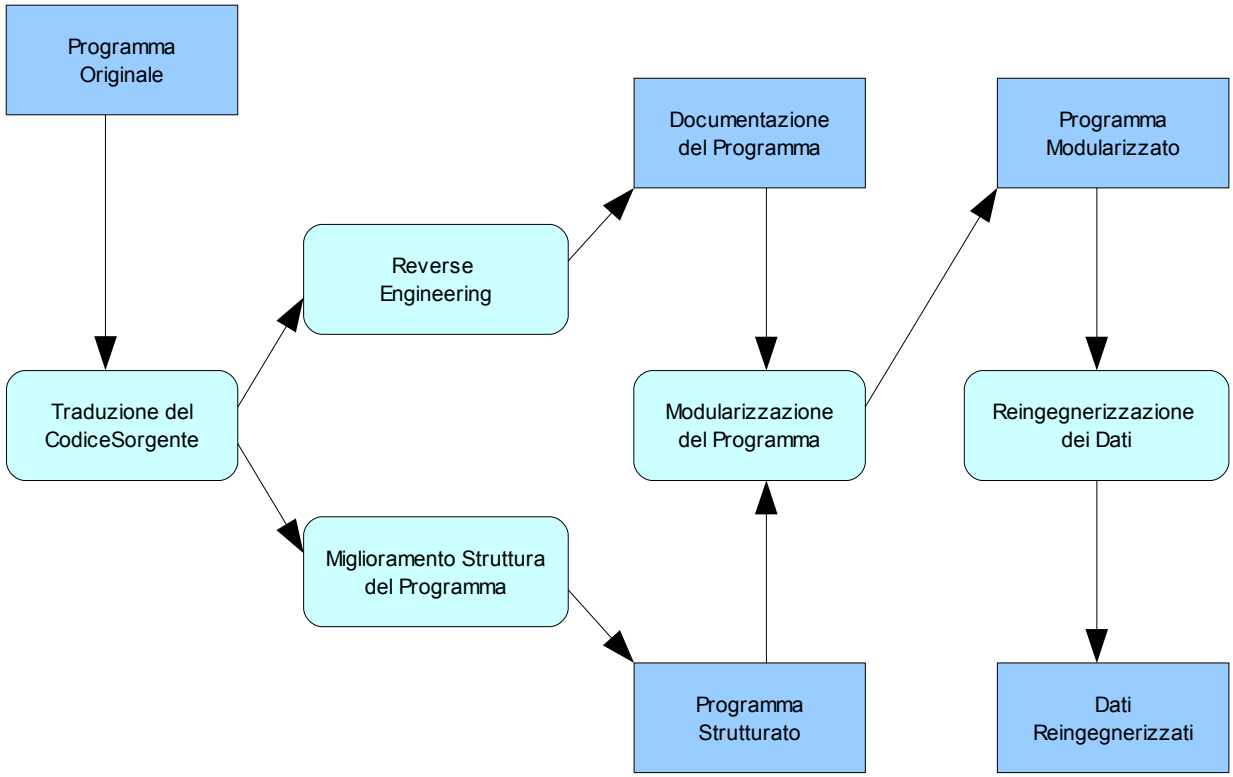

Figura 4: Processo di Software Reengineering

2.3.4 Le fasi del processo di Reengineering: Traduzione del codice sorgente

E' generalmente la fase più semplice di tutto il processo di Reengineering, e consiste nella traduzione, meglio se automatizzata tramite apposito software, del codice sorgente in un altro linguaggio.

La struttura del programma in sé, e la sua organizzazione non viene modificata. Il nuovo linguaggio utilizzato può essere lo stesso del sistema legacy con una versione aggiornata (es. Da COBOL-74 a COBOL-85), oppure completamente diverso (es. Da FORTRAN a C).La traduzione del codice può essere necessaria per i seguenti motivi:

1. Cambiamento della piattaforma hardware:

   L'organizzazione potrebbe avere la necessità di cambiare la struttura hardware del sistema, con la conseguente possibilità di non trovare un compilatore per il linguaggio del software Legacy funzionante con la nuova piattaforma hardware.

2. Carenza di conoscenze da parte dello staff:

   Se il linguaggio di programmazione usato dal software Legacy non è un linguaggio standard oppure un linguaggio caduto in disuso, lo staff addetto alla manutenzione potrebbe avere dei problemi.

3. Decisioni dell'organizzazione:

   Se l'organizzazione decide di standardizzare tutto il software ad un unico linguaggio di programmazione. Mantenere diversi compilatori per linguaggi diversi diventa più oneroso.

4. Mancanza di supporto per il linguaggio.:

   Se il linguaggio di programmazione usato dal sistema Legacy è andato in disuso e non è più presente sul mercato, viene a cessare il supporto per le applicazioni.

Affinché la traduzione del codice sorgente sia economica, è necessario utilizzare dei tools software o dei pattern matching systems per la traduzione del codice.

Purtroppo però la traduzione del codice non può avvenire tutta in automatico, per esempio vi possono essere delle operazioni condizionali in compilazione non previste dal nuovo compilatore, oppure delle istruzioni che non si possono tradurre direttamente nel nuovo linguaggio. In questi casi occorre tradurre manualmente il codice.

2.3.5 Le fasi del processo di Reengineering: Reverse Engineering

La fase di Reverse Engineering consiste nell'analizzare il software con l'obbiettivo di recuperarne le specifiche, la struttura, il funzionamento e l'originaria progettazione. L'obbiettivo di questa fase è quello di recuperare più informazioni possibili sul sistema Legacy riguardo tutte le modifiche fatte al codice, le astrazioni presenti nel software e le parti di codice riutilizzato oppure riutilizzabili.

A differenza delle altre fasi del processo di Reengineering, il programma in se non viene modificato. Generalmente l'input del processo di Reverse engineering è il codice sorgente, ma nel caso esso non sia disponibile, ad esempio perché andato perduto, si deve utilizzare il codice eseguibile. Il processo di Reverse Engineering, come già detto, è diverso da quello di Reengineering. Il primo ha lo scopo di ricavare la documentazione riguardo le specifiche, la struttura, il funzionamento e la progettazione partendo dal codice sorgente; il secondo invece, partendo dalle conoscenze ottenute dal primo, produce nuovo codice, più mantenibile rispetto a quello Legacy.

Da quanto detto, si può evincere che il processo di Reverse engineering è necessario per un buon Reengineering dell'applicazione, viceversa non è detto che dopo la fase di Reverse Engineering siano necessarie le altre fasi:

1. Le conoscenze acquisite dalla fase di Reverse Engineering vengono utilizzate come input per la modifica del codice Legacy.
2. Le conoscenze acquisite dalla fase di Reverse Engineering sono ora disponibili e sufficienti al personale addetto per una corretta e meno dispendiosa attività di manutenzione.

Anche la fase di Reverse Engineering può essere svolta completamente o meno da tools che automatizzano e riducono notevolmente il tempo da dedicare a questa fase. L'output del processo di Reverse engineering può essere costituito da grafici, diagrammi anche collegati al codice per far vedere il sistema Legacy da diversi punti di vista, annotazioni personali, stime e statistiche.

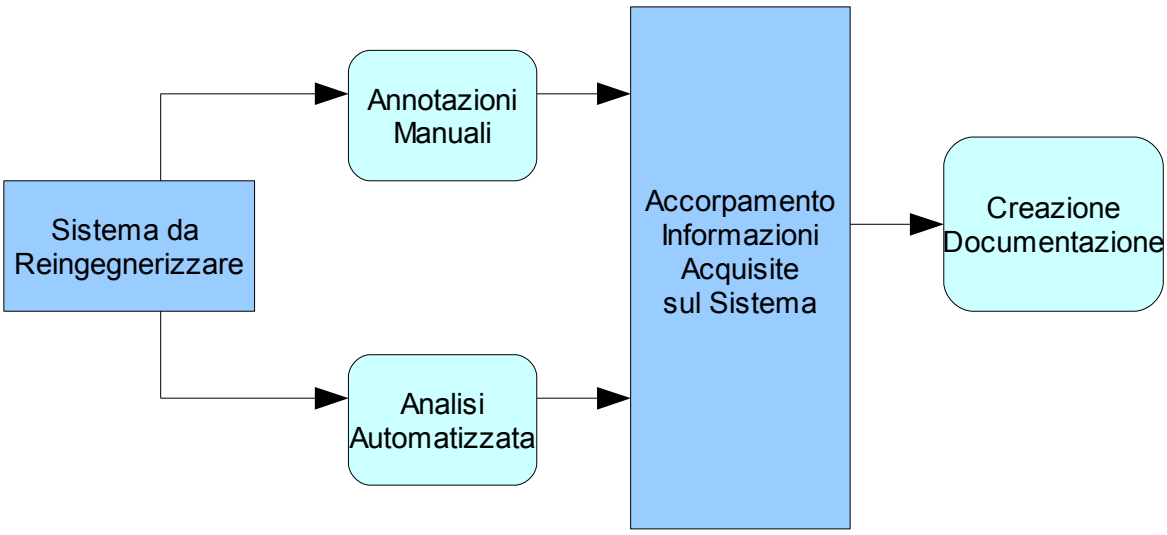

Figura 5: Processo di Reverse Engineering

2.3.6 Le fasi del processo di Reengineering: Miglioramento della struttura del programma

La necessità di migliorare la struttura di un programma Legacy deriva principalmente dai svariati interventi manutentivi che ha subito. Infatti generalmente i programmatori per non rischiare di creare dei bug o dei system faults si limitano solamente a modificare o inserire nuove clausole condizionali o cicli nella porzione di codice di loro interesse, senza preoccuparsi della logica globale di controllo.

Ne consegue che dopo diversi interventi di manutenzione la struttura di controllo di anche un semplice codice può risultare incomprensibile. Se poi si aggiunge il fatto che non sempre il nome delle nuove variabili introdotte è intuitivo, e che se ci sono limiti di memoria, i programmatori devono riuscire a riutilizzare più codice possibile: si intuisce l'importanza della ristrutturazione del codice.

La ristrutturazione del codice può essere effettuata manualmente, oppure tramite degli appositi tools. La ristrutturazione automatica del codice si basa sul teorema di Bohm-Jiacopini che afferma che ogni programma può essere riscritto esclusivamente mediante semplici clausole if-then-else e cicli loop, rendendo non necessari i salti incondizionati che sono causa di incomprensioni del funzionamento del codice.

I problemi principali della ristrutturazione automatica sono i seguenti:

1. Perdita dei commenti. Nel caso in cui siano stati inseriti dei commenti nel codice, una gran parte di essi sarà persa.
2. Perdita della documentazione. Cambiando la struttura del codice molto probabilmente la documentazione esistente non sarà più attendibile.
3. Problemi di complessità algoritmica. Anche se si hanno a disposizione macchine moderne e hardware con grande capacità di elaborazione, il tempo necessario a ristrutturare automaticamente un grosso Software Legacy sarà molto elevato.
4. Linguaggio di programmazione non standard. Se il codice non è stato scritto con un linguaggio di programmazione standard, il software non riesce ad ottenere buoni risultati e quindi è necessario passare alla ristrutturazione manuale.

I primi due problemi non sono molto significativi in quanto riguardano il vecchio codice. Il terzo dipende dalla grandezza del software Legacy, ma la ristrutturazione automatizzata risulta sempre conveniente in termine di costi a maggior ragione. Il problema principale che porta alla ristrutturazione manuale è il quarto elencato.

Nel caso in cui il codice sia fortemente accoppiato con strutture dati condivise, la ristrutturazione del codice non porta ad un miglioramento di leggibilità e comprensione. Talvolta purtroppo ristrutturare l'intero programma non porta ad un gran beneficio in termini di costi e di risultato. Si può scegliere quali parti di codice sia opportuno ristrutturare secondo 3 metriche:

1. Failure rate, ovvero la percentuale di systems fault
2. Percentuale di codice modificato su base annua, ovvero quanto frequente è il lavoro di manutenzione sul software Legacy.
3. Complessità dei componenti.

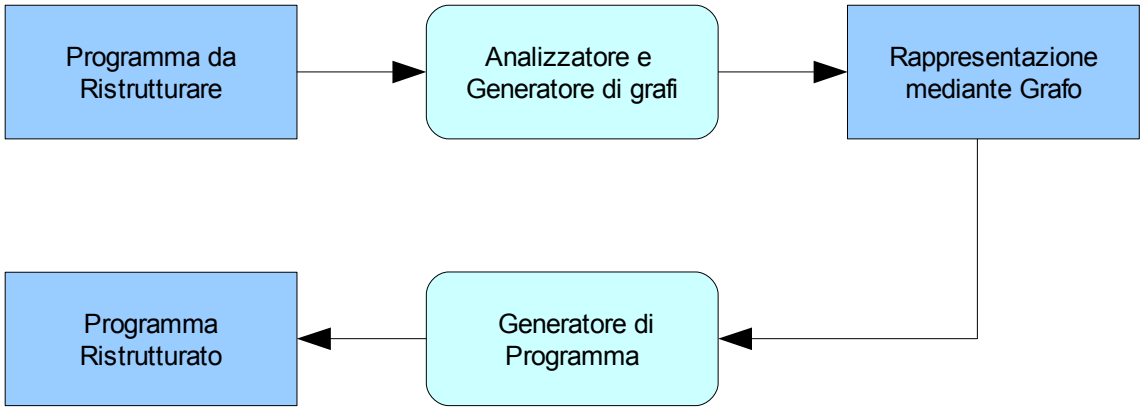

Figura6: Processo Automatizzato di Ristrutturazione del Programma

2.3.7 Le fasi del processo di Reengineering: Modularizzazione del programma

La modularizzazione di un programma è il processo di riorganizzazione di esso in modo che le parti di codice collegate tra loro logicamente e funzionalmente siano accorpate assieme, in modo da essere considerate come un un singolo modulo.

Una volta suddiviso il programma in moduli , risulterà più semplice eliminare le ridondanze e ottimizzare le interazioni tra essi, in più si potrà anche migliorare le loro interfacce con il resto del programma. In questa fase si possono creare diversi tipi di moduli a seconda della loro funzione, della dimensione e dei criteri di protezione e astrazione adottati.

I principali sono:

1. Astrazione dei dati. Tipo di dato astratto creato dall'associazione di dati e parti elaborate.
2. Moduli Hardware. Sono legati all' astrazione di dati. Contengono tutte le procedure per il funzionamento di una particolare periferica hardware.
3. Moduli funzionali. Questi moduli contengono tutte le procedure relative a una funzione o per una stessa operazione sui dati.Per esempio si può creare un modulo che contiene tutte le funzioni per l'input e la validazione di esso.
4. Modulo di supporto ai processi. In questi moduli vengono contenute tutte le funzioni necessarie gli scopi del software.

2.3.8 Le fasi del processo di Reengineering: Reingegnerizzazione dei dati

Finora le problematiche affrontate e gli interventi necessari erano causati dal cambiamento del codice o dell'hardware del programma.

Talvolta però, in un processo di Reengineering ci si imbatte anche in problemi legati all'evoluzione dei dati. Infatti in un Legacy System la memorizzazione, l'organizzazione e il formato dei dati utilizzati possono diventare obsoleti per le modifiche apportate nel processo di traduzione del codice o nella modularizzazione di esso.

Il processo di riorganizzazione delle strutture dati e talvolta la modifica dei valori presenti in un sistema per renderlo più comprensibile, è detto reingegnerizzazione dei dati (Data-Reengineering). All'inizio questo processo può sembrare non necessario, eppure vi sono diverse ragioni per cui lo diventa. Le principali sono:

1. Decomposizione dei dati. Col passare del tempo la qualità dei dati tende a decadere. Questo è dovuto alle modifiche dei dati che provocano errori che si possono ripercuotere a catena, duplicazioni di valori non previste, cambiamenti del mini-mondo di interesse che non rispecchia più i dati. Questo è inevitabile a causa del tempo che passa.
2. Limiti inerenti la creazione del programma. Quando i programmatori hanno progettato il sistema possono aver inserito delle costanti sul numero massimo di dati da poter memorizzare o elaborare contemporaneamente. Purtroppo col passare del tempo cambiano le esigenze dell'organizzazione.

3. Evoluzione dell'architettura del sistema. Dopo che un sistema centralizzato è migrato ad una architettura distribuita, è necessario che il core del sistema di gestione dei dati possa accedere ai dati remoti. Perciò è necessario che siano reingegnerizzate le strutture dati, dai file separati al database del server remoto.

In base ai motivi per cui è necessario reingegnerizzazione i dati esistono tre diversi tipi di approccio:

1. Pulizia dei dati. I dati vengono analizzati e viene aumentata la qualità di essi eliminando ridondanze e applicando un formato coerente a tutti i record. Generalmente questo non necessita nessun cambiamento al programma.
2. Estensione dei dati. Viene effettuato il Reengineering di tutto il sistema per eliminare i vincoli riguardanti l'elaborazione dei dati come per esempio la lunghezza dei campi o il numero massimo di dati che si possono memorizzare.
3. Migrazione dei dati. In questo caso i dati vengono trasportati e messi sotto il controllo di un moderno DBMS

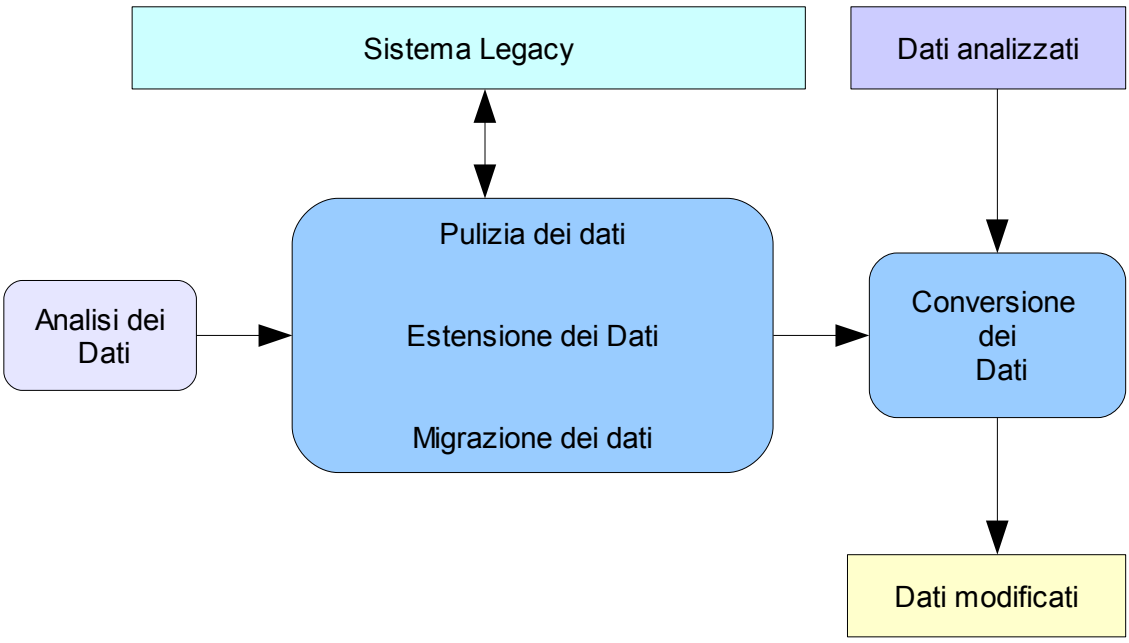

Figura 7: Processo di Reingegnerizzazione dei Dati

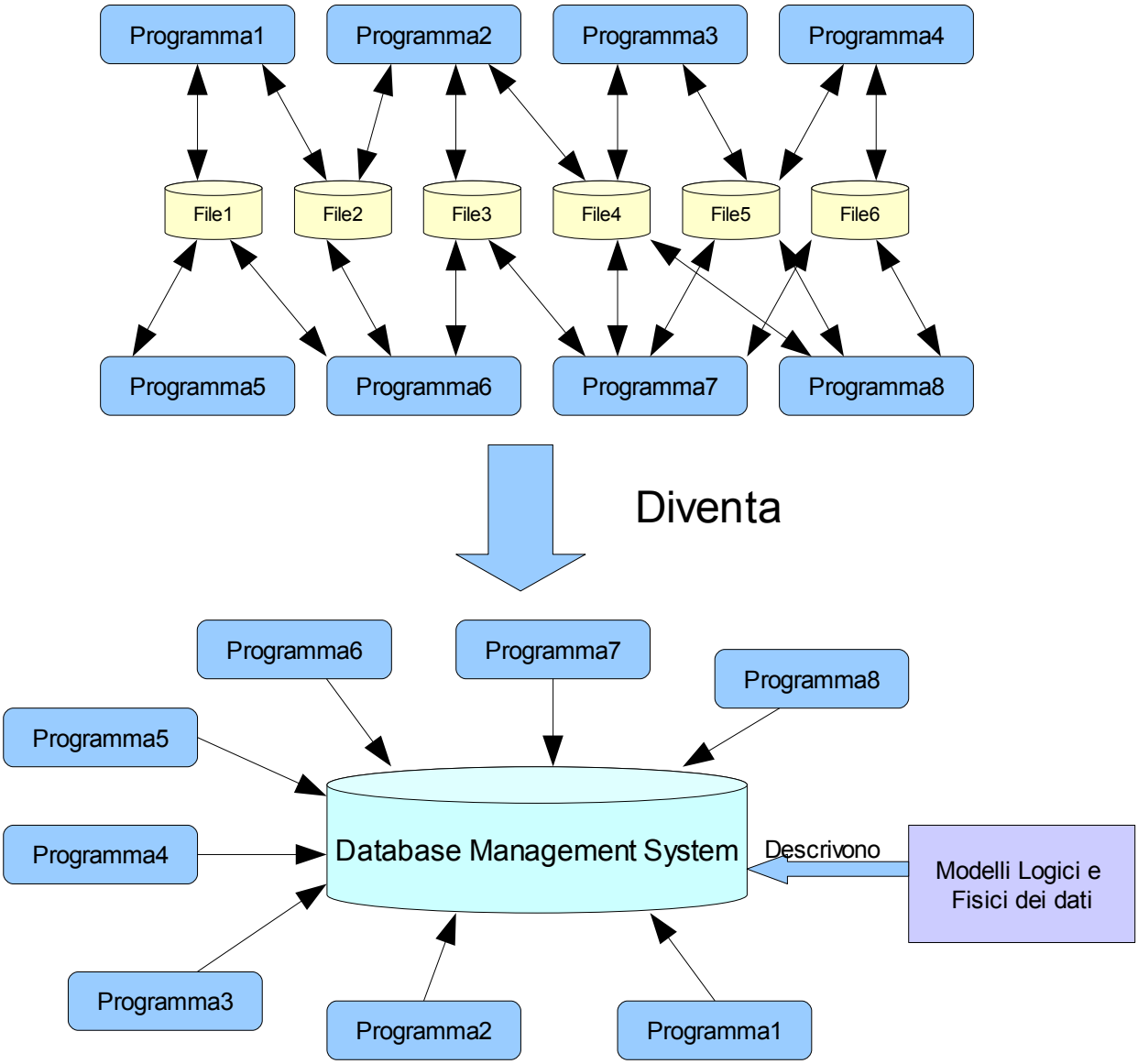

Figura 8: Migrazione dei dati

2.3.9 Costi del Software Reengineering

Dopo quanto spiegato, si evince che i costi dell'attività di Reengineering sono direttamente proporzionali alla mole di lavoro richiesta, che a sua volte viene influenzata dalle scelte effettuate in fase di pianificazione del lavoro. Principalmente i fattori che incidono sui costi sono.

1. La qualità del software esistente. Minore è la qualità del software e la sua relativa documentazione e maggiori saranno i costi sostenuti.
2. I tools di supporto disponibili. Automatizzando il più possibile il processo di Reengineering con CASE tool si riducono i costi rispetto alle modifica manuale di essi.
3. La mole di dati da reingegnerizzare. Maggiore è la mole maggiori saranno i costi.
4. Le capacità del team di lavoro. Se il team di lavoro non è composto da sviluppatori o manutentori dell'applicazione Legacy, i costi aumenterenno in quanto sarà necessario investire maggior tempo nella comprensione dell'organizzazione e del funzionamento di essa.

2.3.10 Limiti e svantaggi del Software Reengineering

Lo svantaggio principale del processo di Software Reengineering riguarda il limite pratico dei miglioramenti ottenibili mediante la reingegnerizzazione del sistema Legacy. E' impossibile per esempio convertire una applicazione scritta completamente in linguaggio procedurale in una equivalente in linguaggio orientato agli oggetti. I cambiamenti di architettura del sistema o la radicale riorganizzazione del sistema di data management, devono essere svolti manualmente e questo comporta un aumento notevole dei costi.

Il secondo limite è dato dal fatto che, sebbene il sistema reingegnerizzato sia notevolmente più efficiente e di più facile manutenzione rispetto a prima, quasi sicuramente un sistema scritto e progettato secondo moderne tecniche di software engineering risulterebbe senza dubbio ancora più efficiente e la manutenzione sarebbe ancora più facile. Per questo prima di iniziare il processo di Reengineering è necessario analizzare attentamente se esso sia o meno conveniente.

2.4 La fase di Test

L'attività di test costituisce la fase più critica dell'intero ciclo di vita del software: dallo unit test, effettuato dal programmatore sul componente appena sviluppato, al system test effettuato dal capo progetto sull'intera applicazione, qualsiasi carenza o superficialità può indurre gravi inaffidabilità del software. Per disporre di un software robusto e pronto a successive evoluzioni è sempre indispensabile un test accurato. Infatti solo l'esecuzione di un esauriente piano di test, dopo ogni modifica o evoluzione, è in grado di fornire garanzie sufficienti sulla non regressione del software in uso. Tuttavia, a causa del suo elevato costo (può incidere anche per il 50% sul costo del prodotto software), il test è spesso eseguito in modo superficiale ed incompleto, rendendo così inaffidabile il software ed inducendo una continua crescita dei costi di manutenzione.

La fase di test e collaudo serve a verificare che il prodotto software sia corretto.

Il test e collaudo è costituito a sua volta da due fasi.

- Verifica (verification) che viene svolta all'interno dell'azienda del produttore e che vuole scoprire se il prodotto realizzato funziona correttamente. Viene confrontato con il documento delle specifiche e con il progetto di dettaglio. La domanda alla quale si cerca di rispondere è: "Are we building the product right?".
- Certificazione (Validation), si tratta in pratica dell'ultimo test. Viene svolta nell'ambiente finale, nell'azienda del cliente, davanti al cliente. Si verifica che sia stato ottenuto il prodotto richiesto. Viene fatto il confronto con il documento dei requisiti. La domanda alla quale si cerca di rispondere è "Are we building the right product?".

Vi sono diverse tecniche per verificare un prodotto software.

- Tecniche statiche. Viene verificata la correttezza del programma senza eseguirlo,attraverso:
  - ✗ Verifiche informali, in base all'esperienza.
  - ✗ Verifiche formali, utilizzando tecniche matematiche.

L' ispezione del codice può apparire come una tecnica primitiva, ma invece è molto efficace nel rilevamento degli errori.

- Tecniche dinamiche e testing. Il codice viene eseguito su opportuni dati campione per individuare discrepanze tra il comportamento atteso e quello effettivo.

Una verifica dinamica può rilevare la presenza di errori, ma non la loro assenza.

Pertanto non si può avere la certezza assoluta della correttezza del prodotto software:

- La prova di correttezza può a sua volta essere sbagliata
- Può essere provata la correttezza relativamente alle specifiche funzionali, ma non si può stabilire se le specifiche non funzionali (ad esempio prestazioni) siano state soddisfatte.

Queste tecniche non vanno utilizzate in modo esclusivo ma è vantaggioso integrarle in quanto sono tecniche complementari.

2.4.1 Gli obbiettivi e i livelli della fase di Test

L'obiettivo di questa fase non deve essere quello di far vedere che il software non contiene errori, bensì di trovarne il maggior numero possibile. Infatti per grandi programmi è utopistico aspettarsi che il codice venga prodotto senza errori.

Per raggiungere questo scopo non è opportuno far testare il codice allo stesso team che lo ha prodotto, infatti si assegna di solito un team specifico per il test.

La fase si effettua a vari livelli e consiste in operazioni distinte.

- Test di unità. Una unità è il più piccolo blocco di software che ha senso collaudare, ad esempio una singola funzione di basso livello viene verificata come un'entità a se stante.
- Test di modulo. Un modulo è un insieme di unità interdipendenti.
- Test di sottosistema. Un sottosistema è un aggregato significativo di moduli spesso progettati da team diversi, e quindi vi possono essere problemi di interfaccia.
- Test di sistema o integrazione. Si tratta del test del prodotto completo.
- α-test. Se il sistema è sviluppato per un unico cliente, viene portato nel suo ambiente finale e collaudato con i dati sui quali dovrà normalmente operare.
- β-test. Se il sistema viene distribuito ad una comunità di utenti, viene dato in prova a più utenti, che lo utilizzano e forniscono al produttore le loro osservazioni ed il rilevamento degli errori riscontrati.
- Benchmark. Il sistema viene testato su dati standardizzati e di pubblico dominio per il confronto con altri prodotti equivalenti già presenti sul mercato. Può venire richiesto per contratto.
- Stress testing. Si verifica come si comporta il sistema quando è sovraccarico di lavoro, portandolo al limite. Questo test sotto sforzo permette di causare un errore (da stress) e verificare che il sistema fallisca in un modo accettabile (fail-soft).

Testing e debugging sono due concetti diversi. Il Testing serve a rilevare la presenza di un errore. Il Debugging consiste nella localizzazione dell'errore, della sua analisi e della sua correzione. Terminato il debugging si fa seguire il test di regressione: si verifica il comportamento del modulo che è stato corretto nei confronti dei moduli con i quali coopera, assicurandosi così che la correzione dell'errore non ne abbia introdotti di nuovi.

2.4.2 La distribuzione degli errori nella manutenzione del software

Esistono molti studi sulla distribuzione di errori i quali hanno dimostrato che gli errori tendono a concentrarsi in certi moduli, generalmente quelli più “complessi”. Un modulo che presenti molti errori va quindi verificato con attenzione, probabilmente ne conterrà altri: secondo alcuni studi statistici la probabilità che vi siano altri errori “latenti” cresce con il numero di errori già trovati secondo una curva di tipo “logistico”.

Figura 9: Rappresentazione della probabilità dell'esistenza di nuovi errori nell'asse delle ordinate in rapporto agli errori già trovati (espressi in decine) nelle ascisse.

Nell’attività di manutenzione, sia rivolta a modifiche evolutive sia alla correzione di errori, si introducono nuovi errori. Infatti molte delle decisioni prese inizialmente dal programmatore non vengono documentate, non sono evidenti dal codice, altre decisioni prese successivamente in fase di modifica tendono ad essere dimenticate da chi le ha fatte, oltre ad essere sconosciute agli altri.

Modificare una parte di codice scritto da altri, o semplicemente “datato”, con lo scopo di eliminare errori, porta facilmente ad introdurre ulteriori errori. La curva tipica degli errori in funzione del tempo di manutenzione, tende spesso a seguire una curva di tipo quasi parabolico: Inizialmente si ha una riduzione del numero di errori, al passare del tempo di manutenzione i continui interventi tendono ad introdurre nuovi errori. Quando un modulo è soggetto a continua manutenzione va programmata una riprogettazione per ottimizzare manutenibilità e la estendibilità.

Figura 10: Errori introdotti dalla manutenzione del software(espressi in unità) nelle ordinate, in rapporto al tempo di manutenzione(espresso in semestri) nelle ascisse.

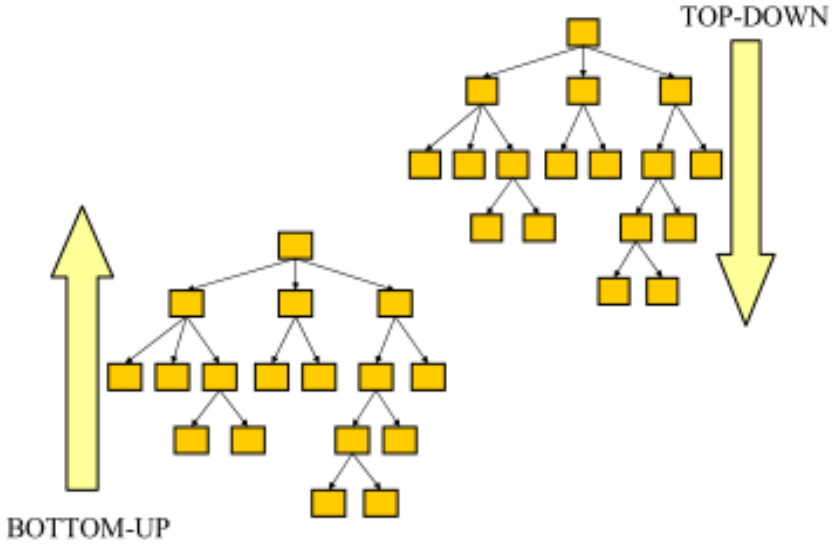

Figura 11: Strategie di Test Bottom Up e Top Down a confronto

2.4.3 Strategia di Test

Come la progettazione anche il test può essere effettuato top-down oppure bottom-up.

Con la strategia bottom-up vengono prima collaudati i moduli di più basso livello nella gerarchia prodotta dalla progettazione, si tratta cioè delle unità più piccole ed elementari che compongono il programma.

Quando sono valutati corretti, si passa al livello superiore che viene testato utilizzando le funzionalità del precedente livello, si risale quindi così fino al sistema intero.

Si tratta di un approccio che richiede alcune complicazioni: generalmente sono necessari dei driver, cioè del software ausiliario che simula la chiamata ad un modulo generando dati da passare come parametri. I driver fanno le veci della parte di codice non ancora integrata nella porzione testata del programma: la parte alta della gerarchia dei moduli.

Vantaggi e svantaggi della strategia bottom-up:

- Il processo di test ed integrazione delle unità è intuitivamente più semplice da realizzare .
- Più tardi viene scoperto un errore, più è costoso eliminarlo, in quanto richiede la sua correzione, solitamente con un modulo foglia e la ripetizione parziale del test fino ai moduli nei quali è stato trovato l'errore.

L'approccio top-down si comporta in modo diametralmente opposto. Inizialmente si testa il modulo corrispondente alla radice, senza utilizzare gli altri moduli del sistema. In loro vece vengono utilizzati dei simulatori, detti stub, si tratta di elementi che possono restituire risultati casuali, richiedere i dati al collaudatore, possono essere una versione semplificata del modulo.

Dopo aver testato il modulo radice della gerarchia si integrano i suoi figli diretti simulando i loro discendenti per mezzo di stub. Si prosegue in questo modo fino a quando non vengono integrate tutte le foglie della gerarchia.

Vantaggi e svantaggi della strategia top-down.

- Vengono scoperti prima gli errori più costosi da eliminare. Ciò risulta particolarmente utile se la progettazione è anch'essa di tipo top-down e le fasi di progettazione, realizzazione e test sono parzialmente sovrapposte: un errore di progetto nella parte alta della gerarchia può essere scoperto e corretto prima che venga progettata la parte bassa
- In ogni istante è disponibile un sistema incompleto ma funzionante
- Gli stub possono essere costosi da realizzare

Come già per la progettazione, viene solitamente adottata una soluzione di compromesso tra le due strategie, per attenuare gli inconvenienti che ognuna presenta.

Con entrambe le metodiche non è conveniente integrare ad ogni passo il maggior numero possibile di moduli. Ad esempio in una strategia bottom-up testare i figli di un modulo e poi integrarli tutti assieme con il padre per testare questo, porta a codici non funzionanti senza una precisa idea di dove sia l'errore. Risulta invece conveniente introdurre i moduli testati uno alla volta, si riesce in questo modo a localizzare meglio gli errori. In generale si osserva che è più conveniente effettuare test frequenti e brevi.

2.5 Metodo di lavoro

Il metodo di lavoro che si è adottato nello svolgimento delle attività è schematizzato nella figura seguente.

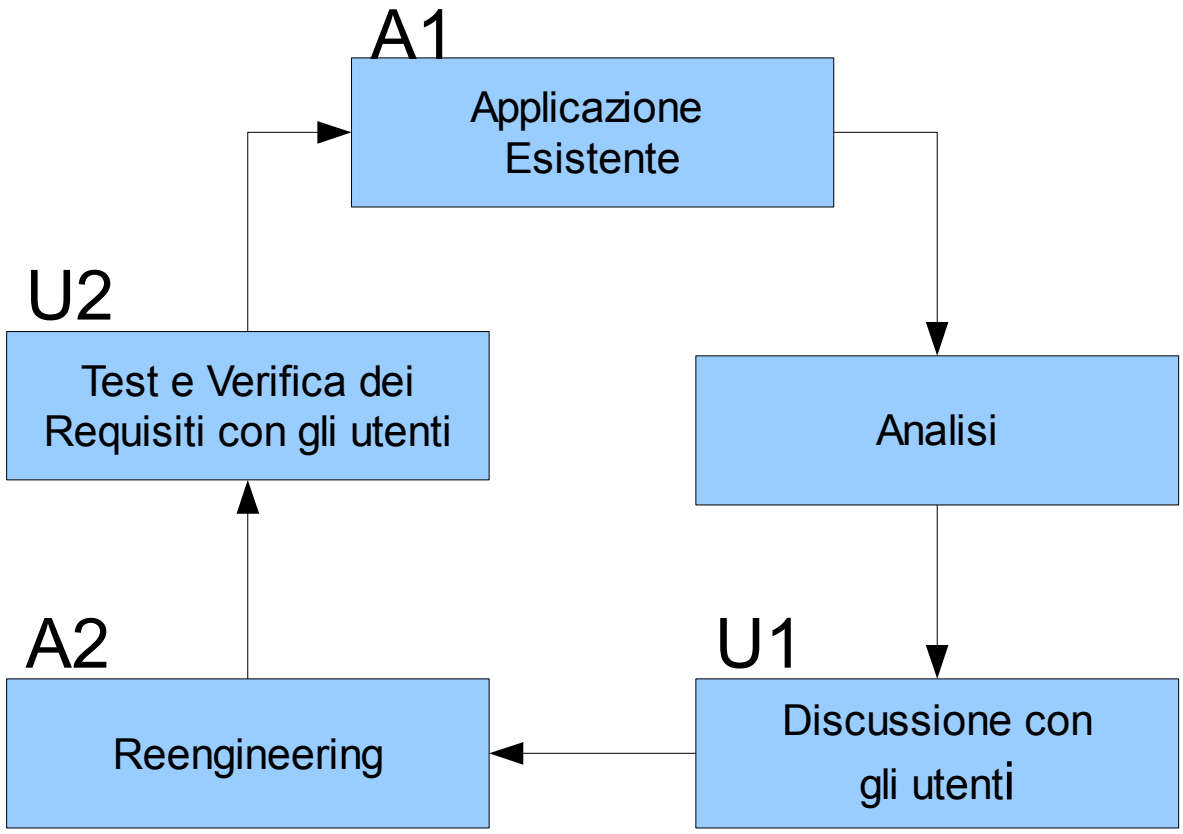

Figura 12: Schema del metodo di lavoro adottato

Il processo è a ciclo continuo ed è costituito dalle seguenti fasi:

- A1, Applicazione esistente. E' la fase di partenza del processo. Essa indica lo stato attuale dell'applicazione esistente così come viene consegnata.

- Analisi. E' la seconda fase, ed è molto importante. L'Output di questa fase è dato dall'insieme delle funzionalità e dall'organizzazione del sistema di calcolo esistente, ottenute sfruttando il più possibile: :

  - ✗ Documentazione e manuali esistenti

  - ✗ Interviste con gli utilizzatori per capire come funzionali

  - ✗ Analisi ed ispezione del codice

  - ✗ Reengineering manuale o coadiuvato da appositi Tool

- U1 Discussione con gli utenti. Questa fase consiste nell'analisi dei requisiti d'utente. Essa ha lo scopo di ricavare nel modo più chiaro ed esauriente possibile tutte le richieste e le specifiche degli utenti. Ciò si ottiene attraverso interviste e incontri formali e non con tutti gli utenti.

- A2 Reengineering. Questa fase, dopo le opportune analisi, prevede il reengineering del sistema di calcolo nelle modalità discusse in precedenza. Come Output si ottiene la nuova applicazione reingegnerizzata.

- U2 Test e verifica dei requisiti con gli utenti. In questa fase vengono effettuati gli ultimi test e viene consegnato il prodotto all'utente. Dall'esito di questa fase dipende il proseguimento del processo:

  1. Se i test sono soddisfacenti e gli utenti hanno trovato soddisfazione a tutte le loro esigenze, il ciclo termina e l'applicazione reingegnerizzata viene ufficialmente consegnata e messa in funzionamento.

  2. Se i test non sono soddisfacenti oppure gli utenti hanno nuove esigenze non espresse in precedenza, il ciclo continua tornando alla prima fase. Questa volta però si partirà non più con l'applicazione A1, ma con l'applicazione A2.

Questo metodo di lavoro a differenza di altri (per esempio a cascata) è stato adottato perché rispecchia meglio il continuo mutare di una applicazione.

3.CASI APPLICATIVI

In questo capitolo si analizzeranno i tre casi di studio di interesse, descrivendo per ognuno di essi prima il loro utilizzo e poi analizzando come è stato svolto il lavoro di manutenzione e reingegnerizzazione. Nell'analisi dei requisiti dei lavori, non essendoci un team di lavoro composto da molte persone, al massimo due, non era possibile decidere di applicare dei processi di reingegnerizzazione avanzati o complessi.

3.1 Scuola di Dottorato

3.1.2 Cos'è l'applicazione Scuola di Dottorato

L'applicazione accessibile via web Scuola di Dottorato, è una applicazione realizzata da uno studente laureatosi nel 1993, con lo scopo di poter gestire numerose informazioni riguardanti gli studenti alla scuola di dottorato al DEI e i loro docenti, tra cui il lavoro svolto, le loro trasferte, le loro valutazioni e molte altre ancora. All'applicazione possono accedere gli studenti e i docenti per regolare i dati del loro profilo, ma soprattutto le segretarie di dipartimento che periodicamente hanno l'obbligo di registrare delle statistiche riguardanti i corsi della Scuola stessa. L'applicazione risiede in un server al DEI, utilizza un Database Postgress e un server web Apache. E' stata scritta in php ed è composta da un discreto numero di pagine. E' accessibile dall'interno della rete dipartimentale seguendo il link alla pagina http://www.dei.unipd.it/dottorato .

Figura 13: Finestra di Login dell'applicazione web Scuola Di Dottorato

3.1.2 Le attività svolte

La richiesta di svolgere questo lavoro è pervenuta dalle segretarie di dipartimento in quanto chiedevano di poter avere dei miglioramenti nell'applicazione. Quando è stata progettato il database non esisteva l'esigenza di tenere traccia dei giorni di mobilità o permanenza all'estero degli studenti, nella realizzazione applicazione non era stato progettato un sistema per filtrare i risultati quando si effettuava una ricerca e alle segretarie era richiesti dei dati e delle stime diversi da quelli attuali. Il lavoro è stato quindi così strutturato:

1. Reverse Engineering dell'applicativo. La documentazione a disposizione era poca ed è stato necessario avvalersi dell'ispezione del codice.

2. Analisi dei requisiti richiesti dalle segretarie. S sono svolti degli incontri che hanno coinvolto anche il direttore della Scuola di Dottorato per definire quali erano le specifiche da realizzare. In base a ciò si è visto che le richieste erano fattibili e si è deciso di realizzarle.

3. Scrittura del codice. Si è deciso di mantenere la struttura di programma già utilizzato in quanto rivoluzionare sarebbe stato troppo complesso e avrebbe richiesto troppo tempo. La scrittura del codice riguardava sia la realizzazione delle nuove funzionalità, che la modifica dell'applicazione esistente per eliminare alcuni problemi di visibilità o di compatibilità tra diversi browser.

4. Incontri di dimostrazione e colloqui con le segretarie per verificare che fossero soddisfatte le richieste.

3.1.3 Problemi riscontrati

I problemi principali hanno riguardato la comprensione delle specifiche in quanto non è risultato subito chiaro quali fossero le necessità e le funzionalità richieste . Per ottenere maggior chiarezza è stato necessario svolgere più incontri. Un altro problema ha riguardato la comprensione del codice precedentemente scritto: non essendo presente una documentazione dell'applicazione, ma solo dell'implementazione del Database come diagramma E-R, è risultato abbastanza faticoso capire la funzione di certi blocchi di codice o moduli. Un altro problema ha riguardato la modifica del codice affinché fosse W3C-compatibile: ciò è stato impossibile in quanto la mole di lavoro richiesta sarebbe stato troppo elevata, ci si è limitati quindi, dopo una opportuna analisi, a modificare il codice solo nelle parti che richiedevano meno tempo.

3.2 Moodle

3.2.1 Cos'è Moodle

Moodle è un CMS (Course Management System), ovvero una piattaforma web Open Source per l'e-learning (chiamata anche LMS: Learning Management System), progettato per aiutare gli insegnanti e gli educatori a creare e gestire corsi on-line con ampie possibilità di interazione tra studente e docente, permettendo efficaci e coinvolgenti esperienze di apprendimento in rete.

Moodle è l'acronimo di Modular Object-Oriented Dynamic Learning Environment (Ambiente di Apprendimento Dinamico Modulare e Orientato ad Oggetti), anche se originariamente la M stava per "Martin", il nome del primo sviluppatore Martin Dougiamas. Le funzionalità di base di Moodle spaziano dalla creazione e all'organizzazione di corsi e lezioni on-line a strumenti per la comunità, i principali sono:

- Forum
- Gestione dei Contenuti
- Quiz
- Blog
- Wiki
- Chat
- Messaggeria Istantanea
- Glossari

Inoltre, essendo progettato in modo modulare è possibile ad esempio il supporto di più lingue, l'installazione di temi grafici diversi da quello predefinito o la creazione di nuove funzionalità non previste dalla distribuzione di base.

Infatti grazie alla sua compatibilità con lo standard SCORM 2004 rende possibile l'inserimento di Learning Object scritti in tale formato all'interno di un "corso" di Moodle, che possono essere prodotti i e resi disponibili alla vasta comunità di utilizzatori di Moodle.

3.2.2 Origini e caratteristiche di base di Moodle

Moodle è stato creato da Martin Dougiamas, un amministratore web alla Curtin University, in Australia, laureato in informatica ed educazione. Il suo Ph.D. esaminò "L'uso dell'Open source per supportare un'epistemologia sociale costruttiva dell'insegnamento e dell'apprendimento nelle comunità di internet che fanno inchieste riflessive" ("The use of Open Source software to support a social constructionist epistemology of teaching and learning within Internet-based communities of reflective inquiry").Questa ricerca ha fortemente influenzato il progetto di Moodle, procurando gli aspetti pedagogici che mancavano in molte altre piattaforme di e-learning.

La filosofia di Moodle include un approccio costruzionista e sociale all'educazione, enfatizzando il fatto che gli studenti (non solo i professori) possano contribuire all'esperienza educativa in molti modi. Le caratteristiche di Moodle riflettono questo in vari aspetti progettuali, come il rendere possibile agli studenti il commentare i contenuti in un database (o contribuire all'inserimento di dati) o lavorare in modo collaborativo in un wiki. Detto questo, Moodle è abbastanza flessibile per permettere una gamma completa di insegnamento. Può essere usato sia per la consegna introduttiva e avanzata di contenuti (ad esempio pagine HTML) o di giudizi e non è vincolato a un approccio di insegnamento costruzionista. Moodle può anche essere considerato un verbo (in inglese), che descrive il processo di improvvisazione nel “fare” ciò che è necessario, un approccio che porta spesso all'intuizione e alla creatività.

3.2.3 Statistiche

Gli utilizzatori di Moodle si stanno sempre più diffondendo in tutto il mondo. Attualmente è presente in 211 stati, e più di 49000 siti registrati e validati lo utilizzano.
Gli stati con il maggior numero di utenti registrati sono visibili nella seguente figura e riportati in tabella.

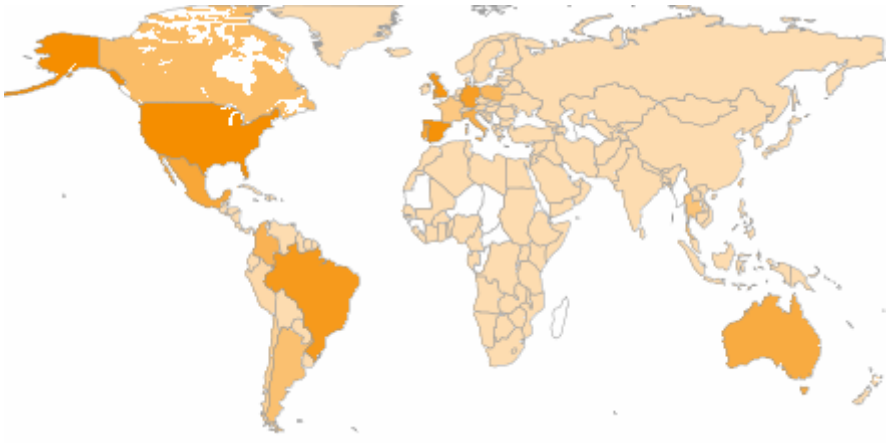

Figura 14: Diffusione di Moodle nel mondo

Paese	Registrazioni
Stati Uniti d'America	8706
Spagna	4221
Regno Unito	3044
Brasile	3004
Germania	2207
Portogallo	1786
Messico	1681
Australia	1261
Italia	**1145**
Colombia	1085

Tabella 1: Diffusione di Moodle nel mondo

3.2.3 Caratteristiche

Il sistema di e-Learning Moodle è multipiattaforma: funziona con sistemi operativi Windows, mac e la maggiorparte dei sistemi linux. Funziona senza modifiche su server Web che utilizzino sistemi Unix, Linux, FreeBSD, Windows, Mac OS X, NetWare e qualsiasi altro sistema che supporti PHP, inclusa la maggior parte dei provider di Internet.

Per l'accesso alle sue funzionalità è sufficiente un normale browser Internet. I dati sono immagazzinati in un singolo database: MySql e PostgreSql erano le uniche possibilità in Moodle 1.6. La versione 1.7, rilasciata nel novembre del 2006 usa appieno l'astrazione del database. Il software è liberamente scaricabile dal sito http://Moodle.org/ e protetto dalla licenza GPL. Attualmente è disponibile nella versione 1.9.8 rilasciata nel marzo 2010.

Moodle è strutturato in corsi: la pagina principale è il Macro-corso della Home Page da cui è possibile accedere a tutti gli altri corsi.

Ogni utente che accede a Moodle appartiene a una categoria, le principali, in ordine di permessi e autorizzazioni concesse sono:

- Guest, ovvero l'utente casuale
- Studente, ovvero uno studente che accede alle informazioni e ai corsi in Moodle
- Docente, ovvero chi si occupa della gestione di almeno un corso in Moodle.
- Amministratore, ovvero chi ha il pieno accesso a tutti i corsi e ha la possibilità di gestire l'applicazione.

A seconda delle particolari esigenze è possibile poi creare altri tipi di utenze ad hoc. Il login avviene tramite nome utente e password e l'utente rimane attivo prima dello scadere del Timeout di sistema.

3.2.4 Moodle al DEI

Moodle è stato scelto come sostituto delle “Bacheche DEI”, l'applicazione accessibile via web usata principalmente dagli studenti per registrarsi agli appelli d'esame e consultare i risultati. Senza alcun dubbio Moodle offre a studenti e docenti un numero sicuramente maggiore di funzionalità rispetto alle più obsolete “Bacheche DEI” ma come ogni cambiamento, ciò comporta inevitabilmente la necessità di apprendere e adattarsi all'utilizzo delle nuove tecnologie e l'abbandono delle vecchie abitudini di lavoro.

Moodle è stato scelto perchè si cercava una applicazione via Web che:

- Fosse gratuita, e compatibile con diversi browser web.
- Garantisse la gestione dei diversi profili utente.
- Permettesse l'inserimento dei voti da parte dei docenti e la visione degli stessi da parte degli studenti, garantendo la privacy.
- Fosse facilmente gestibile e intuitiva da utilizzare.
- Permettesse l'inserimento di news riguardanti il dipartimento e i vari corsi di laurea.

Moodle rispondendo a tutte queste caratteristiche, permettendo anche molte altre funzionalità ed essendo già in utilizzo e gestito da CINECA con buoni risultati, per il corso di laurea in Ingegneria Informatica, è stato scelto come sostituto delle “vecchie bacheche DEI”.

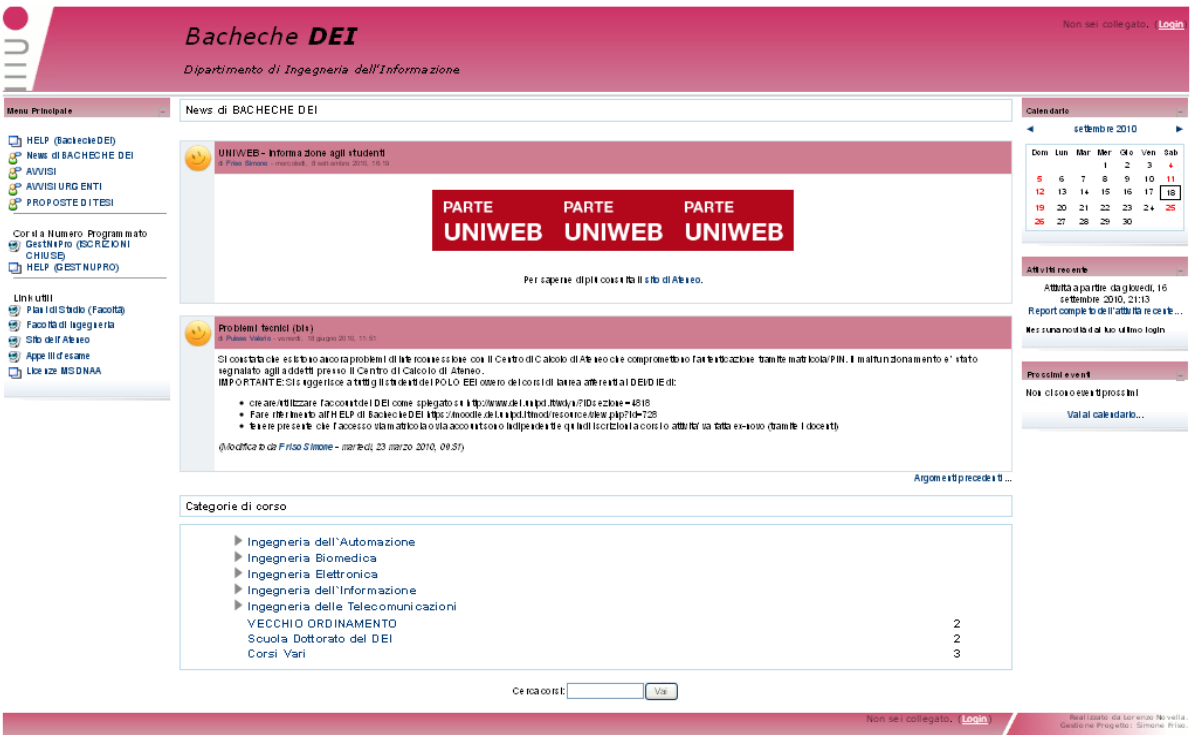

Figura 15: Moodle:l'HomePage

3.2.5 Le attività svolte

Le attività svolte hanno riguardato nell'ordine:

- I test e l'utilizzo dell'applicazione con lo scopo di acquisire padronanza e dimestichezza nella sua installazione, configurazione e utillizzo. Per fare ciò ho installato Moodle prima sul mio pc, configurando il web server Apache, il DBMS MySql e l'interprete Php.

- I test e l'utilizzo dell'applicazione nel server web della rete informatica del DEI. In questo modo si ha avuto la possibilità di testare e verificare le funzionalità dell'applicazione come nel funzionamento a regime.

- I test di utilizzo del Login tramite Matricola e Password oltre che tramite NomeUtente e password del dominio dei.unipd.it .

- L'attività di supporto e aiuto nei corsi di apprendimento dell'uso di moodle per i docenti.

- I Test di funzionamento di un modulo, creato dal precedente volontario in Servizio Civile, per l'inserimento degli appelli d'esame e l'assegnazione e visione dei voti ottenuti dagli studenti.

- La gestione dell'applicazione per i primi corsi che hanno iniziato a sfruttare Moodle prima della formale entrata in funzione del 15 settembre 2009. Questa fase è stata molto utile per capire quali erano le attese da parte dei docenti nei confronti di questa applicazione, e anche quali erano i problemi e le principali difficoltà da loro incontrati.

- La creazione della nuova tipologia di utenza Segreteria, per poter collaborare con i docenti nella gestione dei corsi.

- La redazione di una guida per i docenti. Questa guida aveva lo scopo di supportare i docenti passo passo nella gestione del loro corso in Moodle per le attività didattiche fondamentali quali :

  - ✗ Impostazioni del corso.

  - ✗ Impostazione per l'iscrizione ad esso da parte degli studenti, e il suo monitoraggio.

  - ✗ creazione di un appello d'esame.

  - ✗ inserimento dei voti ottenuti dagli studenti in un appello d'esame.

  - ✗ creazione di un forum per le comunicazioni docenti-studenti.

  - ✗ Pubblicazione di un documento

- La gestione dell'applicazione per tutti i corsi di laurea che dal 15 settembre 2009 in seguito alla chiusura delle vecchie Bacheche DEI sul sito http://sis.dei.unipd.it/ sono passati all'utilizzo a questa nuova applicazione. In questa fase la mia attività ha riguardato principalmente:

  - ✗ L'associazione delle docenze nei corrispondenti corsi.

  - ✗ L'aiuto ai docenti nella gestione dei propri corsi risolvendo eventuali problemi e cercando di rispondere alle varie esigenze.

  - ✗ La gestione delle utenze e la modifica degli account degli studenti in seguito al cambio di matricola.

3.2.6 Problemi riscontrati

I principali problemi riscontrati sono stati causati senza dubbio dal passaggio ad un nuovo tipo di tecnologia: chi è passato gradualmente a provare le funzionalità di Moodle prima del drastico spegnimento del vecchio sistema sis.dei.unipd.it ha avuto meno difficoltà di utilizzo e ha contribuito a migliorare il servizio offerto.

Le molteplici e necessità di ogni singolo docente non hanno potuto esere accontentate in tutto e per tutto: sono state fatte delle scelte e si è cercato di mediare sfruttando le funzionalità di Moodle il più possibile.

Altri problemi sono stati causati dalla dipendenza dell'accesso tramite matricola e pin da parte degli studenti che non possedevano un account al dominio dei.unipd.it, i quali erano soggetti all'autenticazione tramite il dominio unipd.it. A questi utenti l'accesso alle BachecheDEI in caso di disservizi nel server di autenticazione unipd.it non era consentito.

La grande mole di studenti iscritti in un unico corso perchè mutuato e/o gestito centralmente da più docenti ha causato qualche problema, verificatasi in poche occasioni, di anomalia nelle iscrizioni agli esami da parte di qualche studente.

3.3 Giga:Gestione Iscrizioni Gestione Accessi

3.3.1 Cos'è Giga

Giga è un'applicazione accessibile via web utilizzata dal personale afferente al DEI e dai responsabili dei vari servizi offerti dal dipartimento del DEI per gestire, monitorare, permettere o negare l'accesso a essi da parte delle persone. Il suo acronimo infatti significa Gestione Iscrizioni Gestione Accessi.

L'applicazione è stata iniziata da un volontario in Servizio Civile due anni prima ed è stata ultimata la versione 1.0 nell'anno successivo dal volontario che l'ha sostituito nel servizio, e nell'anno di servizio civile da me svolto, ha realizzato la versione 2.0. Il motore di Giga è costituito da una macchina a stati che processa e segue l'iter burocratico di ogni pratica passo-passo registrando chi e quando ha concesso le autorizzazioni ad un utente.

Figura 16: Giga: La finestra di Login

3.3.2 Le attività svolte

Le attività svolte hanno riguardato principalmente il Testing dell'applicazione con lo scopo di verificare:

1. La correttezza delle sue funzionalità.

   Per verificare la correttezza dell'applicazione si è creato un Test Plan, consistente in una serie di verifiche sia informali che formali. Per le verifiche formali si sono creati una serie di dati campione e i rispettivi risultati attesi. Questi test venivano ripetuti spesso per verificare che l'avanzare della scrittura del codice e la correzione dei bachi non producessero di nuovo gli stessi errori o non ne introducessero di nuovi.

2. Le correttezza delle interfaccie grafiche.

   La correttezza delle interfaccie grafiche si è verificata attraverso un Test Plan informale. Esso prevedeva, per ogni campo dati la verifica delle corrette proprietà di: posizione, dimensione, colore, lettura del dato, scrittura di un nuovo dato, modifica del dato, cancellazione del dato. I test venivano ripetuti su più browser web per verificare la percentuale di compatibilità di visualizzazione in quanto la piena compatibilità si è dimostrata purtroppo impossibile.

3. La robustezza dell'applicazione

   Per verificare la robustezza dell'applicazione si sono utilizzate delle verifiche informali per quanto riguarda le prestazioni, per esempio quanto tempo l'applicazione impiegava per rispondere ad una interrogazione, quanto per caricare delle pagine in vari tipi di regime simulando le contemporanee richieste di più utenti. Per prevenire l'immunità da attacchi di SQL injection si è poi provveduto a verificare il corretto riconoscimento dei dati inseriti nell'applicazione. Infine sono stati effettuati dei stress test per verificare il carico di lavoro del server e il comportamento dell'applicazione nel caso in cui dovessi trovarsi nella situazione di processare grandi moli di dati e diverse richieste contemporanee.

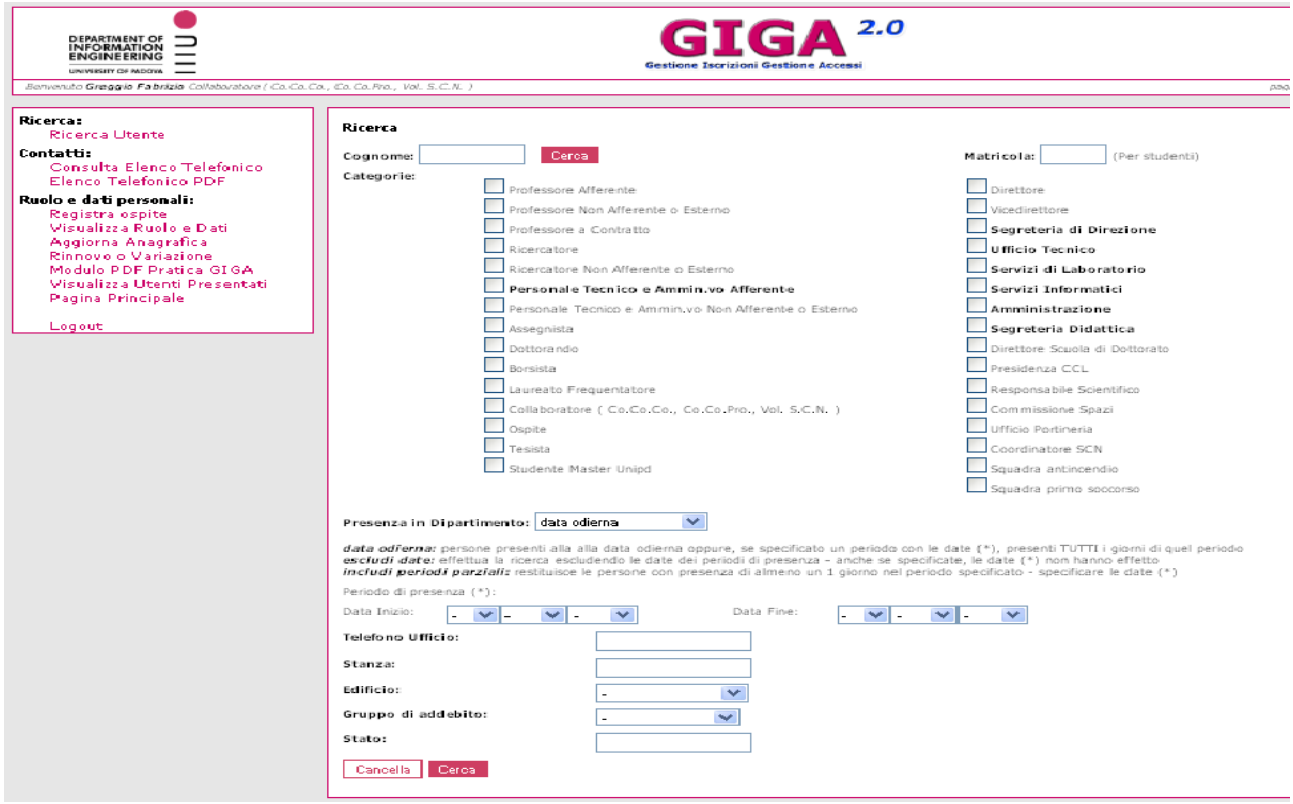

Figura 17: Le funzionalità di ricerca in Giga

3.3.3 Problemi riscontrati

I principali problemi sono stati incontrati nella selezione dei dati campione per i test in quanto non è risultato semplice riuscire a identificare tutte le possibili azioni eseguite da tutte le possibili categorie di utenti. L'analisi dei casi di test si è via via perfezionata con l'avanzare del tempo.

4. CONCLUSIONI

In conclusione si può affermare che il lavoro da me svolto è stato utile perché nonostante gli interventi apportati siano stati di modesta portata perché proporzionati alle disponibilità di tempo e manodopera, il risultato ha prodotto benefici per studenti e docenti che possono avvalersi di strumenti migliori per lo svolgimento delle attività didattiche.

5. BIBLIOGRAFIA

- A. Abran, J. W. Moore,
  Guide to the Software Engineering Body of Knowledge,
  IEEE Computer Society , California, 2004

- C. Batini, B. Pernici, G. Santucci (a cura di),
  Sistemi Informativi – Volume 5.
  Franco Angeli, 2001

- D. J. Mosley, B. A. Posey,
  Collaudo del software,
  Mc Gram Hill, 2003

- R. S. Pressman,
  Principi di ingegneria del software,
  Syxtb Edition, 2004

- I. Sommerville,
  Ingegneria del Software,
  PEARSON Addison-Wesley, 2005

6. SITOGRAFIA

- http://www.dei.unipd.it/
- http://www.moodle.dei.unipd.it/
- http://www.moodle.org/
- http://www.php.net/
- Http://www.w3c.org/