Università degli Studi di Padova

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Tesi di laurea triennale

Studio e realizzazione di applicazioni web a supporto del conseguimento della certificazione di qualità ISO 9000 da parte degli Istituti Scolastici

Relatore:
Chiar.mo Prof. Tullio Vardanega

Candidato:
Carlo Maria Massimo

Anno Accademico 2007/2008

dedica

Indice

Introduzione

Data la sempre più accentuata diversificazione in atto nel panorama degli Istituti Scolastici, resa possibile dall'autonomia di cui essi hanno goduto in questi ultimi anni, gli istituti stessi si sono trovati in posizioni di forte concorrenza gli uni con gli altri.

Questa situazione ha fatto si che l'offerta formativa potesse migliorare sensibilmente, fino a raggiungere livelli per i quali un genitore in procinto di scegliere una scuola per i propri figli colga come punti a favore non più solo i programmi proposti bensì sempre di più le eventuali certificazioni che un istituto ha saputo meritarsi.

In particolare diviene molto importante adottare un sistema di gestione per la qualità certificato. Questo sistema, per avere valenza concreta, deve essere sottoposto appunto a certificazione secondo norme stabilite dagli organismi internazionali preposti. Nasce da qui l'esigenza delle scuole di dotarsi di un'infrastruttura a sostegno del percorso di certificazione, e lo stage oggetto di questa tesi si inserisce proprio in questo punto, ove l'informatica diviene parte integrante ed importante di questa infrastruttura.

In questa sede si parlerà quindi di come sia stato condotto lo stage relativo alla realizzazione di strumenti che aiutino gli istituti scolastici nel percorso verso il conseguimento o comunque il mantenimento della certificazione del proprio sistema di gestine della qualità secondo le norme ISO 9000.

Nel primo capitolo saranno brevemente trattati i riferimenti normativi succitati per favorire la comprensione del contesto.

Nel secondo capitolo verrà trattato il contesto in cui lo stage si è inserito, verranno delineate le aspettative e i bisogni della committenza e le problematiche da risolvere.

Nel terzo capitolo si descriverà da un punto di vista prettamente tecnico lo svolgimento dello stage, portando a corredo la documentazione preparata

durante lo stage stesso.
Infine nel quarto capitolo verrà condotta un'analisi critica a posteriori del lavoro svolto e dei risultato ottenuti, sia dal punto di vista dello studente che del progetto denominato "Progetto Qualità nella Scuola".

1 Committenza

Lo stage ha visto la partecipazione nel ruolo di committenti di vari soggetti, appartenenti a settori diversi. Da un lato è stato voluto dal responsabile del Progetto Qualità nella Scuola per il Provedditorato agli Studi di Padova, professor Ugo Zambello, in collaborazione con la sede padovana di Unindustria, nella persona di Cristina Felicioni.
Questa sinergia tra mondo della scuola e mondo dell'industria si è resa necessaria si per veicolare lo stage verso l'università attraverso Unindustria, ma anche per garantire la presenza di un ente di supporto e di incentivo per gli stagisti stessi.

1.1 Aspettative e bisogni

Quello che la committenza ha dichiarato aspettarsi da questo stage è la realizzazione di un insieme di strumenti che facilitassero la gestione di alcuni aspetti necessari ad ottenere o mantenere una certificazione ISO di qualità. Poste le aspettative, sono stati individuati alcuni bisogni essenziali per le scuole, bisogni che andavano coperti prioritariamente, e che una volta definiti mediante alcuni incontri con i committenti e soggetti appartenenti a realtà scolastiche, sono stati formalizzati come obiettivi di stage. Tali obiettivi sono come già accennato solo un sottoinsieme della totalità degli aspetti che necessitano di interventi di informatizzazione in ambito della certificazione di qualità, questo per evitare di formulare richieste dispersive o addirittura fuorvianti.

1.2 Stage come parte di un progetto più ampio

Il "Progetto Qualità nella Scuola" per la provincia di Padova nasce a seguito della firma del Protocollo d'intesa firmato il 31 Maggio 2001 tra Camera di Commercio, Provincia, Unindustria Padova e Direzione Regionale MIUR

Veneto. Il Progetto costituisce una modalità per offrire alle scuole attività di informazione, consulenza e formazione finalizzate allo sviluppo di una cultura organizzativa e delle competenze necessarie per l'articolazione di un'offerta formativa rispondente ai principi e alle indicazioni dell'autonomia. Scopo di questo progetto è supportare le scuole certificate nel percorso intrapreso e seguire le altre scuole nell'avvicinamento ad una cultura della qualità attraverso il Polo Qualità di Padova.

Capitolo 1

Sistemi di Gestione per la Qualità

I sistemi di gestione per la qualità nascono dall'esigenza di un'organizzazione di essere in grado di garantire la massima soddisfazione dei propri clienti. Il cliente tipicamente richiede un prodotto (o un servizio) con delle caratteristiche derivanti dalle proprie aspettative ed esigenze spesso formalizzate in un insieme denominato "requisiti del cliente", insieme che determina in maniera fondamentale l'accettabilità di un prodotto. In questo senso i sistemi di gestione della qualità suggeriscono un approccio di analisi dei requisiti del cliente, definizione di processi di realizzazione di un prodotto accettabile e controllo degli stessi, in un'ottica di miglioramento continuo, mirato ad accrescere la fiducia si dei clienti ma anche dell'organizzazione stessa di essere in grado di fornire prodotti che rispondano sistematicamente ai requisiti. La normativa di riferimento suddivide peraltro i requisiti in "requisiti per i sistemi di gestione per la qualità" e "requisiti di prodotto", i primi specificati nella norma ISO 9001 appunto, i secondi specificati dai clienti, dall'organizzazione stessa oppure da disposizioni cogenti. La norma per l'appunto indica un approccio a più fasi per l'attuazione di un sistema di gestione di qualità, riassumibili così:

- determinare le esigenze e le aspettative dei clienti e delle altre parti interessate;
- stabilire la politica e gli obiettivi per la qualità dell'organizzazione;
- determinare i processi e le responsabilità necessari per conseguire gli obiettivi per la qualità;

- determinare e fornire le risorse necessarie per conseguire gli obiettivi per la qualità;
- stabilire metodi per misurare l'efficacia di ciascun processo;
- mettere in atto queste misure per determinare l'efficacia e l'efficienza di ciascun processo;
- determinare i mezzi per prevenire le non conformità ed eliminarne le cause;
- stabilire ed applicare un processo per il miglioramento continuo del sistema di gestione per la qualità.

L'approccio appena descritto può contribuire ad aumentare la soddisfazione dei clienti ed il successo dell'organizzazione che lo adotta.

1 Regolamenti

Come già accennato, esistono delle norme che aiutano le organizzazioni ad attuare ed applicare sistemi di gestione per la qualità efficaci, queste norme vanno sotto la denominazione ISO 9000.

- ISO 9000 che descrive i fondamenti dei sistemi di gestione della qualità e ne specifica la terminologia.
- ISO 9001 che specifica i requisiti di un sistema di gestione della qualità da utilizzarsi quando un'organizzazione debba dimostrare la propria capacità a fornire prodotti che soddisfino i requisiti dei clienti e quelli cogenti applicabili e miri a conseguire la soddisfazione dei clienti.
- ISO 9004 che fornisce linee guida che tengono conto sia dell'efficienza sia dell'efficacia dei sistemi di gestione per la qualità. Lo scopo della presente norma è il miglioramento continuo delle prestazioni dell'organizzazione e la soddisfazione dei clienti e delle altre parti interessate.
- ISO 19011 che fornisce una guida sulle verifiche ispettive di sistemi di gestione per la qualità ed ambientali.
  In questa sezione si tratterà brevemente delle norme 9000 e 9001 in quanto fondamentali per comprendere il contesto in cui lo stage era inserito.

1.1 La norma ISO 9000

La base per le norme sui sistemi di gestione della qualità può essere individuata in otto principi applicabili in un'ottica di miglioramento delle prestazioni di un'organizzazione, enunciati nella norma in oggetto.

Orientamento al cliente Le organizzazioni dipendono dai loro clienti e pertanto dovrebbero capire le loro esigenze presenti e future, soddisfare i loro requisiti e mirare a superare le loro stesse aspettative

Leadership I capi stabiliscono unità di intenti e di indirizzo dell'organizzazione. Essi dovrebbero creare e mantenere un ambiente interno che coinvolga pienamente il personale nel perseguimento degli obiettivi dell'organizzazione.

Coinvolgimento del personale Le persone, a tutti i livelli, costituiscono l'essenza dell'organizzazione ed il loro pieno coinvolgimento permette di porre le loro capacità a servizio dell'organizzazione.

Approccio per processi Un risultato desiderato si ottiene con maggiore efficienza quando le relative attività e risorse sono gestite come un processo.

Approccio sistemico alla gestione Identificare, capire e gestire (come fossero un sistema) processi tra loro correlati contribuisce all'efficacia ed all'efficenza dell'organizzazione nel conseguire i propri obiettivi.

Miglioramento continuo Il miglioramento continuo delle prestazioni complessive dovrebbe essere un obiettivo permanente dell'organizzazione.

Decisioni basate su dati di fatto Le decisioni efficaci si basano sull'analisi di dati e di informazioni.

Rapporti di reciproco beneficio con i fornitori Un'organizzazione e i suoi fornitori sono interdipendenti ed un rapporto di reciproco beneficio migliora, per entrambi, la capacità di creare valore.

La norma inoltre tratta una serie di aspetti fondamentali per l'attuazione di un sistema di qualità quali:

politica ed obiettivi per la qualità , vengono stabiliti per fornire indirizzi per guidare l'organizzazione. Entrambi individuano i risultati da raggiungere ed assistono l'organizzazione nell'utilizzazione delle risorse per raggiungerli.

2 Implicazioni

3 Ciclo PDCA

4 Automazione della Gestione dei Processi di Qualità

Capitolo 2

Problema Specifico

1 Implicazioni derivabili dal contesto

2 Obiettivi

Obiettivo finale

Requisiti impliciti

Capitolo 3

Resoconto dello Stage

In questo capitolo è descritto il profilo del progetto di stage, il suo svolgimento e le tecnologie utilizzate. Nel descrivere lo svolgimento si farà uso principalmente di concetti in uso nell'ingegneria del software.
Data la grande autonomia data dal contesto di stage, si è scelto di far procedere le attività secondo un ciclo di vita definito dallo standard ISO 12207 ovvero il ciclo di vita evolutivo, per questo composto di fasi ben delineate.

1 Complessità

Lo stage in oggetto si proponeva di fornire agli Istituti coinvolti nel Progetto Qualità per la Scuola degli strumenti che facilitassero il percorso verso una certificazione di qualità appunto.
Il progetto si proponeva di tradurre generici bisogni degli istituti scolastici in merito alla gestione della qualità in requisiti formali ed inoltre di realizzare un'applicazione web a sostituzione di alcune infrastrutture esistenti, per migliorare e rendere più omogeneo l'insieme di funzionalità. Gran parte della complessità di questo stage si è concentrata nel dialogo con i committenti, perlopiù provenienti da percorsi di formazione non informatica. Date queste premesse si è dovuto porre grande attenzione nell'individuazione dei bisogni, in modo da evitare di tralasciare aspetti importanti ovvero di includere nell'analisi aspetti superficiali se non addirittura inutili. Questo ha quindi portato naturalmente a procedere in maniera iterativa:

- organizzando incontri con la committenza nei quali discutere idee, bisogni ed aspettative
- formalizzando queste discussioni in documentazione
- sottoponendo il tutto nuovamente al vaglio della committenza

sino all'approvazione e alla conseguente cristallizzazione di quanto redatto nella documentazione formale.

Superata questa fase, la maggior fonte di complessità è stato il sistema di automazione del deploy e della gestione del prodotto in uso. Ci si è scontrati con una grande eterogeneità di ambienti (anche all'interno dello stesso istituto) dove le problematiche prettamente informatiche venivano spesso aggravate dalla mancanza di adeguato supporto del personale scolastico.
Questa fase dello stage ha inoltre richiesto ulteriore impegno non preventivato e quindi non incluso nella pianificazione originaria.

2 Analisi dei Requisiti Preliminare

In questa fase, svoltasi qualche tempo prima dell'inizio dell'attività vera e propria, sono stati individuati dei requisiti generali, e un sottoinsieme dei requisiti specifici, non essendo ancora chiaro il numero di partecipanti e quindi quantificabile l'impegno.

Studio delle Funzionlità

Lo studio delle funzionalità è stato condotto mediante la redazione di diagrammi Use Case. Nello sviluppo del progetto, la componente analitica degli Use Case si è rivelata fondamentale in quanto andando essi ad individuare le possibilità d'azione, ed essendo queste in buona misura note alla commitenza, ci si è focalizzati su una copertura quanto maggiore possibile degli scenari derivabili dai vari colloqui tenuti con la committenza stessa.

Esempio di Use Case

Segue un esempio di Use Case tratto dal documento "Analisi dei Requisiti" redatto per il modulo di 'Gestione Iter Segnalazioni di Non Conformità' riguardo l'assegnamento delle segnalazioni ad un agente risolutore.

Diagramma il diagramma UML:

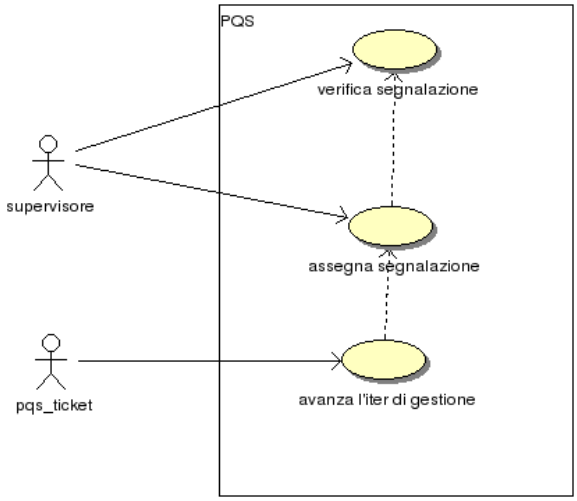

Figura 3.1: Assegnamento di una segnalazione da parte di un supervisore

Attori Coinvolti Gli attori coinvolti sono:

- un utente con ruolo supervisore
- l'applicazione stessa

Flusso degli Eventi L'utente supervisore una volta verificata la segnalazione (processo esterno al sistema) la assegna ad un utente preposto al trattamento, di fatto convalidando la segnalazione stessa (plan). Il sistema fa quindi avanzare di un passo la gestione vincolata, idealmente fino allo stato di "assegnata".

Precondizioni L'utente deve disporre dei privilegi necessari a compiere le azioni prefissate.

2.1 Individuazione dei Requisiti

I prodotti dell'analisi condotta mediante Use Case sono stati quindi utilizzati per l'elaborazione dell'insieme di requisiti, suddivisi nei seguenti insiemi:

- Requisiti funzionali
- Requisiti di qualità
- Requisiti di interfacciamento

ad ogni requisito individuato è stato assegnato un codice univoco per garantirne il corretto tracciamento attraverso le fasi di sviluppo.

Requisiti funzionali

I requisiti funzionali sono stati sintetizzati nella seguente tabella.
Il prodotto avrebbe dovuto permettere:

codice	descrizione
RF-01	l'archiviazione delle segnalazioni suddivise per macro aree tematiche definite dall'utilizzatore
RF-02	l'archiviazione di segnalazioni accompagnate da allegati
RF-03	l'inserimento di una segnalazione
RF-04	di specificare il trattamento di una segnalazione
RF-05	la verifica dello stato di una segnalazione
RF-06	la chiusura dell'iter di una segnalazione risolta
RF-07	di effettuare ricerche parametriche sull'archivio delle segnalazioni
RF-08	l'esportazione di report dall'archivio delle segnalazioni
RF-09	la gestione degli utenti dell'applicazione (creazione, modifica, cancellazione)
RF-10	operazioni di backup dell'archivio delle segnalazioni ovvero dello stato corrente dell'applicazione

Requisiti di qualità

I requisiti di qualità sono stati sintetizzati nella seguente tabella.

codice	descrizione
RQ-01	Il sistema deve garantire la soddisfazione dei presenti requisiti
RQ-02	Il prodotto sarà sottoposto a processi di verifica continua per garantire l'idoneità alla messa in produzione

Requisiti di interfacciamento

I requisiti di interfacciamento si suddividono a loro volta nelle seguenti aree:

Ambiente di installazione ed uso

Requisiti riguardanti l'insieme di tecnologie con cui il prodotto dovrà interagire per quanto concerne installazione ed uso.

codice	descrizione
RI-A-01	Il sistema deve poter essere reso operativo sul più gran numero di piattaforme, per far fronte all'eterogeneità delle infrastrutture scolastiche
RI-A-02	Il sistema deve apparire graficamente omogeneo al portale già in uso alla scuola

Processi produttivi e modalità d'uso presso l'utente

Requisiti riguardanti l'integrazione del prodotto con il processi già in funzione presso l'utente.

codice	descrizione
RI-P-01	L'avanzamento dell'iter delle segnalazioni dovrà essere vincolato dal sistema, ovvero non dovrà essere possibile modificare o forzare l'iter al di fuori delle fasi previste dalla norma ISO 9001

Operatore

Requisiti riguardanti l'interfacciamento con un operatore generico (in questo caso umano).

codice	descrizione
RI-O-01	Il sistema dovrà presentare un'interfaccia grafica accessibile ed usabile, conforma alla normativa Stanca.

Prodotti e FeedBack

I prodotti della fase di Analisi dei Requisiti preliminare identificabili nelle tabelle precedenti sono stati quindi sottoposti al vaglio della committenza per approvazione ed eventuale correzione prima di procedere alla fase successiva.

3 Piano di Lavoro

La pianificazione delle attività di stage proposta nel piano di lavoro approvato per lo stage consisteva di una durata di 7 settimane per un totale di 304 ore suddivise nelle fasi seguenti.

3.1 Aggiornamento Analisi dei Requisiti

Questa fase, della durata di 49 ore, consisteva nella rielaborazione del documento "Analisi dei requisiti" presentato nella sezione precedente, ovvero eventuale aggiornamento dei requisiti elaborati ed integrazione dei nuovi documenti di analisi redatti a fronte dell'individuazione di nuovi moduli da realizzare dovuti alla partecipazione di un singolo studente all'attività di stage.

3.2 Studio dell'ambiente di sviluppo

Questa fase si è resa necessaria per permettere allo studente di ottenere una buona padronanza della tecnologia scelta per realizzare i prodotti di stage.
Nella durata di 15 ore ci si prefiggeva lo studio del framework di sviluppo denominato "Ruby On Rails" mediante studio approfondito di documentazione specifica e realizzazione di un prototipo la cui complessità consentisse di esplorare le principali funzionalità del framework stesso. Date le tempistiche con cui è stata elaborata la pianificazione delle attività, questa fase si è trovata ad essere completamente sovrapposta alla precedente (cfr. diagramma di Gantt) ed il ridotto numero di ore assegnatele è dovuto alla minor priorità rispetto alla fase di analisi.

3.3 Progettazione

Di grande importanza, questa fase ha visto un'assegnazione di ben 88 ore lavorative.
In questa fase è stata studiata, elaborata e prodotta la progettazione architetturale e di dettaglio dei moduli software da svilupparsi nelle fasi successive. Si è quindi proceduto all'individuazione dell'architettura del sistema e delle sue componenti, seppur in ciò coadiuvati da vincoli imposti dalla tecnologia scelta. Questi vincoli tutt'altro che limitanti hanno permesso una maggior concentrazione sulla parte di dettaglio, con evidente guadagno in termini di tempo e qualità, essendo la soluzione proposta dal framework già molto definita e collaudata.

3.4 Implementazione

Questa fase, della durata di 44 ore lavorative, ha visto la completa implementazione del sistema a seguito dei prodotti della fase precedente. In questa fase è stato prodotto il codice applicativo, la documentazione, le procedure di installazione e gran parte del codice di test da utilizzarsi nella successiva fase di qualifica.

3.5 Qualifica e Collaudo

Questa fase consta in realtà di due sottofasi della durata rispettivamente di 52 e 56 ore.
Nella prima sottofase è stata condotta la qualifica sui prodotti della fase di implementazione utilizzando a tale scopo le facilities di test messe a disposizione dal framework di sviluppo. Si è quindi proceduto alla redazione della manualistica richiesta (manuale utente e manuale amministratore). Infine si è testata la procedura di automazione del deploy e al collaudo del prodotto alla presenza dei committenti.

3.6 Milestone

La pianificazione ha previsto le seguenti milestone:

Milestone 1 - Requisiti

deadline: 25 Ottobre 2007
obiettivi:

- Analisi dei Requisiti per verifica del committente
- Documento relativo alla progettazione architetturale
- Prototipo dell'applicazione derivato da un sottoinsieme dei requisiti

Per questa milestone è stata prevista una release esterna del prototipo ed una presentazione ai committenti degli obiettivi sino a questo punto raggiunti.

Milestone 2 - Prototipo

deadline: 10 Novembre 2007
obiettivi:

- La documentazione riguardante le fasi precedenti
- La logica dell'applicazione in oggetto

Anche per questa milestone è stata prevista una release esterna del prodotto in modo da raccogliere feedback per raffinamenti in corso d'opera.

Milestone 3 - Applicazione Completa

deadline: 23 Novembre 2007
obiettivi:

- L'applicazione completa e testata
- Analisi consuntiva delle ore lavorate

È stata prevista una release esterna del prodotto per dimostrare ai committenti il sostanziale soddisfacimento dei requisiti e la completezza delle funzionalità.

Milestone 4 - Collaudo

deadline: 30 Novembre 2007
obiettivi:

- La procedura di automazione del deploy dell'intera applicazione su un'infrastruttura generica

- Manuale Utente
- Manuale Amministratore

Questa milestone prevede il collaudo dell'intero sistema alla presenza dei committenti.

4 Diagramma di Gantt

Segue il diagramma di Gantt elaborato per la pianificazione.

stage PQS

Start: October 10, 2007
Finish: November 30, 2007
Report Date: October 29, 2007

Gantt Chart

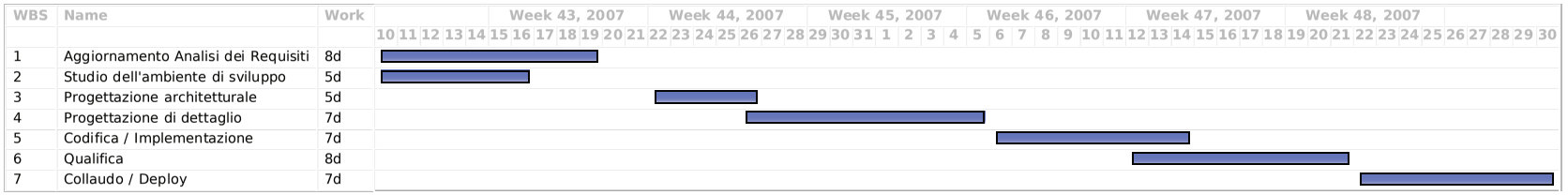

Figura 3.2: La suddivisione delle attività nelle 7 settimane di stage

5 Analisi dei Requisiti

Lo stage pensato inizialmente per coinvolgere 4 o 5 studenti ha visto poi all'inizio dei lavori la presenza di un solo candidato.
Data la scarsità di risorse umane, è stato necessario un ridimensionamento dell'insieme degli obiettivi a breve termine, mentre si è cercato di mantenere un'impostazione che sostenesse una certa continuità a lungo termine. Perciò si è posto l'accento sulla fase di analisi nella quale sono stati condotti degli studi su un numero di moduli maggiori rispetto a quanto poi sarebbe stato effettivamente realizzato, questo per fornire eventuale base di partenza per futuri partecipanti a stage analoghi. I moduli analizzati sono:

- Modulo software per la gestione delle segnalazioni e le non conformià (modulo gestione non conformità)
- Modulo software per la gestione della somministrazione e raccolta dati da questionari di autovalutazione degli istituti scolastici (modulo gestione questionari)
- Modulo software per la gestione dei rapporti con i fornitori (modulo gestione fornitori)

Per ognuno di questi tre moduli è stato prodotto un documento di analisi dei requisiti descritto di seguito.

5.1 Modulo Gestione delle Non Conformità

Studio delle Funzionlità

Lo studio delle funzionalità per il modulo oggetto di questo documento di analisi è stato condotto mediante Use Case. Segue quindi un esempio di diagramma use case realizzato nell'ambito dello studio.

Esempio di Use Case

Esempio tratto dal documento 'Analisi dei Requisiti'.

Diagramma il diagramma UML:

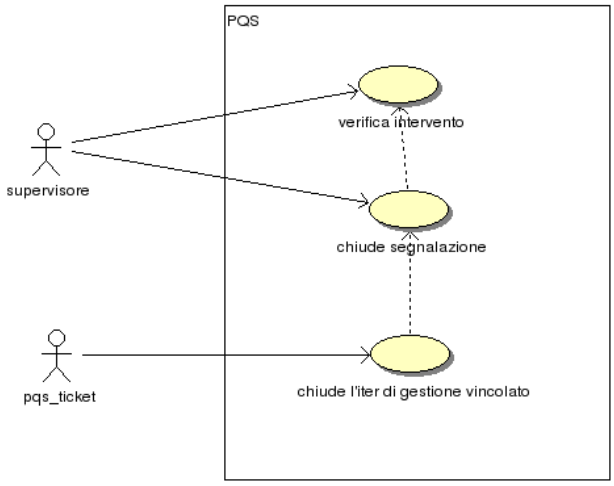

Figura 3.3: Chiusura di una segnalazione e fine dell'iter di gestione.

Attori Coinvolti Gli attori coinvolti sono:

- un utente con ruolo supervisore
- l'applicazione stessa

Flusso degli Eventi L'utente supervisore, una volta verificato per suo conto l'avvenuto trattamento della segnalazione (check), può marcarla come risolta e quindi permettere al sistema di concludere l'iter vincolato di gestione e memorizzare l'avvenuta soluzione.

Flusso Alternativo degli Eventi Qualora l'utente supervisore non ritenga soddisfatta la segnalazione (check), potrà decidere di riassegnarla ad un utente risolutore perchè agisca di conseguenza (act).

Precondizioni L'utente deve disporre dei privilegi necessari a compiere le azioni prefissate.

5.2 Individuazione dei Requisiti

I prodotti dell'analisi condotta mediante Use Case sono stati quindi utilizzati per l'elaborazione dell'insieme di requisiti, suddivisi nei seguenti insiemi:

- Requisiti funzionali

- Requisiti di qualità
- Requisiti di interfacciamento

ad ogni requisito individuato è stato assegnato un codice univoco per garantirne il corretto tracciamento attraverso le fasi di sviluppo.

Requisiti funzionali

I requisiti funzionali sono stati sintetizzati nella seguente tabella.
Il prodotto avrebbe dovuto permettere:

codice	descrizione
RF-01	l'archiviazione delle segnalazioni suddivise per macro aree tematiche definite dall'utilizzatore
RF-02	l'archiviazione di segnalazioni accompagnate da allegati
RF-03	l'inserimento di una segnalazione
RF-04	di specificare il trattamento di una segnalazione
RF-05	la verifica dello stato di una segnalazione
RF-06	la chiusura dell'iter di una segnalazione risolta
RF-07	di effettuare ricerche parametriche sull'archivio delle segnalazioni
RF-08	l'esportazione di report dall'archivio delle segnalazioni
RF-09	la gestione degli utenti dell'applicazione (creazione, modifica, cancellazione)
RF-10	operazioni di backup dell'archivio delle segnalazioni ovvero dello stato corrente dell'applicazione

Requisiti di qualità

I requisiti di qualità sono stati sintetizzati nella seguente tabella.

codice	descrizione
RQ-01	Il sistema deve garantire la soddisfazione dei presenti requisiti
RQ-02	Il prodotto sarà sottoposto a processi di verifica continua per garantire l'idoneità alla messa in produzione

Requisiti di interfacciamento

I requisiti di interfacciamento si suddividono a loro volta nelle seguenti aree:

Ambiente di installazione ed uso

Requisiti riguardanti l'insieme di tecnologie con cui il prodotto dovrà interagire per quanto concerne installazione ed uso.

codice	descrizione
RI-A-01	Il sistema deve poter essere reso operativo sul più gran numero di piattaforme, per far fronte all'eterogeneità delle infrastrutture scolastiche
RI-A-02	Il sistema deve apparire graficamente omogeneo al portale già in uso alla scuola

Processi produttivi e modalità d'uso presso l'utente

Requisiti riguardanti l'integrazione del prodotto con il processi già in funzione presso l'utente.

codice	descrizione
RI-P-01	L'avanzamento dell'iter delle segnalazioni dovrà essere vincolato dal sistema, ovvero non dovrà essere possibile modificare o forzare l'iter al di fuori delle fasi previste dalla norma ISO 9001

Operatore

Requisiti riguardanti l'interfacciamento con un operatore generico (in questo caso umano).

codice	descrizione
RI-O-01	Il sistema dovrà presentare un'interfaccia grafica accessibile ed usabile, conforma alla normativa Stanca.

Prodotti e FeedBack

I prodotti della fase di Analisi dei Requisiti preliminare identificabili nelle tabelle precedenti sono stati quindi sottoposti al vaglio della committenza per approvazione ed eventuale correzione prima di procedere alla fase successiva.

5.3 Modulo Questionari

Studio delle Funzionlità

Lo studio delle funzionalità per il modulo oggetto di questo documento di analisi è stato condotto mediante Use Case. Segue quindi un esempio di diagramma use case realizzato nell'ambito dello studio.

Esempio di Use Case

Segue un esempio di Use Case tratto dal documento "Analisi dei Requisiti" redatto per il modulo di "Gestione Questionari".

Diagramma il diagramma UML:

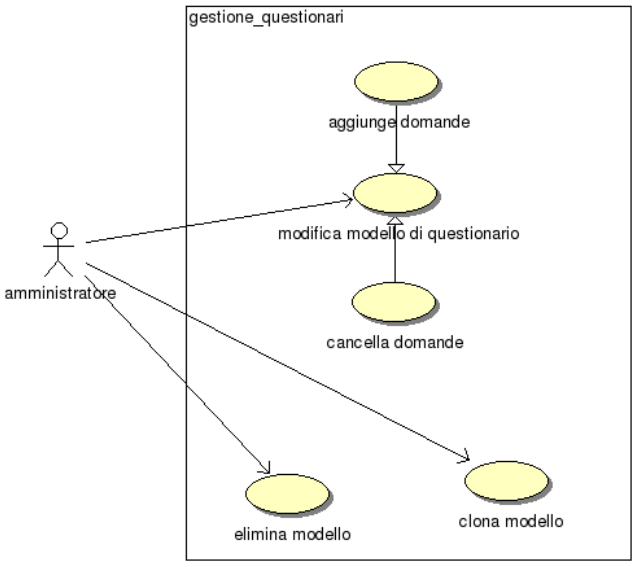

Figura 3.4: Modifica di un modello di questionario creato in precedenza

Attori Coinvolti Gli attori coinvolti sono:

- l'amministratore dell'applicazione

Flusso degli Eventi L'amministratore, dato un modello di questionario precedentemente archiviato, può aggiungere ulteriori quesiti, eliminarne, oppure decidere di lavorare su una copia perfetta (un clone) del modello in

oggetto. È prevista la possibilità di eliminare un modello e quindi tutte le eventuali domande ad esso associate, non dovessero essere utilizzate da altri modelli.

Precondizioni L'utente deve disporre dei privilegi necessari a compiere le azioni prefissate.

5.4 Individuazione dei Requisiti

I prodotti dell'analisi condotta mediante Use Case sono stati quindi utilizzati per l'elaborazione dell'insieme di requisiti, suddivisi nei seguenti insiemi:

- Requisiti funzionali
- Requisiti di qualità
- Requisiti di interfacciamento

ad ogni requisito individuato è stato assegnato un codice univoco per garantirne il corretto tracciamento attraverso le fasi di sviluppo.

Requisiti funzionali

I requisiti funzionali sono stati sintetizzati nella seguente tabella.
Il prodotto avrebbe dovuto permettere:

codice	descrizione
RF-01	creare/modificare/cancellare un nuovo modello di questionario
RF-02	creare/modificare/cancellare quesiti di varia tipologia da assegnare ad un modello di questionario
RF-03	abilitare o meno la compilazione di un questionario
RF-04	compilare il questionario in forma anonima
RF-05	archiviare i questionari compilati
RF-06	esportare i dati in formato compatibile con fogli di calcolo
RF-07	permettere di eseguire query direttamente alla base dati sottostante

Requisiti di qualità

I requisiti di qualità sono stati sintetizzati nella seguente tabella.

codice	descrizione
RQ-01	Il sistema deve garantire la soddisfazione dei presenti requisiti
RQ-02	Il prodotto sarà sottoposto a processi di verifica continua per garantire l'idoneità alla messa in produzione

Requisiti di interfacciamento

I requisiti di interfacciamento si suddividono a loro volta nelle seguenti aree:

Ambiente di installazione ed uso

Requisiti riguardanti l'insieme di tecnologie con cui il prodotto dovrà interagire per quanto concerne installazione ed uso.

codice	descrizione
RI-A-01	Il sistema deve poter essere reso operativo sul più gran numero di piattaforme, per far fronte all'eterogeneità delle infrastrutture scolastiche
RI-A-02	Il sistema deve apparire graficamente omogeneo al portale già in uso alla scuola

Processi produttivi e modalità d'uso presso l'utente

Requisiti riguardanti l'integrazione del prodotto con il processi già in funzione presso l'utente.

codice	descrizione
RI-P-01	L'avanzamento dell'iter delle segnalazioni dovrà essere vincolato dal sistema, ovvero non dovrà essere possibile modificare o forzare l'iter al di fuori delle fasi previste dalla norma ISO 9001

Operatore

Requisiti riguardanti l'interfacciamento con un operatore generico (in questo caso umano).

codice	descrizione
RI-O-01	Il sistema dovrà presentare un'interfaccia grafica accessibile ed usabile, conforma alla normativa Stanca.

Prodotti e FeedBack

I prodotti della fase di Analisi dei Requisiti preliminare identificabili nelle tabelle precedenti sono stati quindi sottoposti al vaglio della committenza per approvazione ed eventuale correzione prima di procedere alla fase successiva.

5.5 Modulo Gestione Fornitori

Studio delle Funzionlità

Lo studio delle funzionalità per il modulo oggetto di questo documento di analisi è stato condotto mediante Use Case. Segue quindi un esempio di diagramma use case realizzato nell'ambito dello studio.

Esempio di Use Case

Segue un esempio di Use Case tratto dal documento "Analisi dei Requisiti" redatto per il modulo di "Gestione Fornitori".

Diagramma il diagramma UML:

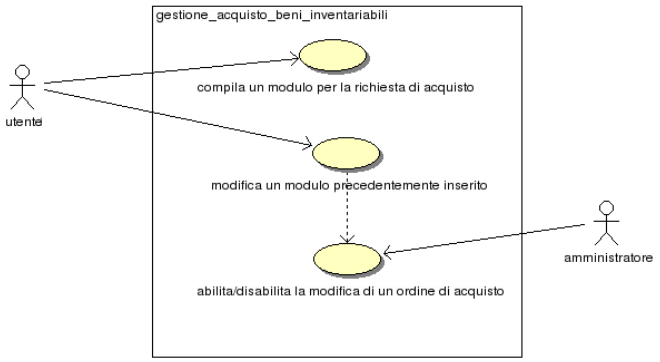

Figura 3.5: Inserimento e modifica di un modulo di acquisto di beni o servizi

Attori Coinvolti Gli attori coinvolti sono:

- l'amministratore dell'applicazione
- un utente generico

Flusso degli Eventi Un generico utente compila ed inserisce il modulo di richiesta di acquisto dei beni o servizi. Ha successivamente la possibilità di modificare l'ordine inserito salvo veto da parte dell'amministratore che può abilitare o disabilitare l'operazione per una intero gruppo di utenti.

Precondizioni L'amministratore deve essere correttamente autenticato al sistema come tale.

5.6 Individuazione dei Requisiti

I prodotti dell'analisi condotta mediante Use Case sono stati quindi utilizzati per l'elaborazione dell'insieme di requisiti, suddivisi nei seguenti insiemi:

- Requisiti funzionali
- Requisiti di qualità
- Requisiti di interfacciamento

ad ogni requisito individuato è stato assegnato un codice univoco per garantirne il corretto tracciamento attraverso le fasi di sviluppo.

Requisiti funzionali

I requisiti funzionali sono stati sintetizzati nella seguente tabella.
Il prodotto avrebbe dovuto permettere:

codice	descrizione
RF-01	la compilazione di un modulo d'acquisto
RF-02	la modifica di un modulo di acquisto già inoltrato
RF-03	il veto sulle operazioni al requisito RF-02 da parte dell'amministratore
RF-04	la registrazione di una scheda dettagliata per ogni fornitore
RF-05	la successiva modifica o cancellazione della scheda al requisito RF-04
RF-06	la selezione di criteri di valutazione per un dato fornitore
RF-07	l'assegnazione e la modifica ad un fornitore di un punteggio
RF-08	effettuare report sui dati presenti nel sistema
RF-09	compilare una scheda dettagliata riguardo il collaudo
RF-10	modificare o cancellare tale scheda
RF-11	effettuare report sui dati, sia nella loro interezza che mediante parametri

Requisiti di qualità

I requisiti di qualità sono stati sintetizzati nella seguente tabella.

codice	descrizione
RQ-01	Il sistema deve garantire la soddisfazione dei presenti requisiti
RQ-02	Il prodotto sarà sottoposto a processi di verifica continua per garantire l'idoneità alla messa in produzione

Requisiti di interfacciamento

I requisiti di interfacciamento si suddividono a loro volta nelle seguenti aree:

Ambiente di installazione ed uso

Requisiti riguardanti l'insieme di tecnologie con cui il prodotto dovrà interagire per quanto concerne installazione ed uso.

codice	descrizione
RI-A-01	Il sistema deve poter essere reso operativo sul più gran numero di piattaforme, per far fronte all'eterogeneità delle infrastrutture scolastiche
RI-A-02	Il sistema deve apparire graficamente omogeneo al portale già in uso alla scuola

Processi produttivi e modalità d'uso presso l'utente

Requisiti riguardanti l'integrazione del prodotto con il processi già in funzione presso l'utente.

codice	descrizione
RI-P-01	L'avanzamento dell'iter delle segnalazioni dovrà essere vincolato dal sistema, ovvero non dovrà essere possibile modificare o forzare l'iter al di fuori delle fasi previste dalla norma ISO 9001

Operatore

Requisiti riguardanti l'interfacciamento con un operatore generico (in questo caso umano).

codice	descrizione
RI-O-01	Il sistema dovrà presentare un'interfaccia grafica accessibile ed usabile, conforma alla normativa Stanca.

Prodotti e FeedBack

I prodotti della fase di Analisi dei Requisiti preliminare identificabili nelle tabelle precedenti sono stati quindi sottoposti al vaglio della committenza per approvazione ed eventuale correzione prima di procedere alla fase successiva.

6 Tecnologia Adottata

Ruby on Rails, o più semplicemente Rails, è un ambiente completo per lo sviluppo web, che contiene al suo interno tutti gli elementi necessari alla realizzazione di siti complessi permettendo di gestire facilmente la creazione di pagine (X)HTML, di accedere semplicemente a vari database, e di integrare le funzionalità che caratterizzano le applicazioni web moderne, come AJAX ed i Web Service. Rails è un framework di nuova generazione che negli ultimi due anni ha creato un vero e proprio terremoto nella comunità degli sviluppatori, diventando spesso motivo di dibattito e ispirando la nascita di progetti analoghi realizzati con tecnologie differenti, come Cake per PHP, Trails per Java, Turbogears e Subway per Python e molti altri. Rails, in altre parole, ha introdotto un fattore di novità rilevante nell'ambito della programmazione Web. Andando a guardare Rails nel dettaglio si scopre che esso usa tecniche di programmazione già sperimentate e non rivoluzionarie. L'autore originale tiene a ribadire che Rails non è stato sviluppato da subito come una piattaforma indipendente, ma che è il risultato dell'estrazione di funzionalità già provate in un'applicazione funzionante, e che ogni feature è mirata alla soluzione di problemi reali e non è frutto di ragionamenti astratti. L'opinione condivisa è che sia proprio questo a renderlo così efficace. Indubbiamente parte del successo di Rails è dovuto al linguaggio con cui è scritto, ovvero Ruby, un linguaggio completamente ad oggetti di estrema espressività e potenza, che riesce a fondere in una sintassi semplice e chiara funzionalità ereditate da Perl, Python, Lisp e Smalltalk.

6.1 Analisi

Il Linguaggio

Ruby è un linguaggio di programmazione orientato agli oggetti e general purpose, che combina sintassi ispirata da Perl a funzionalità di Smalltalk. Ruby è nato in Giappone a metà degli anni 90 ed è stato inizialmente sviluppato da Yukihiro Matsumoto. Ruby supporta vari paradigmi di programmazione, funzionale, orientato agli oggetti, imperativo e reflection. È inoltre dinamicamente tipato e possiede una gestione automatica della memoria, è quindi sotto vari aspetti molto simile a Python, Perl e Lisp. L'implementazione standard (1.8.6 stable) è interamente scritta in C, come linguaggio interpretato in singola passata. Non esiste al momento una specifica del linguaggio quindi l'implementazione originale viene considerata reference de facto. Inoltre si assiste alla nascita di un certo numero di differenti implementazioni alternative, complete o in fase di completamento come YARV, JRuby, Rubinius, IronRuby e MacRuby, ognuna delle quali affronta l'implementazione secondo un approccio differente, addirittura offrendo funzionalità di compilazione just-in-time come JRuby. La branca di sviluppo ufficiale (1.9) usa YARV (Yet Another Ruby VM) e così farà la branca 2.0 ove il nuovo interprete sostituirà l'originale e molto più lento Ruby MRI.

Ruby On Rails

Peculiarità

6.2 Pro e Contro

7 Progettazione Architetturale e di Dettaglio

7.1 Vincoli Tecnologici

Data la tecnologia utilizzata, la progettazione architetturale ha assorbito quelle che sono le scelte del framework, ovvero una struttura secondo il design pattern MVC.

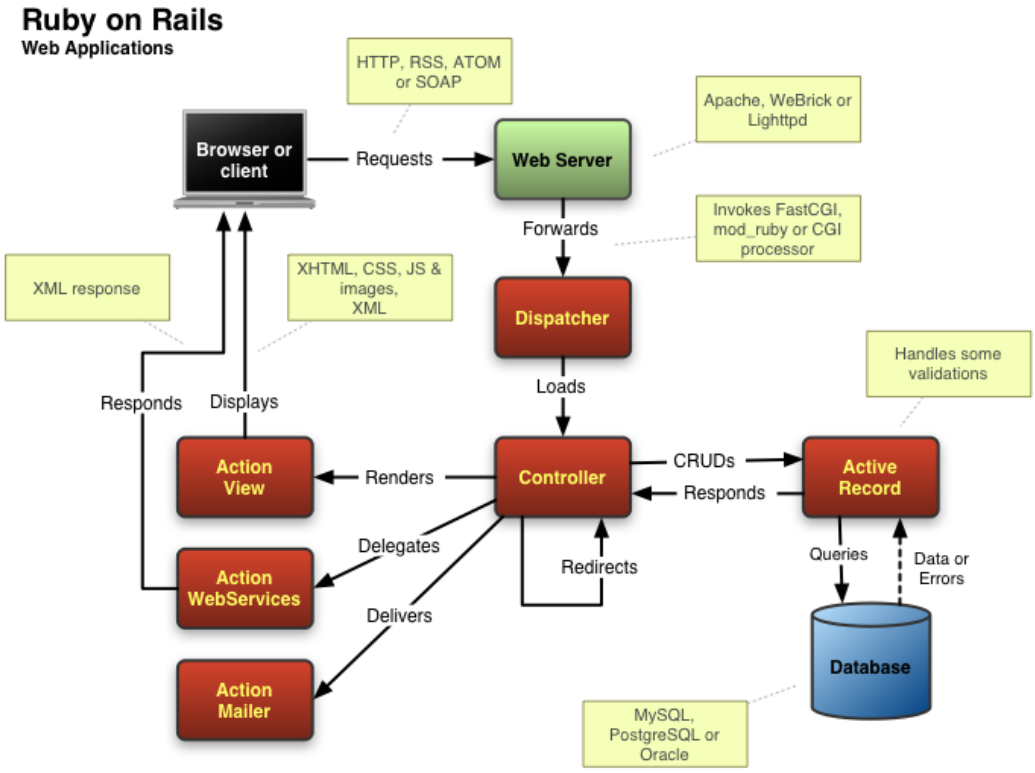

Figura 3.6: L'architettura MVC del framework Ruby On Rails

Il modulo "ActiveRecord" in figura 3.6, contiene tutte le componenti dello strato Model, il modulo "ActionController", come suggerisce il nome tutte le componenti dello strato Controller ed infine abbiamo lo strato View che è rappresentato dal modulo "ActionView". Questi tre moduli sono alla base del funzionamento del framework e ne costituiscono le componenti principali.

ActiveRecord

ActiveRecord è il livello ORM di Rails. Si occupa delle connessioni alla base dati, della mappatura delle tabelle e della manipolazione e persistenza dei dati. Gestisce inoltre le relazioni tra le tabelle, il ciclo di vita degli oggetti aspetti quali la validazione e il filtraggio. ActiveRecord segue molto da vicino il model ORM standard: le tabelle mappano a classi, le righe delle stesse ad oggetti, le colonne ad attributi. Differisce da molti altri prodotti ORM per la maniera in cui è configurato. Disponendo da subito di un gran numero di configurazioni di default, ActiveRecord minimizza l'ammontare di configurazione richiesta normalmente allo sviluppatore. (si pensi ad esempio a mappature tramite file XML)

ActionController

ActionController è il centro di un'applicazione Ruby On Rails. Coordina l'interazione tra utente, view e model. Molta di questa interazione in ogni caso avviene dietro le quinte, favorendo quindi il concentrarsi dell'attenzione dello sviluppatore sulle funzionalità dell'applicazione, rendendola più semplice da sviluppare e mantenere. ActionController svolge anche una serie di servizi accessori.

- È responsabile di dirigere le richieste esterne verso le corrette azioni interne; gestisce infatti tranquillamente la riscrittura degli url.
- Gestisce la cache, che può garantire ad un'applicazione incrementi di performance anche di ordini di grandezza.
- Gestisce i cosiddetti moduli helper, che estendono le capacità delle view senza appesantirne il codice.
- Gesisce le sessioni, dando all'utente l'impressione di un'interazione continua con l'applicazione.

ActionView

ActionView è responsabile della creazione di tutta o parte della pagina che verrà mostrata a schermo. In estrema semplicità una view è un frammento di HTML che mostra del testo statico. Tipicamente però una view mostra contenuti dinamici, generati richiesta per richiesta dagli strati sottostanti. In

Rails, i contenuti dinamici sono generati tramite modelli, disponibili in tre tipologie.

- Erb (Embedded ruby), ovvero frammenti di codice ruby inseriti all'interno del codice della view tramite markup particolare.
- Generatore XML che costruisce documenti XML partendo direttamente da codice ruby con struttura specifica.
- Viste RJS, modelli di vista che permettono, tramite codice ruby, la creazione di frammenti di codice Javascript.

7.2 Modulo “Gestione delle Non Conformità”

Segue una descrizione dettagliata per ogni sottoinsieme di classi del diagramma UML relativo al modulo in oggetto.

Dettaglio dello Strato Model

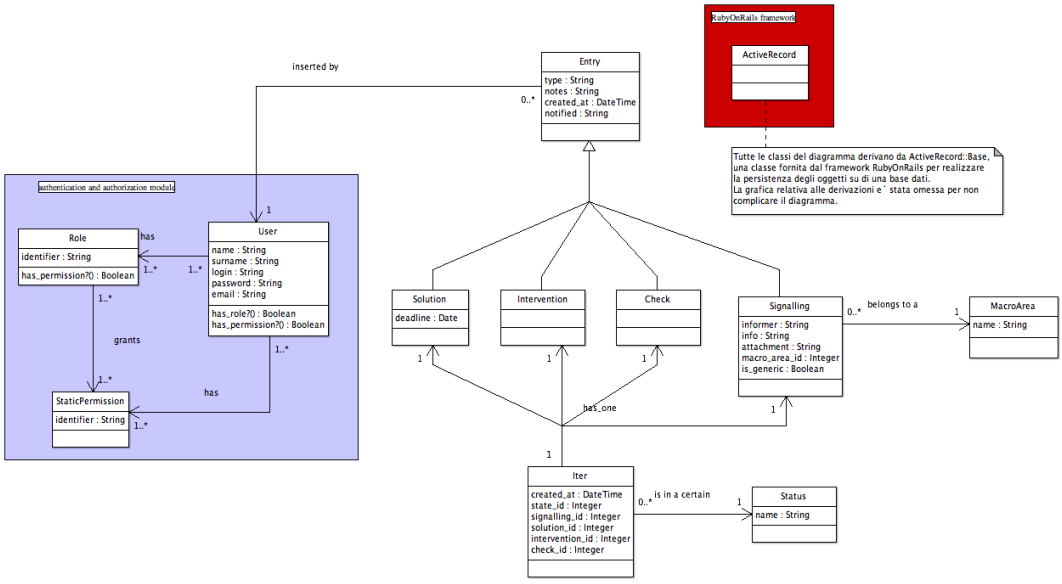

Figura 3.7: Componenti del Model del modulo “Gestione Non Conformità”

Questo diagramma descrive la composizione dello strato Model, le cui classi come già accennato derivano tutte dalla classe ActiveRecord::Base.

La gerarchia Entry modella una componente del ciclo di vita di una segnalazione di non conformità. Si specializza nelle classi:

- Signalling, la segnalazione (generica o di non conformità);
- Solution, ovvero la specifica della correzione individuata;
- Intervention, la registrazione della correzione effettuata;
- Check, il resoconto dell'azione di controllo sull'intervento effettuato;

Si è scelto di utilizzare l'ereditarietà dato l'importante sottoinsieme di funzionalità comuni alle suddette classi, che si differenziano tra loro essenzialmente per alcuni campi dato. Per dare organicità alla base dati, alla classe Signalling è stata associata una cosiddetta MacroArea che ne specificasse l'ambito di apppartenenza.

La classe Iter modella il ciclo di vita di una segnalazione di non conformità ovvero di una generica segnalazione. In ogni istante si trova in un certo stato definito dalla classe Status. Ogni stato è caratterizzato dalla presenza o meno delle istanze di Entry associate, in pratica l'avanzamento dell'iter è dato oltre che dal cambio di stato, dalla creazione ed aggiunta dell'istanza di Entry relativa allo stato corrente.

Dettaglio dello Strato Controller

Segue la descrizione dettagliata della composizione dello strato Controller del modulo "Gestione Non Conformità".

Il Namespace Admin racchiude tutti i controller deputati all'utilizzo da parte di un utente con particolari privilegi detto "amministratore". Un utente di questo tipo ha accesso a tutti gli aspetti dell'applicazione, dalla gestione dei metadati (le classi Status e MacroArea del Model) alla gestione dell'avanzamento dell'iter di una segnalazione. I controller del namespace `Admin` sono di due tipologie:

- **funzionali**
  - `OpenController` che gestisce le segnalazioni appena inserite e tramite il quale si può specificare una soluzione tramite l'action `solve()`;

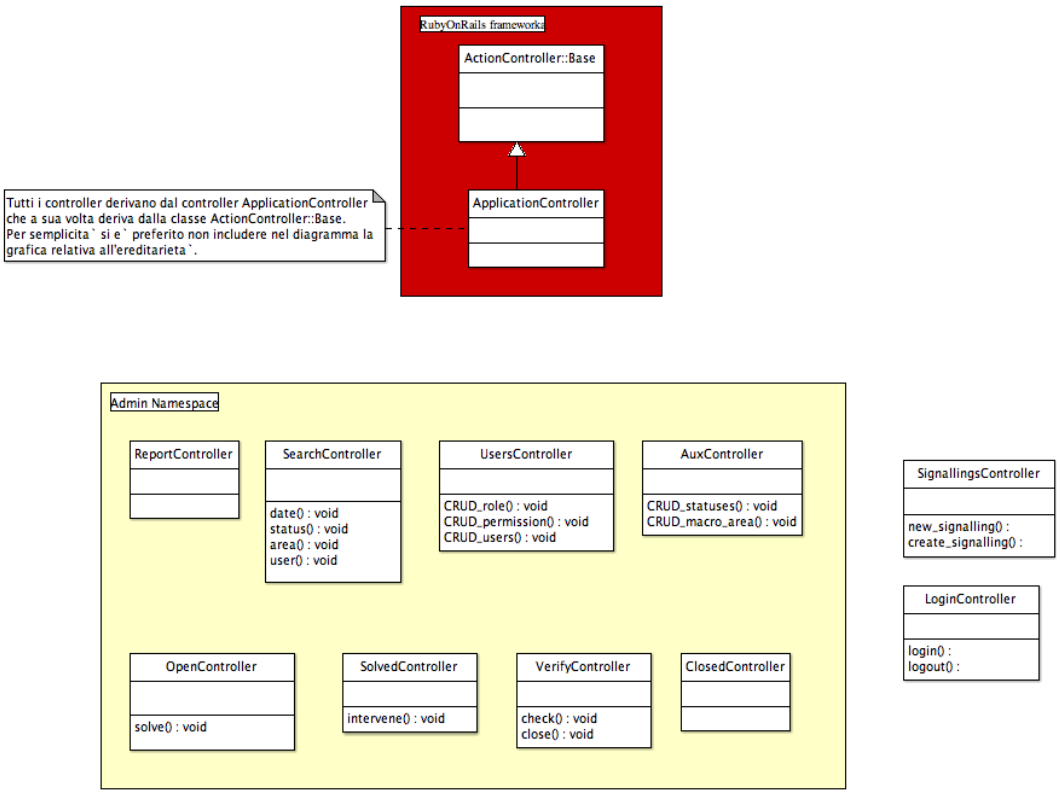

Figura 3.8: Componenti del Controller del modulo “Gestione Non Conformità”

  - `SolvedController` che permette di far avanzare l’iter di una segnalazione mediante l’action `intervene()`;
  - `VerifyController` dove si specifica l’avvenuta verifica dell’intervento correttivo tramite l’action `check()`e si può chiudere l’iter di gestione tramite l’action `close()`;
  - `ClosedController` che permette di visualizzare tutti gli iter giunti a conclusione.

- **ausiliari**
  - `ReportController` che svolge funzioni di reportistica in vari formati;
  - `SearchController` che come suggerisce il nome permette di effettuare ricerche su vari parametri (data, stato, area, utente) all’interno della base dati degli iter;
  - `UsersController` che consente la gestione degli utenti e dei livelli di permesso di accesso all’applicazione;

  - `AuxController` che si occupa della gestione dei metadati necessari al corretto funzionamento dell'applicazione

Controller Utente come mostrato in Figura 3.8, al di fuori del namespace `Admin` vi sono 2 controller, gli unici accessibili da un utente generico (senza particolari privilegi), che permettono solamente l'inserimento delle segnalazioni, siano esse generiche o di non conformità, la visione di soluzioni assegnate a se stessi ed ovviamente l'accesso all'applicazione.

Dettaglio dello Strato View

La modellazione dello strato View in Ruby On Rails è diretta coseguenza della progettazione dello strato sottostante ovvero Controller. Tipicamente ogni metodo di una classe controller (detto action) ha associate delle viste sottoforma di template ma può in realtà accedere anche a viste di altre action se non addirittura di altri controller.

Plugin e Componenti Accessori

Durante la fase di progettazione si è ritenuto opportuno utilizzare alcune componenti già pronte, dette plugin, realizzate proprio per permettere allo sviluppatore Ruby On Rails di riutilizzare con il minimo sforzo componenti altrui.

- *ActiveRbac* controllo degli accessi basato su ruoli per Ruby On Rails, come da Figura 3.7 mette a disposizione il model User, Role e Static Permission, permettendo di scegliere il livello di granularità dei permessi desiderato.
- *ActiveScaffold* sistema di scaffold che va oltre il semplice aiuto in fase di prototipazione ma che permette di realizzare applicazioni anche complesse con poche semplici direttive di configurazione.
- *ActiveScaffoldLocalize* questo plugin si occupa della traduzione delle stringhe presenti all'interno del plugin precedente.
- *ActiveScaffoldTools* questo plugin estende ActiveScaffold aggiungendo funzionalità di stampa ed esportazione dei dati in vari formati.
- *Engines* plugin nato per permettere l'integrazione di componenti strutturate come applicazioni a sè stanti piuttosto che come classici plugin. Vi dipende ActiveRbac.

- *Globalite* plugin che si occupa della traduzione generale del testo presente in un'applicazione.
- *RFPDF* plugin che contiente una libreria per la generazione di documenti PDF, utilizzato per la reportistica.

8 Codifica

8.1 Generazione Codice

La fase di codifica è iniziata con la generazione del codice relativo ai diagrammi UML. La generazione è avvenuta mediante gli strumenti messi a disposizione dal framework, denominato per l'appunto generatori, che consentono di automatizzare le operazioni elementari di codifica come:

- creazione dei file e cartelle necessari;
- dichiarazione delle classi;
- creazione infrastruttura di test;
- integrazione delle funzionalità dei plugin nella base di codice dell'applicazione.

Segue un esempio di procedura di generazione di una classe del Model con relativi risultati.
Esecuzione del generatore:

```
script/generate model Entry
        exists  app/models/
        exists  test/unit/
        exists  test/fixtures/
        create  app/models/entry.rb
        create  test/unit/entry_test.rb
        create  test/fixtures/entries.yml
        exists  db/migrate
        create  db/migrate/001_create_entries.rb
```

Il contenuto dei file generati è il seguente, la classe Entry:

```
class Entry < ActiveRecord::Base
end
```

l'infrastruttura di test:

```
test/unit/entry_test.rb

require File.dirname(__FILE__) + '/../test_helper'

class EntryTest < Test::Unit::TestCase
  fixtures :entries

  # Replace this with your real tests.
  def test_truth
    assert true
  end
end

test/fixtures/entries.yml

# Read about fixtures at http://ar.rubyonrails.org/classes/Fixtures.html
one:
  id: 1
two:
  id: 2
```

infine la cosiddetta migration, file che contiene il codice necessario per creare e mantenere la struttura del Model sulla base dati effettiva:

```
class CreateEntries < ActiveRecord::Migration
  def self.up
    create_table :entries do |t|

    end
  end

  def self.down

  end
end
```

8.2 Estensione del Codice Generato

Premessa

Una volta generati tutti i file necessari si è proceduto con la codifica vera e propria delle parti mancanti. L'inesperienza dello stagista con framework di sviluppo di questo tipo ha favorito l'adozione di un approccio di codifica bottom up, realizzando quindi prima la parte Model, in seguito il Controller e per finire le viste. È invece prassi consolidata prototipare l'applicazione, quindi estendere il prototipo in maniera top-down delegando continuamen-

te agli strati inferiori sino a quando non è più possibile. In questo modo si cerca di garantire il ruolo dei singoli strati, lasciando alle viste il compito di mostrare i dati e interagire con l'utente, al Controller il semplice compito di scambio dati e delegando tutta la business logic allo strato Model.

Con riferimento all'esempio della sezione 8.1 si descriveranno ora nel dettaglio le principali attività della codifica.

Codifica delle classi model con definizione:

- relazioni (righe 2 e 3)
- validazioni (riga 5)
- callback (righe 7-12)
- metodi (righe 14-25)

```
1  class Signalling < Entry
2    belongs_to :macro_area
3    has_one :iter
4
5    validates_presence_of :notes, :macro_area
6
7    before_save do |record|
8      iter = Iter.new
9      iter.state = State.find_by_name 'Aperta'
10     record.iter = iter
11     record.user = current_user
12   end
13
14   def to_label
15     if is_generic then
16       "Vedi segnalazione"
17     else
18       "Vedi non conformitÃ "
19     end
20   end
21
22   attr_reader :allegato
23
24   private
25   attr_accessor :notified
26 end
27
28 class FileTypeException < Exception
29
30 end
```

Codifica delle migration con definizione dello schema del database sull'esatto modello del Model; Le migration sono incrementali, ciò significa che è possibile versionare il proprio schema di base dati e poter in ogni momento dello sviluppo tornare ad una versione piuttosto che un'altra.

- Il metodo statico `up()` porta lo schema alla versione della migration dove è definito.
- Il metodo statico `down()` come facilmente intuibile porta lo schema alla versione precedente la migration in cui è definito.

```
1  class CreateEntries < ActiveRecord::Migration
2    def self.up
3      create_table :entries do |t|
4        t.column :type, :string
5
6        # attributi comuni alle sottoclassi
7        t.column :notes, :text
8        t.column :created_at, :datetime
9        t.column :user_id, :integer
10       t.column :notified, :string
11     end
12   end
13
14   def self.down
15     drop_table :entries
16   end
17 end
```

Codifica delle classi Controller e relative viste. La prima sezione di questo controller consta di alcune chiamate a metodi statici, tipicamente utilizzate per dare direttive di configurazione. Qui nell'ordine troviamo:

- definizione di un filtro (riga 3)
- override della configurazione di default di ActiveScaffold (righe 6-18)

```
1  class Admin::ClosedController < ApplicationController
2
3    append_before_filter :update_table_config
4
5    # Configurazione di ActiveScaffold per questo controller
6    active_scaffold :iter do |config|
7      config.label = "Segnalazioni verificate e chiuse"
8      config.list.sorting = {:created_at => :desc}
9      config.columns = [:selected, :created_at, :state, :signalling, :solution
           , :intervention, :check]
10     columns[:created_at].label = 'Data inserimento'
```

```
11      columns[:state].label = 'Stato'
12      columns[:signalling].label = 'Segnalazione'
13      columns[:solution].label = 'Soluzione'
14      columns[:intervention].label = 'Intervento'
15      columns[:check].label = 'Verifica'
16      columns[:selected].label = ''
17      columns[:selected].form_ui = :checkbox
18    end
19
20    # Banale action per l'index
21    def index
22      render :template => 'admin/closed/index'
23    end
24
25    private
26
27    def update_table_config
28      swap_actions
29    end
30
31    # Override dell'action per la stampa od esportazione dei record.
32    # Ricerca tutti quei record che sono stati selezionati tramite
33    # checkbox e li ritorna ad active_scaffold_tools per la stampa od
34    # esportazione effettiva.
35    def do_print_list(print_list_config)
36      unless (session[:selected_export]['search'].nil? || session[:
          selected_export]['search'].size == 0) then
37        pager = ::Paginator.new(controller_name, 0) do |offset, per_page|
38          Iter.find(:all,
39            :conditions => ['iters.id IN (?)', session[:selected_export]['
                search']])
40        end
41      else
42            pager = ::Paginator.new(controller_name, 0) do |offset, per_page|
43                Iter.find(:all,
44                  :conditions => ['iters.id IN (?)', session[:
                        selected_export]["#{controller_name}"]])
45              end
46             end
47             @records = pager.page(1)
48           end
49         end
```

Una vista come già menzionato altro non è se non codice di markup nel quale viene inserito direttamente del codice ruby.

```
app/views/admin/show_signalling.rhtml
        <div>
          <dl>
            <dt><h5>Data inserimento:</h5></dt>
            <dd><%=h @entry.created_at %></dd>
            <dt><h5>Note:</h5></dt>
```

```
            <dd><%=h @entry.notes %></dd>
            <dt><h5>Segnalatore:</h5></dt>
            <dd><%=h "#{@entry.user.print_data}" %></dd>
            <dt><h5>Area:</h5></dt>
            <dd><%=h @entry.macro_area.name unless @entry.macro_area.nil?
                %></dd>
            <dt><h5>Informatore:</h5></dt>
            <dd><%=h @entry.informer %></dd>
            <dt><h5>Informazioni:</h5></dt>
            <dd><%=h @entry.info %></dd>
            <dt><h5>Scarica allegato:</h5></dt>
            <dd><%= (link_to File.basename(@entry.attachment), "../#{@entry.
                attachment}") unless @entry.attachment.nil? %></dd>
          </dl>
        </div>
```

8.3 ActiveScaffold

Un ruolo molto importante per la fase di codifica lo ha svolto il plugin ActiveScaffold. Si tratta come accennato di un plugin che permette di creare un'interfaccia cosiddetta CRUD per un dato model con poche righe di configurazione. Questo plugin fonda la sua potenza e flessibilitŝ sul processo della reflection proprio del linguaggio ruby. Data una qualsiasi classe model (che derivi quindi da ActiveRecord::Base) il plugin è in grado di raccogliere tutte le informazioni necessarie per rendere le proprie viste e controller perfettamente compatibili con il model in oggetto. Per chiarire prendiamo ad esempio un frammento di codice derivato dal modulo deputato a controller generico.

```
1  vendor/plugins/active_scaffold/lib/actions/create.rb
2
3  module ActiveScaffold::Actions
4    module Create
5      def self.included(base)
6        base.before_filter :create_authorized?, :only => [:new, :create]
7        base.verify :method => :post,
8                    :only => :create,
9                    :redirect_to => { :action => :index }
10     end
11
12         ...
13
14       def do_create
15       begin
16         active_scaffold_config.model.transaction do
17           @record = update_record_from_params(active_scaffold_config.model.
                 new, active_scaffold_config.create.columns, params[:record])
18           apply_constraints_to_record(@record, :allow_autosave => true)
19           before_create_save(@record)
```

```
20            self.successful = [@record.valid?, @record.associated_valid?].all?
                  {|v| v == true} # this syntax avoids a short-circuit
21            if successful?
22              @record.save! and @record.save_associated!
23              after_create_save(@record)
24            end
25          end
26        rescue ActiveRecord::RecordInvalid
27        end
28      end
29
30            # override this method if you want to inject data in the record
                  (or its associated objects) before the save
31            def before_create_save(record); end
32
33            # override this method if you want to do something after the
                  save
34            def after_create_save(record); end
35
36          ...
37
38    end
39  end
```

Come si può osservare, alla riga 17

```
@record = update_record_from_params(active_scaffold_config.model.new,
    active_scaffold_config.create.columns, params[:record])
```

vi è l'istanziazione di una variabile (che diverrà un record sulla base dati) mediante l'operatore `new` richiamato su di una struttura dati contente l'informazione riguardo la classe su cui il plugin è stato configurato. Questa struttura dati all'atto della sua istanziazione (nel controller, cfr. sezione 8.2, par. "Codifica delle classi Controller") riceve tutta una serie di informazioni utilizzate poi all'interno di codice che nulla presume a prescindere sui dati che sta manipolando.

Il risultato è il seguente.

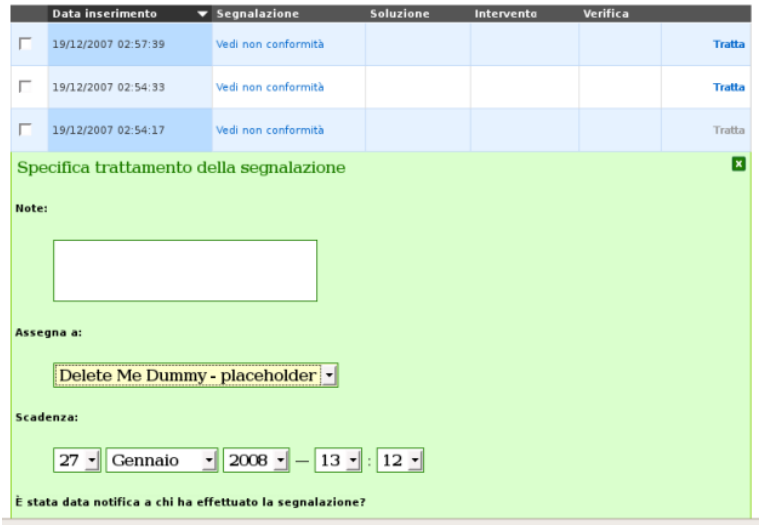

Figura 3.9: Lista record e maschera inserimento per il modulo "Gestione Non Conformità" realizzate mediante ActiveScaffold

9 Qualifica

Anche nella fase di Qualifica, come nelle precedenti, le scelte principali sono state dettate dal framework utilizzato. Nel periodo i cui si è svolto lo stage l'approccio più condiviso era il cosiddetto "test driven development" (TDD) che consiste nel codificare i test delle funzionalità previste prima ancora di averle sviluppate.

- Si aggiunge un test, che inizialmente fallirà data l'assenza della funzionalità da testare.
- Si codifica la funzionalità di modo che il test abbia successo.
- Si procede al refactoring del codice applicativo scritto al passo precedente.
- Si ripete il ciclo.

Questo ciclo di codifica viene denominato *red/green/refactor* dove *red* significa che il test fallisce e *green* che ha successo.

Strategia di Qualifica

Le precedenti considerazioni purtroppo sono state apprese dallo studente successivamente alla fase di Qualifica che ha quindi visto l'adozione di una strategia differente ovvero si è proceduto alla codifica dei test solamente dopo

che la maggior parte delle funzionalità erano state codificate, a sola conferma della correttezza del codice. Ruby On Rails contiene a sua volta un framework di testing chiamato Test::Unit grazie al quale è possibile sfruttare un'infrastruttura automatizzata di test che consente tra le altre cose di mantenere l'ambiente di test separato da quello di sviluppo. Data la differente complessità dei vari livelli dell'applicazione e per questioni di tempistiche si è data la preferenza alla codifica dei soli test di unità dello strato Model, strato in cui si concentra la business logic.

Esito della Qualifica

Capitolo 4

Valutazione Retrospettiva

Dove si traccia l'impegno posto nello svolgimento dello stage, si analizzano tematiche di carattere didattico e si analizza il processo di adozione del prodotto realizzato.

L'attività di stage ha seguito l'andamento della previsione nel piano di lavoro. È stato tuttavia necessario intervenire successivamente presso l'istituto scolastico ospitante per piccoli aggiustamenti dovuti all'infrastruttura informatica in loco. Si è verificato così un certo ritardo nell'effettiva conclusione dei lavori, a causa di questa serie di attività non previste.

La campagna di validazione è iniziata con le varie revisioni formali presso la committenza per tracciare di volta in volta i progressi sull'insieme dei requisiti. Poco prima del collaudo si è tenuta una presentazione del prodotto presso la committenza per raccogliere evidenza del fatto che i prodotto soddisfacesse i requisiti.

L'utilizzo di una nuova tecnologia ha fatto si che le conoscenze necessarie allo svolgimento dello stage siano state apprese in una fase preliminare prevista nel piano di lavoro per colmare ogni lacuna appunto.
Fondamentali sono state le nozioni apprese durante il corso di laurea, nella fattispecie dai corsi di Ingegneria del Software, nozioni che si è cercato sempre di applicare al problema specifico quando possibile.

Appendice A

Glossario

in corso di redazione