



DONKEYCODE

RESCUEME: SISTEMA GLOBALE PER LA GESTIONE
DI CATASTROFI NATURALI

Piano di qualifica

Versione 4.0.0

Ingegneria Del Software AA 2010-2011

Informazioni documento

Titolo documento:	Piano di qualifica
Data creazione:	9 Dicembre 2010
Data rilascio:	28 Marzo 2011
Versione attuale:	4.0.0
Utilizzo:	Esterno
Nome file:	Piano-di-qualifica-4.0.0.pdf
Redazione:	De Gaspari Fabio
Revisione:	Bonotto Sara
Approvazione:	Vazzoler Raffaele
Distribuito da:	DonkeyCode
Destinato a:	Prof. Vardanega Tullio Miriade S.p.A.

Sommario

Nel presente documento sono descritte le modalità di qualifica che il gruppo intende adottare a partire dalla stesura dei primi documenti e a seguire per tutto il ciclo di vita del progetto.

Registro delle modifiche

Versione	Data	Persone coinvolte	Descrizione
4.0.0	28-03-2011	Verificatore: <i>Malorgio Antonio</i> Approvatore: <i>De Gaspari Fabio</i>	Rilascio documento per revisione di accettazione.
3.2.3	25-03-2011	Redattore: <i>De Gaspari Fabio</i>	Effettuate correzioni ortografiche e di punteggiatura.
3.2.2	23-03-2011	Redattore: <i>Vazzoler Raffaele</i>	Aggiornati risultati dei test e dell'analisi statica.
3.2.1	23-03-2011	Redattore: <i>Vazzoler Raffaele</i>	Aggiornato il tracciamento.
3.2.0	22-03-2011	Redattore: <i>De Gaspari Fabio</i>	Aggiunti risultati sintetici dei test.
3.1.0	21-03-2011	Redattore: <i>Crosato Emanuele</i>	Effettuati miglioramenti richiesti in revisione di qualifica.
3.0.0	14-03-2011	Verificatore: <i>Bonotto Sara</i> Approvatore: <i>Vazzoler Raffaele</i>	Rilascio documento per revisione di qualifica.
2.3.1	14-03-2011	Redattore: <i>De Gaspari Fabio</i>	Effettuate correzioni ortografiche e di punteggiatura.
2.3.0	11-03-2011	Redattore: <i>De Gaspari Fabio</i>	Aggiunti risultati dei test e dell'analisi statica.
2.2.0	21-02-2011	Redattore: <i>De Gaspari Fabio</i>	Aggiornato il tracciamento.
2.1.0	21-02-2011	Redattore: <i>Crosato Emanuele</i>	Effettuate correzioni emerse dalla revisione di progetto.

Versione	Data	Persone coinvolte	Descrizione
2.0.0	31-01-2011	Verificatore: <i>Tronchin Luca</i> Approvatore: <i>Bonotto Sara</i>	Rilascio documento per revisione di progetto.
1.5.1	28-01-2011	Redattore: <i>Crosato Emanuele</i>	Effettuate correzioni ortografiche e di punteggiatura.
1.5.0	27-01-2011	Redattore: <i>De Gaspari Fabio</i>	Aggiunto tracciamento requisito-componente e viceversa.
1.4.0	24-01-2011	Redattore: <i>De Gaspari Fabio</i>	Definite metriche. Aggiunto capitolo “Re-soconto delle attività di verifica” e “Test di integrazione”.
1.3.0	20-01-2011	Redattore: <i>Crosato Emanuele</i>	Aggiunto capitolo “Pianificazione ed esecuzione collaudo”.
1.2.0	19-01-2011	Redattore: <i>Crosato Emanuele</i>	Definiti metodi di valutazione e test di accettazione.
1.1.0	14-01-2011	Redattore: <i>Crosato Emanuele</i>	Effettuate correzioni emerse dalla revisione dei requisiti. Corrette incongruenze terminologiche e parte relativa a SQA.
1.0.0	20-12-2010	Verificatore: <i>Sotomayor Jorge</i> Approvatore: <i>Tronchin Luca</i>	Rilascio documento per revisione dei requisiti.
0.2.0	16-12-2010	Redattore: <i>Vazzoler Raffaele</i>	Aggiunto capitolo riguardante le metriche.
0.1.0	14-12-2010	Redattore: <i>Vazzoler Raffaele</i>	Prima stesura del documento.

Indice

1	Introduzione	1
1.1	Scopo del documento	1
1.2	Scopo del prodotto	1
1.3	Glossario	1
1.4	Riferimenti	1
1.4.1	Normativi	1
1.4.2	Informativi	2
2	Obiettivi di qualità	3
2.1	Qualità di processo	3
2.2	Qualità di prodotto	5
2.3	Processi per la gestione della qualità	8
3	Strategia di verifica	9
3.1	Organizzazione, pianificazione strategica e temporale, responsa- bilità	9
3.1.1	Pianificazione strategica e temporale	10
3.1.2	Organizzazione	10
3.1.3	Responsabilità	11
3.2	Risorse necessarie, risorse disponibili	11
3.2.1	Risorse necessarie	11
3.2.2	Risorse disponibili	12
3.3	Strumenti, tecniche e metodi	12
3.3.1	Strumenti per l'analisi	12
3.3.2	Tecniche di analisi	13
3.3.3	Strategie per la verifica	16
4	Metriche di misurazione	18
5	Gestione amministrativa della revisione	21
5.1	Comunicazione e risoluzione anomalie	21
5.2	Procedure di controllo di qualità di processo	21
5.2.1	Software Quality Assurance	21
6	Dettaglio dell'esito delle revisioni	22
6.1	Revisione dei requisiti	22
6.2	Revisione di progetto	22
6.3	Revisione di qualifica	23

A	Pianificazione ed esecuzione collaudo	24
A.1	Specifica della campagna di validazione	24
A.1.1	Collaudo	24
A.1.2	Test di accettabilità	24
A.1.3	Test di integrazione delle componenti principali	26
A.1.4	Test di integrazione delle sotto-componenti	28
A.1.5	Test di unità	29
A.2	Configurazione dell'ambiente di build	30
B	Resoconto delle attività di verifica	34
B.1	Tracciamento	34
B.1.1	Packages naming	34
B.1.2	Componenti - requisiti	35
B.1.3	Requisiti - componenti	44
B.2	Esito della campagna di analisi dinamica	53
B.2.1	β -testing e test di accettabilità	53
B.2.2	Test effettuati	53
B.2.3	Test di integrazione delle componenti principali	56
B.2.4	Test di integrazione delle sotto-componenti	57
B.2.5	Test di unità	58
B.3	Copertura del codice sorgente	64
B.4	Esito della campagna di analisi statica	72
B.4.1	Metriche RescueMe	72
B.4.2	Metriche RescueApp	75

1 Introduzione

1.1 Scopo del documento

Lo scopo del presente documento è quello di definire e pianificare la strategia e le modalità di validazione e verifica che si ha intenzione di adottare durante lo sviluppo del progetto, al fine di rilevare e correggere anomalie, difetti e incongruenze all'interno del sistema RescueMe_[g].

1.2 Scopo del prodotto

Il progetto ha lo scopo di realizzare un sistema software, RescueMe, per la gestione e coordinazione di segnalazioni e richieste di aiuto in caso di catastrofi naturali. Si tratta di un sistema che fornisce un'interfaccia web utilizzabile sia dalle persone coinvolte nelle catastrofi che dalle autorità che gestiscono tali eventi. In particolare permette alla popolazione di immettere delle richieste di aiuto e di visualizzarne lo stato di avanzamento. Per quanto riguarda le autorità, invece, permette agli operatori_[g] di elaborare le richieste di aiuto e agli *amministratori di autorità*_[g] di gestire le varie catastrofi e gli operatori stessi. Il sistema mette a disposizione della popolazione un'ulteriore interfaccia di tipo mobile.

1.3 Glossario

I termini che compaiono affiancati dal pedice [g] sono descritti nel documento Glossario-3.0.0.pdf che accompagna e completa il presente.

1.4 Riferimenti

1.4.1 Normativi

Il documento Norme-di-progetto-4.0.0.pdf regola e accompagna il qui presente e tutti i documenti ufficiali.

Capitolato d'appalto RescueMe

(www.math.unipd.it/~tullio/IS-1/2010/Progetto/RescueMe.pdf).

1.4.2 Informativi

ISO/IEC 25000:2005

(www.sei.cmu.edu/library/assets/esepg.pdf).

ISO/IEC 15504:1998

(www2.cnipa.gov.it/site/_contentfiles/00310300/310320_15504.pdf).

ISO/IEC 12207:2008

(www.cnipa.gov.it/site/_files/ISO12207.pdf).

ISO/IEC 15939:2007 Software Measurement Process

(http://www2.cnipa.gov.it/site/_contentfiles/01379900/1379952_ISO%2014598.pdf).

Metriche e misurazioni RescueMe

(<http://donkeycode2010.altervista.org/rescuemetriche>).

Metriche e misurazioni RescueApp

(<http://donkeycode2010.altervista.org/rescueappmetriche>).

Copertura del codice

(<http://donkeycode2010.altervista.org/coberturaresults>).

2 Obiettivi di qualità

La seguente sezione cerca di definire e descrivere sia gli obiettivi di qualità che il prodotto richiesto dal committente intende soddisfare, sia quelli relativi ai processi attraverso i quali tale prodotto verrà definito.

2.1 Qualità di processo

Per garantire qualità nel prodotto bisogna necessariamente garantire qualità nei processi che ne compongono lo sviluppo. Il gruppo quindi si impegna a garantire e migliorare continuamente la qualità dei processi impiegati. A tal proposito è stato adottato lo standard ISO/IEC 15504:1998 SPICE “Software Process Improvement Capability dEtermination”, che definisce il modello SPA-I “Software Process Assessment & Improvement” per la valutazione e miglioramento della qualità dei processi.

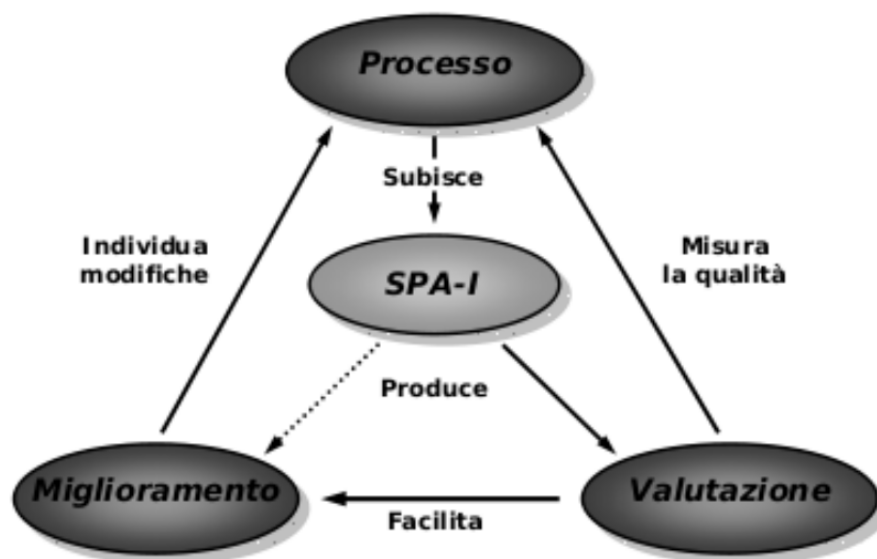


Figura 1: Modello SPA-I

Ogni processo deve essere valutato. Questa valutazione permetterà di misurare la sua qualità e ne faciliterà il miglioramento. Il modello dei processi del ciclo di vita utilizzato come riferimento è quello della norma *ISO/IEC 12207:2008*_[g], esteso con processi riguardanti la gestione del riuso, la gestione delle risorse umane, il miglioramento e la misura. Lo SPICE definisce inoltre nove attributi di processo:

1. Process performance: un processo raggiunge i suoi obiettivi, trasformando input identificabili in output identificabili;
2. Process management: l'attuazione di un processo è pianificata e controllata al fine di produrre risultati che rispondono agli obiettivi attesi;
3. Work product management: l'attuazione di un processo è pianificata e controllata al fine di produrre risultati che siano appropriatamente documentati, controllati e verificati;
4. Process definition: l'attuazione di un processo si basa su approcci standardizzati;
5. Process resource: il processo può contare su risorse adeguate per essere eseguito;
6. Process measurement: i risultati raggiunti e le misure rilevate durante l'attuazione di un processo sono usati per assicurare che l'esecuzione di tale processo supporti efficacemente il raggiungimento di specifici obiettivi;
7. Process control: un processo è controllato attraverso la raccolta, l'analisi e l'utilizzo delle misurazioni sul prodotto e sul processo, al fine di correggere, se necessario, le sue modalità di attuazione;
8. Process change: le modifiche a definizione, gestione ed attuazione di un processo sono controllate;
9. Continuos integration: le modifiche ad un processo sono identificate ed implementate al fine di assicurare il continuo miglioramento nel raggiungere obiettivi rilevanti.

La norma stabilisce inoltre quattro livelli di possesso:

- N (0-15%) Non posseduto;
- P (16-50%) Parzialmente posseduto;
- L (51-84%) Largamente posseduto;
- F (86-100%) Fully, pienamente posseduto.

In base al livello di possesso degli attributi, sono previsti 6 livelli di “maturità” dei processi:

- Livello 0: processo incompleto;
- Livello 1: processo semplicemente attuato;
- Livello 2: processo gestito;

- Livello 3: processo definito;
- Livello 4: processo predicibile;
- Livello 5: processo ottimizzante.

I benefici che emergono dall'uso di questo standard riguardano sia gli sviluppatori che gli utenti/acquirenti. Per i primi questi saranno:

- ottimizzazione dell'uso delle risorse;
- contenimento dei costi;
- maggiore tempestività di consegna;
- migliore stima dei rischi e degli impegni;
- possibilità di confrontarsi con delle best practices.

Per quanto riguarda invece utenti/acquirenti si ha maggior facilità nel:

- selezionare i fornitori;
- valutare i rischi del progetto;
- controllare lo stato di avanzamento in corso d'opera;
- ridurre i costi di correzione degli errori;
- controllare rischi e varianti in corso d'opera.

2.2 Qualità di prodotto

La qualità nel prodotto risulta di difficile valutazione poiché il software è immateriale. Esistono tuttavia metodi per ottenere buoni risultati, come l'utilizzo di metriche e misurazioni, regole per l'interpretazione delle misure effettuate e la loro valutazione. Inoltre si stabiliranno criteri di accettazione che permetteranno di valutare se il prodotto in sviluppo sarà di buona qualità.

Il gruppo DonkeyCode si pone l'obiettivo di sviluppare il prodotto perseguendo alcune delle caratteristiche definite dallo standard ISO/IEC 9126:2001 confluito nello ISO/IEC 25000:2005. Tali caratteristiche sono di seguito descritte.

- **Funzionalità:** la capacità di un prodotto software di fornire funzioni che soddisfano esigenze stabilite, necessarie per operare sotto condizioni specifiche. Questo comprende:
 - Appropriatezza: la capacità del prodotto software di fornire un appropriato insieme di funzioni per i compiti specificati e per gli obiettivi prefissati dall'utente;

- Accuratezza: la capacità del prodotto software di fornire i risultati concordati o precisi effetti richiesti;
 - Interoperabilità: la capacità del prodotto software di interagire ed operare con più sistemi specificati;
 - Aderenza: la capacità del prodotto software di aderire a standard, convenzioni e regolamentazioni rilevanti rispetto al settore operativo a cui vengono applicate.
- **Sicurezza**: la capacità del prodotto software di proteggere informazioni e dati, impedendo che persone o sistemi non autorizzati possano accedervi o modificarli. Questo comprende:
 - Affidabilità: la capacità del prodotto software di mantenere uno specificato livello di prestazioni quando usato in date condizioni per un determinato periodo;
 - Maturità: la capacità di un prodotto software di evitare che si verificano errori, malfunzionamenti o siano generati risultati errati;
 - Tolleranza agli errori: la capacità di mantenere livelli predeterminati di prestazioni anche in presenza di malfunzionamenti o usi scorretti del prodotto;
 - Recuperabilità: la capacità di un prodotto di ripristinare il livello appropriato di prestazioni e di recuperare le informazioni rilevanti in seguito a malfunzionamenti. A seguito di un errore, il software può risultare non accessibile per un determinato periodo di tempo. Tale periodo è valutato proprio dalla caratteristica di recuperabilità;
 - Aderenza: la capacità di aderire a standard, regole e convenzioni inerenti all'affidabilità.
 - **Efficienza**: la capacità di fornire appropriate prestazioni relativamente alla quantità di risorse usate. Questo comprende:
 - Comportamento rispetto al tempo: la capacità di fornire adeguati tempi di risposta ed elaborazione sotto condizioni determinate;
 - Utilizzo delle risorse: la capacità di utilizzo di quantità e tipo di risorse in maniera adeguata;
 - Aderenza: la capacità di aderire a standard e specifiche sull'efficienza.
 - **Usabilità**: la capacità del prodotto software di essere compreso, usato e ben accettato dall'utente, quando usato sotto determinate condizioni. Questo comprende:

- Comprensibilità: la facilità di comprensione dei concetti del prodotto, mettendo l'utente in grado di comprendere se il software è appropriato;
 - Apprendibilità: la capacità di ridurre l'impegno richiesto agli utenti per imparare ad usare l'applicazione;
 - Operabilità: la capacità di mettere in condizione gli utenti di farne uso per i propri scopi e di controllarne l'utilizzo;
 - Attrattiva: la capacità del software di essere piacevole per l'utente;
 - Aderenza: la capacità del software di aderire a standard o convenzioni relativi all'usabilità.
- **Manutenibilità**: la capacità del software di essere modificato, includendo correzioni, miglioramenti o adattamenti. Questo comprende:
 - Analizzabilità: la facilità con la quale è possibile analizzare il codice per localizzarvi possibili errori;
 - Modificabilità: la capacità del prodotto software di permettere l'implementazione di una specificata modifica (sostituzioni componenti);
 - Stabilità: la capacità del software di evitare effetti inaspettati derivanti da modifiche errate;
 - Testabilità: la capacità di essere facilmente testato per validare le modifiche apportate al software;
 - Aderenza: la capacità del prodotto software di aderire a standard e convenzioni relative alla manutenibilità.
 - **Portabilità**: la capacità del software di essere trasportato da un ambiente di lavoro ad un altro. Questo comprende:
 - Adattabilità: la capacità del software di essere adattato a differenti ambienti operativi senza dover applicare modifiche diverse da quelle previste;
 - Installabilità: la capacità del software di essere installato in uno specificato ambiente;
 - Aderenza: la capacità del prodotto software di aderire a standard e convenzioni relative alla portabilità;
 - Sostituibilità: la capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti nello stesso ambiente.

Il raggiungimento di questi obiettivi di qualità interna ed esterna dovrebbe confluire nella qualità in uso del prodotto che osserva le seguenti caratteristiche:

- **Efficacia**: la capacità del software di mettere in grado gli utenti di raggiungere gli obiettivi specificati con accuratezza e completezza;

- **Produttività:** la capacità di mettere in grado gli utenti di spendere una quantità di risorse appropriate in relazione all'efficacia ottenuta in uno specifico contesto d'uso;
- **Soddisfazione:** la capacità del prodotto software di soddisfare gli utenti;
- **Sicurezza:** la capacità del prodotto software di garantire accettabili livelli di rischio riguardanti danni a persone, software ed apparecchiature o all'ambiente operativo d'uso.

Si cercherà inoltre di ottenere buona qualità producendo opportuna documentazione. Come richiesto nel capitolato le parti del sistema verranno documentate utilizzando strumenti specifici.

2.3 Processi per la gestione della qualità

Di seguito elencheremo le linee guida che il gruppo intende adottare per la definizione dei processi di qualifica che accompagneranno il progetto durante tutto il ciclo di vita. I processi si dividono nelle due seguenti categorie:

Software Quality Assurance (SQA_[g]): è un processo che definisce come ottenere qualità del software e come riconoscere che quest'ultimo abbia il livello di qualità richiesto attraverso la definizione di standard da applicare al processo di sviluppo. Gli standard sono di due tipi:

- standard di processo: si applicano ai processi;
- standard di prodotto: si applicano sui prodotti in uscita a processi.

Verifica e Validazione (V&V_[g]): sono due processi il cui scopo è rispondere alle domande: “stiamo costruendo il sistema nella maniera giusta?” (verifica) e “stiamo costruendo il sistema giusto?” (validazione). La verifica, se eseguita in maniera continua e costante, accerta che il sistema funzioni correttamente e senza errori; la validazione invece dimostra che il sistema software rispetti le attese del cliente.

3 Strategia di verifica

Nel corso dello sviluppo di progetto il gruppo DonkeyCode si impegna a seguire due principi. Uno è il KISS “Keep It Short & Simple”, solitamente utilizzato per la codifica, ma applicabile a tutta la progettazione e che richiama in parte il principio filosofico del “Rasoio di Occam”, secondo il quale la soluzione più semplice è la soluzione corretta. Poi vi è la “broken window theory” secondo la quale, per evitare il propagarsi incontrollato degli errori, quest’ultimi vanno segnalati e corretti il prima possibile.

3.1 Organizzazione, pianificazione strategica e temporale, responsabilità

L'organizzazione e la pianificazione del piano di qualità si basa sul modello a “V” di seguito riportato. Il modello mostra come, all'avanzare della specifica del sistema, vengono definite le attività di validazione nella fase di implementazione del sistema.

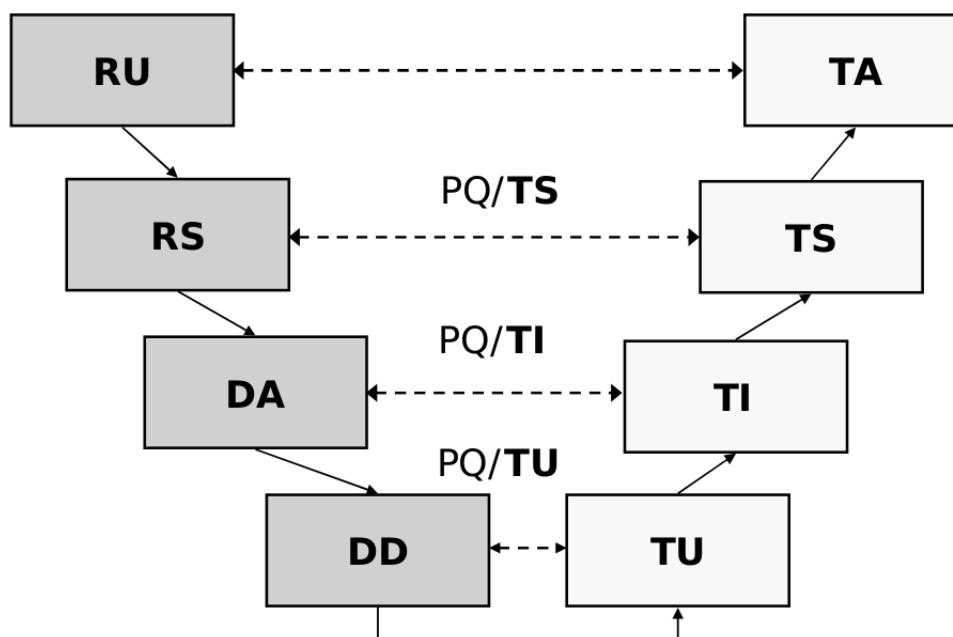


Figura 2: Modello a “V”.

Per l'intero sviluppo del progetto è stato scelto un ciclo di vita incrementale. Quest'ultimo influenza la pianificazione dei processi di validazione in quanto anch'essi rispettano l'implementazione ad incrementi del sistema.

3.1.1 Pianificazione strategica e temporale

La strategia di verifica e validazione sarà gestita dal responsabile, il quale assicurerà che:

- le scadenze, sia interne (definite dal gruppo) sia di revisione (definite dal committente), siano rispettate;
- ogni documento o unità software abbia copertura da parte di un verificatore incaricato.

Per controllare che il sistema si sviluppi nella maniera corretta è previsto un processo di revisione esterna condotta dal committente che corrisponde alla *Audit Process*_[g] nello standard ISO/IEC 12207:2008 e un processo di revisione interna che corrisponde alla *Joint Review Process*_[g] nello standard ISO/IEC 12207:2008. Il periodo di verifica seguirà quanto specificato nel documento Piano di progetto.

3.1.2 Organizzazione

La scelta del modello di ciclo di vita precedentemente citato è stata intrapresa in seguito allo studio di fattibilità e all'analisi di possibili problemi che si potrebbero incontrare nello sviluppo. Per il piano di qualità sono previste quattro fasi:

Analisi dei requisiti: in questa fase, che inizia con il termine della prima stesura dei documenti, quest'ultimi vengono verificati, sia dal punto di vista lessico/grammaticale che, in maniera più approfondita, dei contenuti esposti. In particolare, per quanto riguarda l'analisi dei requisiti, viene controllata la copertura delle richieste del proponente e la tracciabilità dei requisiti emersi. Il verificatore del documento controlla la correttezza e, nel caso siano richieste delle correzioni, apre un apposita *Issue*_[g], che può essere presa in carico solamente dal redattore. Quando il verificatore valida il documento, questo viene poi ulteriormente approvato dal responsabile per la presentazione al committente.

Progettazione: in questa fase si verifica la copertura e la congruenza dell'architettura di sistema progettata, rispetto ai requisiti emersi in analisi. Viene usato lo strumento ReTracker per il tracciamento dei requisiti al fine di facilitare questo compito. Le incongruenze o mancanze riscontrate da parte dei verificatori vengono segnalate attraverso lo strumento delle *Issue*, indicandone, se possibile, una eventuale soluzione.

Realizzazione: in questa fase viene verificato il codice scritto. La verifica è effettuata sia dai programmatori che dai verificatori, utilizzando appositi strumenti e controllando, per esempio, il rispetto delle *Java Code Convention*_[g]. Si utilizzano tecniche di

analisi statica e dinamica.

Validazione: il gruppo si impegna a presentare al collaudo una versione correttamente funzionante del prodotto in sviluppo. Eventuali oneri per l'eliminazione di difetti o non conformità sarà a totale carico del gruppo.

3.1.3 Responsabilità

Il responsabile del progetto prende a suo carico tutte le responsabilità sulle attività di verifica verso il committente e dirige il controllo dello svolgimento e dell'esito delle verifiche all'interno del gruppo. In particolare verificherà le modalità di sviluppo delle attività di verifica in modo che le scadenze siano rispettate.

3.2 Risorse necessarie, risorse disponibili

3.2.1 Risorse necessarie

Le risorse ritenute necessarie per il corretto svolgimento delle attività di verifica si dividono principalmente in tre categorie:

- **Risorse Umane**

- Responsabile di progetto: controlla la qualità interna ed è responsabile dello stato del progetto nei confronti del committente. Di conseguenza deve gestire la copertura di ogni elemento da parte dei verificatori.
- Amministratore di progetto: definisce delle regole e dei metodi all'interno del team di progetto affinché esse vengano rispettate ed attuate. Si occupa inoltre della gestione della documentazione di progetto.
- Programmatore: si occuperanno di verificare il codice da loro scritto usando strumenti e tecniche di debugging.
- Verificatori: presenti durante tutto il ciclo di vita del progetto, avranno come incarico la redazione del "Resoconto delle attività di verifica" in cui saranno riassunti i test e le verifiche effettuate sul prodotto software o su moduli di esso. Questo documento verrà poi presentato all'Amministratore di progetto che controllerà la completezza dei test e delle verifiche.

- **Risorse Software**

- Software per i test di unità compatibile con il linguaggio Java imposto dal proponente;
- software per i test di integrazione;

- software per metriche e misurazioni;
- software per il tracciamento.

- **Risorse Hardware**

Calcolatori con ambienti di lavoro adatti per sviluppo, analisi, testing e reporting del progetto.

3.2.2 Risorse disponibili

- **Risorse Umane**

Saranno definite in base alla pianificazione dei ruoli presente nel documento di Piano di progetto.

- **Risorse Software**

- ReTracker: software sviluppato internamente per il tracciamento dei requisiti attraverso le varie fasi del progetto e il controllo sullo stato di avanzamento delle classi tracciate. Permette inoltre il tracciamento con altre risorse quali: use case-requisiti-classi-test. Nel documento *Norme-di-progetto-4.0.0.pdf* e' presente il link al sistema.
- strumenti messi a disposizione da *Google Project Hosting*_[g] per il tracciamento delle anomalie.
- strumenti inclusi e configurabili resi disponibili dall'ambiente di sviluppo integrato_[g] *Eclipse 3.6 Helios*_[g].
- \LaTeX _[g] come motore di impaginazione per la stesura dei documenti.

- **Risorse Hardware**

Computer personali, portatili e fissi, in possesso ad ogni membro del gruppo. Computer dei laboratori di informatica del dipartimento di Matematica Pura ed Applicata.

3.3 Strumenti, tecniche e metodi

3.3.1 Strumenti per l'analisi

Di seguito saranno indicati alcuni degli strumenti in utilizzo o che si prevede di utilizzare per la verifica. Essi riguardano la stesura di documenti, la codifica, i test e le misurazioni. Nel caso siano esaminati altri strumenti ritenuti migliori la lista verrà aggiornata.ma

- *GNU Aspell*_[g]: strumento open-source_[g] per la correzione ortografica dei documenti. Può essere utilizzato come libreria o come correttore indipendente e può

verificare documenti in formato UTF-8_[g]. Per supportare correttamente la lingua italiana ha bisogno del dizionario appropriato;

- Eclipse 3.6 Helios_[g]: ambiente di sviluppo integrato multilinguaggio e multiplatforma;
- Eclipse Metrics Plugin_[g]: plug-in_[g] installabile su Eclipse per effettuare misurazioni di vario genere sul progetto;
- FindBugs_[g]: programma che cerca errori nel codice usando metodi di analisi statica;
- JUnit: framework per creare ed effettuare test di unità in linguaggio Java;
- Apache Ant_[g]: libreria Java e strumento da riga di comando il cui uso principale è il processo di build di applicazioni Java. Ant fornisce un numero di task integrati che permette di compilare, assemblare, testare ed eseguire applicazioni Java. Più in generale può essere utilizzato per pilotare qualsiasi tipo di processo che può essere descritto in termini di obiettivi e compiti. Ant è scritto in Java, è estremamente flessibile e non impone convenzioni di codifica o layout di directory per i progetti Java che lo adottano come uno strumento di compilazione.
- Cobertura_[g]: strumento che permette di verificare la copertura del codice da parte dei test di unità all'interno di un progetto. Permette di calcolare le seguenti informazioni sui sorgenti del progetto:
 - *Line coverage*_[g]: percentuale di linee di codice che vengono controllate dai test di unità.
 - *Branch coverage*_[g]: percentuale di diramazioni del codice che vengono controllate dai test di unità.
 - Complessità ciclomatica di ogni classe.

Le misurazioni prodotte da questo strumento sono essenziali per capire quali parti del progetto presentano carenze sui test di unità. Permette inoltre di individuare le classi più complesse da mantenere. Cobertura è integrabile direttamente con Ant per automatizzarne l'esecuzione sul codice sorgente. Genera automaticamente rapporti in formato HTML oppure XML.

3.3.2 Tecniche di analisi

Le tecniche di analisi sul software si dividono in due categorie:

Analisi statica: è una tecnica che ci permette di verificare il codice scritto usando la documentazione e i listati, senza far eseguire il codice. I metodi di lettura sul codice si dividono in:

- Inspection: lettura mirata del codice che evidenzia gli errori più frequenti nella scrittura codice sorgente;
- Walkthrough: lettura critica di tutto il codice senza particolare pianificazione.

È facile capire come il metodo dell'inspection sia preferibile al walkthrough, in quanto consente un notevole risparmio di risorse. Tuttavia c'è da considerare che l'inesperienza nel lavoro di gruppo rende questa tecnica poco efficace nel nostro caso. Di seguito sono definiti alcuni degli errori tipici commessi in fase di codifica, dei quali si terrà particolare conto durante l'inspection. Il linguaggio Java limita molti degli errori comunemente commessi sui dati e sulla gestione della memoria.

Tipologia d'errore	Controllo
Errori nei dati	Tutte le variabili di programma sono inizializzate prima che il loro valore sia utilizzato?
	Tutte le costanti hanno avuto un nome?
	Il limite superiore degli array è uguale alla loro dimensione o alla loro dimensione-1?
	Ci sono possibilità di buffer overflow?
Errori di controllo	Per ogni istruzione condizionale la condizione è corretta?
	E' certo che ogni ciclo sarà ultimato?
	Le istruzioni composte sono correttamente messe fra parentesi?
	Se è necessario un break dopo ogni caso nelle istruzioni case, è stato inserito?
Errori di input/output	Sono utilizzate tutte le variabili di input?
	A tutte le variabili di output viene assegnato un valore prima che siano restituite?
	Input imprevisti possono causare corruzione?
Errori di interfaccia	Tutte le chiamate a funzione e a metodo hanno il giusto numero di parametri?
	Il tipo di parametri formali e reali corrisponde?
	I parametri sono nel giusto ordine?
Errori di gestione delle eccezioni	Sono state prese in considerazione tutte le possibili condizioni di errore?

Si userà quindi in un primo momento il metodo di walkthrough e una volta raggiunto

un buon livello di conoscenza riguardo agli errori più comuni dei membri del gruppo addetti alla codifica, si passerà al metodo dell'inspection.

Ci sono svariate metodologie per fare analisi statica, oltre all'attività di inspection, differenti a seconda della criticità del prodotto. Durante lo sviluppo del progetto verranno usate le seguenti:

Analisi del flusso di controllo: segue il flusso nel Program Counter_[g], verificherà che tutti i flussi possibili eseguano sempre nella maniera progettata e che non ci siano parti del codice che non sono raggiungibili o che non terminano. Quest'ultima possibilità può essere generata attraverso chiamate ricorsive indirette o con modifiche indesiderate alle variabili di controllo delle iterazioni. Per controllare il flusso si userà la tecnica di call tree analysis_[g]. Il codice così verificato risulterà ben strutturato.

Analisi del flusso di dati: verificherà che in nessun cammino il codice acceda a variabili prive di significato (non inizializzate o non scritte prima della lettura) e che non ci siano anomalie come due scritture consecutive senza nessuna lettura. La non incapsulazione_[g] e l'aliasing_[g] complicano molto questa analisi, sarà quindi buona prassi creare sempre metodi getter/setter_[g].

Analisi del flusso di informazione: verificherà che le sole dipendenze tra input ed output dei moduli all'interno del codice siano quelle specificate in progettazione. Questa analisi può essere fatta su un singolo modulo, su più moduli o sull'intero sistema.

Analisi dinamica: prevede l'esecuzione del prodotto software, o di sue sottoparti, sotto certi parametri, come la ripetibilità (le prove effettuate sul codice devono essere ripetibili). Pertanto si deve avere conoscenza sullo stato dell'ambiente di esecuzione, sugli input, sull'esecuzione stessa e sull'analisi dei risultati. I risultati così ottenuti sono utili solamente quando vengono rilevati degli errori in uscita, ma l'assenza di errori non garantisce che il codice sia corretto (*"Il test di un programma può rilevare la presenza di malfunzionamenti, ma non dimostrarne l'assenza"* Tesi di Dijkstra (1969)).

Si verificheranno più moduli dipendenti tra loro utilizzando Stub_[g] e Driver_[g]. Per Stub si intende una componente passiva fittizia per simulare un modulo, mentre per Driver una componente attiva fittizia per pilotare un modulo. I test si dividono principalmente in due categorie:

- Test di verifica funzionale (Black-box): test del funzionamento di un'unità a scatola chiusa. Testa il modulo senza conoscere la struttura interna, basandosi solamente sui dati in entrata e confrontando i valori attesi con quelli ottenuti in uscita;

- Test di verifica strutturale (White-box): testa il codice facendo accesso diretto ai metodi del modulo in esame per controllarne il funzionamento e la correttezza logica.

Per entrambe le tipologie di verifica sarà redatto un documento interno riguardante i test effettuati. Tale documento deve indicare:

- modulo testato;
- funzionalità coinvolte nel test;
- dati in input;
- dati attesi in output;
- dati ottenuti in output.

Ogni test richiederà quindi di:

- identificare le funzionalità da testare;
- stabilire i dati in input;
- identificare le risorse necessarie per eseguire il test;
- stabilire la procedura con cui dev'essere eseguito;
- specificare i risultati di output attesi.

In seguito al fallimento di un test si presentano due scenari. Se sono state identificate delle anomalie queste dovranno essere comunicate con lo strumento delle Issue. Verranno quindi corrette e sarà rieseguito il test sul componente corretto.

3.3.3 Strategie per la verifica

La strategia per la verifica prevede le seguenti quattro fasi che saranno osservate più in dettaglio successivamente.

- verifica di accettabilità;
- verifica del disegno architetturale;
- verifica del codice;
- verifica di performance e criticità.

Verifica di accettabilità

Per l'accettazione il gruppo testerà che le funzionalità descritte nei requisiti siano non solamente coperte dai test di unità e di integrazione, ma anche da prove funzionali sul sistema. Queste prove sono state decise durante la fase di analisi dei requisiti.

Verifica del disegno architetturale

Una volta realizzato lo schema architetturale del prodotto verrà verificato il rispetto dei seguenti principi:

- Layering: le classi devono avere una struttura gerarchica;
- Packaging: le classi devono essere accorpate secondo un rigore logico;
- Massima coesione: ogni unità sarà progettata per rispondere solamente ai requisiti a lei associati nel tracciamento dei requisiti;
- Minimo accoppiamento: le classi dovranno essere progettate in maniera che siano il più indipendenti possibili.

Questa verifica verrà effettuata anche grazie all'ausilio di metriche. Purtroppo non è automatizzabile e deve essere eseguita manualmente. Richiede quindi una notevole quantità di tempo ricompensata però da una migliore estendibilità e manutenibilità del sistema. I diagrammi delle classi utilizzati in questa fase saranno poi una buona base di partenza per la codifica, in quanto strutturalmente corretti.

Verifica del codice

La verifica del codice ha come obiettivo:

- Trovare, segnalare e isolare le parti di codice che non rispettano le norme prestabilite nel documento *Norme-di-progetto-4.0.0.pdf* attraverso analisi statica ;
- Identificare quali sono le parti di codice soggette ad errori di programmazione mediante analisi dinamica.

Utilizzando questi strumenti è possibile scoprire lacune nel sistema che, una volta corrette, garantiranno una migliore flessibilità, manutenibilità e affidabilità.

Verifica di performance e criticità

Performance e criticità dipendono fortemente dalla piattaforma *Google App Engine*_[g] e dall'infrastruttura di rete. Il gruppo si impegnerà ad effettuare misurazioni riguardanti il carico di lavoro e la stabilità del sistema nel caso di picchi di richieste, in modo da verificare la robustezza e la reattività del sistema, pur sapendo la che non sarà possibile agire direttamente sulla piattaforma.

4 Metriche di misurazione

Sapendo che le metriche per la verifica saranno uno strumento importante per la qualità del codice e dell'architettura abbiamo deciso di adottare le seguenti:

- numero di classi per package;
- quantità e complessità delle interfacce;
- Afferent Coupling (per package) : indica il valore di fan-in;
- Efferent Coupling (per package) : indica il valore di fan-out;
- numero di parametri formali per metodo;
- complessità ciclomatica;

Per queste metriche sono state definite delle soglie di accettabilità che dovranno essere rispettate. Eventuali sforamenti dovranno essere giustificati e saranno consentiti solo se coerenti con le specifiche architetturali. Si può osservare come alcune metriche prese in osservazione influenzino direttamente la qualità di prodotto, come mostrato nello schema seguente:

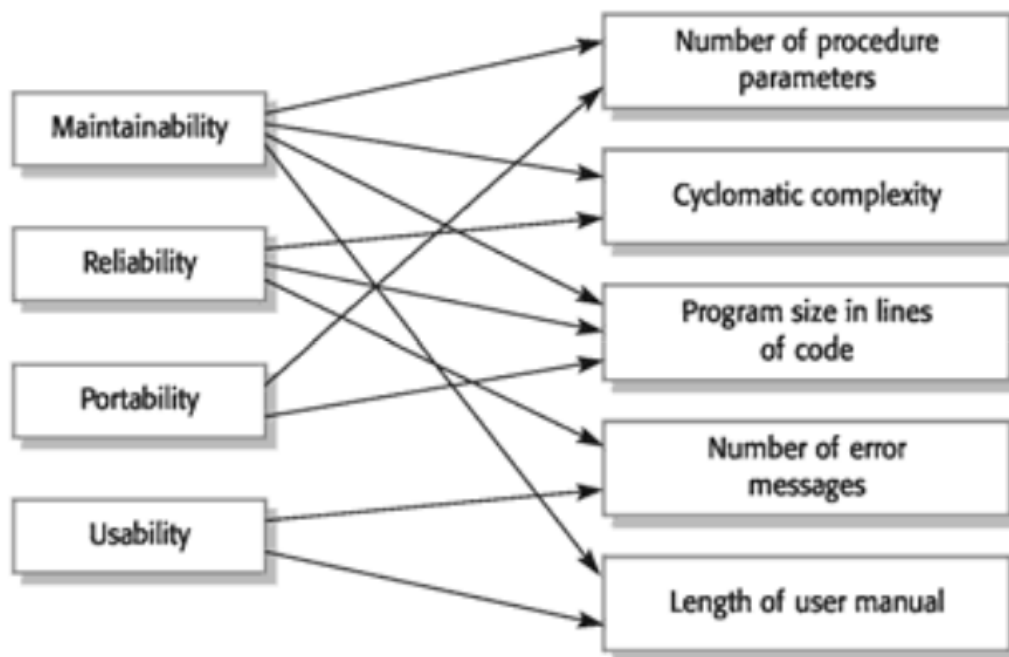


Figura 3: Relazione qualità - metriche

Fan-in e fan-out: Con fan-in si intende il numero di chiamate entranti in un modulo e rappresenta l'indice di utilità del modulo. Con fan-out si definisce invece il numero di chiamate uscenti fatte dal modulo verso altri moduli e rappresenta l'indice di dipendenza del modulo. Il modulo perfetto dovrebbe avere un fan-in infinito e un fan-out pari a zero, il che è banalmente impossibile. Il verificatore dovrà accertarsi che ogni modulo soddisfa questo vincolo: l'indice di fan-in sia maggiore o uguale all'indice di fan-out. In caso contrario dovrà notificare un'anomalia e il programmatore si adopererà per risolverla.

Numero di parametri formali per metodo: Questa metrica è il conteggio dei numeri di parametri di un metodo. molti metodi che presentano un alto numero di parametri possono essere semplificati raggruppando parametri in gruppi e creando classi che li racchiudano. Può succedere però che, così facendo, si ottengano delle nuove classi con comportamenti simili ad altre classi già esistenti.

Complessità ciclomatica: Per calcolare la complessità ciclomatica si sommano tutti i cammini linearmente indipendenti di un metodo, che può essere rappresentato con un grafo ad albero. Maggiore è il numero di cammini, maggiore è la complessità ciclomatica e quindi anche il rischio di commettere errori, almeno secondo lo studio di Rich Sharpe - McCabe Cyclomatic Complexity: the proof in the pudding. La complessità ciclomatica (**C**) è definita come:

$$C = E - N + P$$

dove:

- **E** è il numero di archi;
- **N** il numero di nodi;
- **P** le componenti connesse da ogni arco.

La complessità ciclomatica è utile non solo per determinare il numero di test da eseguire su un metodo allo scopo di coprire tutti i possibili cammini, ma anche per controllare l'adattabilità, la testabilità e l'amministrabilità del software. Può capitare che un alto valore di **C** non necessariamente sia indice di effettiva complessità del codice perché, per esempio, un'istruzione di switch viene trasformata in un grafo ad albero molto più complesso di quanto lo sia oggettivamente il codice. Viceversa, un basso valore di **C** potrebbe non rivelare un codice scritto in modo confuso, criptico e complicato. Per i valori massimi di complessità ciclomatica il programmatore e il verificatore dovranno attenersi alle limitazioni imposte nel documento *Norme-di-progetto-4.0.0.pdf*.

L'utilizzo di queste metriche aiuterà il gruppo nella garanzia della qualità di prodotto. Lo standard ISO/IEC 15939:2007 definisce le seguenti direttive per effettuare misurazioni software:

Fase di pianificazione:

- fissare gli obiettivi di misurazione;
- specificare misure utili;
- specificare come collezionare le misure ottenute;
- specificare come analizzare le misure.

Fase di raccolta e analisi:

- comunicare i risultati ottenuti;
- memorizzare i risultati in condivisione per renderli disponibili velocemente;
- analizzare le misure confrontandole con i limiti prefissati;
- collezionare i risultati finali ottenuti dalle precedenti attività di misurazione.

5 Gestione amministrativa della revisione

5.1 Comunicazione e risoluzione anomalie

Con anomalia si intende un'incongruenza del codice rispetto all'architettura o alle norme, il che produrrebbe una deviazione inaccettabile del programma, rispetto a funzionalità o qualità specificate. Una volta che il verificatore avrà trovato una anomalia, la segnalerà indicandone, dopo una veloce analisi, una stima delle risorse necessarie per la correzione. Il programmatore incaricato della risoluzione darà una possibile soluzione con informazioni riguardanti l'anomalia e la sua risoluzione. Alla fine di ogni ciclo di vita i verificatori redigeranno un documento contenente i test eseguiti e i risultati ottenuti al fine di permettere al responsabile di verificare la copertura totale del progetto. Per avvisare della presenza di eventuali anomalie useremo lo strumento delle Issue messo a disposizione da Google Project Hosting. Ogni Issue sarà accompagnata da titolo, descrizione del problema, possibili cause, referente e priorità. Sarà possibile inoltre inserire parti di codice o dell'output che siano di aiuto alla comprensione.

5.2 Procedure di controllo di qualità di processo

5.2.1 Software Quality Assurance

Per ciò che concerne la Software Quality Assurance_[g] ci si riferisce al documento Norme-di-progetto-4.0.0.pdf al paragrafo 8.6.

6 Dettaglio dell'esito delle revisioni

Dopo ogni revisione, una volta a conoscenza della valutazione della documentazione e della presentazione, il gruppo procederà a modificare, come richiesto, le parti che presentino problemi per poter proseguire nello sviluppo con una base corretta e verificata.

6.1 Revisione dei requisiti

Alla revisione dei requisiti la documentazione prodotta e il lavoro svolto sono stati valutati modesti. Dalla valutazione è emerso un problema rilevante sul piano di progetto per quanto riguarda il preventivo. Di seguito sono riportate le principali modifiche ai prodotti presentati per la revisione.

- **Analisi dei requisiti:** sono stati totalmente rivisitati alcuni Use Case che risultavano errati. È stato introdotto uno Use Case che definisce in modo chiaro i vari attori ed il loro ruolo nel sistema. Inoltre la gerarchia degli Use Case è stata riformulata seguendo un ordine preciso. È stato aggiunto un capitolo riguardante la convalida dei requisiti;
- **Piano di progetto:** è stato corretto il preventivo considerandolo solamente dopo la consegna dei documenti della revisione con conseguente modifica del diagramma di Gantt e modificate le attività di progetto. È stata inserita l'appendice riguardante il consuntivo dell'investimento prima della consegna dei documenti e infine è stata aggiornata la parte di gestione dei rischi;
- **Piano di qualifica:** sono state corrette le incongruenze terminologiche presenti, corretta la parte riguardante la SQA. Nella campagna di validazione sono stati definiti i metodi di validazione, i test dei requisiti e i test di accettazione del prodotto.

Maggiori informazioni a riguardo sono presenti nel diario delle modifiche di ogni documento.

6.2 Revisione di progetto

- **Norme di progetto:** è stato inserito e rivisto il capitolo riguardante la software quality assurance precedentemente inserito nel Piano di qualifica.
- **Analisi dei requisiti:** sono stati riviste le precondizioni degli Use Case.
- **Specifiche tecniche:** specificata la versione di UML utilizzata e adattati i diagrammi allo standard. Contestualizzati i pattern descritti ed evidenziate le dipendenze tra i package. Modificati i nomi dei package. Aggiunta descrizione del

package `clientandroid` e delle classi non descritte in precedenza. Spostato il tracciamento.

- Piano di progetto: resa in forma tabellare la descrizione delle tempistiche delle attività. Aggiornato il diagramma di Gantt e la tabella della rotazione dei ruoli.
- Piano di qualifica: spostati i capitoli riguardanti la pianificazione e gli esiti delle campagne di verifica.

Maggiori informazioni a riguardo sono presenti nel diario delle modifiche di ogni documento.

6.3 Revisione di qualifica

- Norme di progetto: aggiunta descrizione dello script e del processo di “commit”.
- Specifica tecnica: aggiunte implementazioni interfacce actions, corretta condizione sul diagramma e corrette figure.
- Definizione di prodotto: aggiunto diagramma di sequenza per la creazione di viste e presenter. Corrette le immagini e aggiunto un esempio di mapping di un converter. Aggiunte e aggiornate classi e immagini.
- Piano di progetto: corretto il termine Gantt, corretto il preventivo a finire in ingresso alla RQ. Aggiunto il consuntivo in ingresso alla RA.
- Piano di qualifica: inserite tabelle sui test. Inseriti risultati di Ant e aggiornate copertura e metriche. Aggiornate inoltre i risultati del β -testing e dei test di accettabilità.

Maggiori informazioni a riguardo sono presenti nel diario delle modifiche di ogni documento.

A Pianificazione ed esecuzione collaudo

Nel seguente capitolo vengono illustrati i test definiti dopo ogni fase di progetto. Questi si dividono in test di accettabilità, integrazione ed unità. Una volta cominciata la campagna di validazione verrà aggiunta la sezione riguardante gli esiti della stessa.

A.1 Specifica della campagna di validazione

A.1.1 Collaudo

Per il collaudo sono previste due modalità di conduzione. La prima definita α -test ed è un collaudo interno. Consisterà nella verifica finale del tracciamento dei requisiti attraverso le varie fasi dello sviluppo. Sarà anche effettuata una prova di esecuzione. Quest'ultima non è considerata come una tecnica per cercare errori o evidenziare difetti, ma serve solamente a dare evidenza che il prodotto funzioni correttamente.

La seconda modalità è definita β -test ed è un collaudo esterno. I test saranno affidati a persone esterne allo sviluppo del sistema, che ne testeranno le funzionalità, l'accessibilità e la fruizione.

A.1.2 Test di accettabilità

Di seguito sono riportati i test di accettabilità pianificati in seguito all'analisi dei requisiti. Questi test verificheranno che il sistema sviluppato soddisfi i requisiti funzionali emersi dallo studio del capitolato e degli use case.

TA1 Test di immissione di una richiesta di aiuto

Verifica che si possa effettivamente immettere una richiesta di aiuto nel sistema e che tale operazione possa essere effettuata nel modo giusto. Si verifica che si possano inserire la catastrofe associata, il titolo, la descrizione, la gravità ed altri eventuali campi o allegati. Il sistema non deve accettare l'immissione di dati incoerenti (non compatibili con il tipo di campo). Si accerta che i dati della richiesta siano effettivamente memorizzati nel database, con l'aggiunta delle coordinate geografiche, della data e delle-mail del segnalante (solo se la richiesta in questione non è anonima). I dati memorizzati devono essere corretti, ossia devono rispecchiare quelli inseriti dall'utente oppure, nel caso di quelli inseriti automaticamente, la situazione reale. Per visualizzare questi dati si accede al sistema come operatore e si seleziona la richiesta in questione.

TA1.1 E-mail di conferma

Verifica che il segnalante riceva una e-mail di conferma dopo l'immissione di una richiesta di aiuto. Si immette una segnalazione e si attende un tempo ragionevole per

ricevere l'e-mail.

TA1.2 Visualizzazione richieste del segnalante

Verifica che il segnalante possa visualizzare le proprie segnalazioni. Egli deve poter visualizzare tutte le segnalazioni con i relativi stati di avanzamento e i commenti degli operatori. Si accerta che il segnalante possa aggiungere propri commenti ad ogni richiesta e che tali siano memorizzati in modo corretto nel database.

TA2 Test di elaborazione delle segnalazioni

Verifica che gli operatori possano effettivamente elaborare le richieste di aiuto. Si accerta che l'operatore possa modificare lo stato di avanzamento ed immettere commenti. Si verifica che il database venga aggiornato correttamente. Per fare questo si aggiorna la pagina dei dati della richiesta e si cerca un riscontro nei nuovi dati.

TA3 Test di gestione degli operatori

Verifica che l'amministratore di autorità possa effettivamente gestire gli operatori. Si controlla che si possano creare nuovi operatori immettendone i dati personali, oppure che si possano elaborare quelli già esistenti. In particolare si devono poter assegnare gli operatori a gruppi ed a catastrofi. Si può inoltre gestire il loro privilegio a prendere in carico le nuove richieste. Per verificare che i cambiamenti siano effettuati correttamente si devono ricontrollare i dati dell'operatore in questione.

TA4 Test di gestione delle autorità

Verifica che l'amministratore di sistema possa gestire le autorità. Si controlla che si possano creare nuove autorità inserendone i dati e associandone un amministratore di autorità. Per verificare che i cambiamenti siano effettuati correttamente si devono ricontrollare i dati delle autorità sempre dalla pagina dedicata all'amministratore di sistema.

TA5 Test di registrazione

Verifica che l'utente possa effettivamente registrarsi nel sistema. Si controlla che tutti i dati necessari possano essere inseriti e che non vengano accettati dati non validi. Si accerta che non si possa effettuare la registrazione senza aver inserito l'e-mail, la password ed altri campi obbligatori. Una volta terminata la registrazione si prova ad autenticarsi al sistema con le credenziali appena stabilite. Si appura che l'operazione vada a buon fine e che venga visualizzata la pagina del segnalante. Si accede alla sezione di gestione dei dati personali e si controlla che questi siano stati memorizzati correttamente nel database.

TA6 Test di autenticazione

Verifica che l'utente si possa autenticare nel sistema. Si inseriscono le credenziali di un segnalante, un operatore o un amministratore di autorità e si controlla che il sistema visualizzi la relativa pagina. Si verifica che inserendo una coppia e-mail password non presente nel database il sistema non esegua l'autenticazione, ma che visualizzi un messaggio di errore.

A.1.3 Test di integrazione delle componenti principali

Di seguito saranno definiti i test di integrazione sui moduli funzionali del sistema, ovvero le classi che implementano dei metodi. Interfacce e classi dati non saranno pertanto singolarmente trattate in questi paragrafi, poiché le prime non definiscono metodi, ma solo contratti da implementare, mentre le seconde sono già state trattate dai test di unità.

TI1 - Test moduli `server.model.businesslogic` - `server.model.dao`

Una volta testate le funzionalità interne del `dao` e `businesslogic` a livello di unità, e di integrazione interna tra di loro (come fare questi test sarà definito in seguito al rilascio dell'architettura di dettaglio), si procederà ad integrare questi due moduli tra di loro e ad effettuare test di integrazione con le seguenti modalità:

- Saranno richiamati in modo diretto metodi delle classi contenute nella `businesslogic` (dunque isolando questo modulo dai suoi utilizzatori tramite driver) che vanno ad interagire con il database con parametri in ingresso prestabiliti.
- Se i metodi in questione andranno ad accedere al database in lettura, si calcoleranno a priori i valori in uscita attesi e se ne verificherà in modo automatizzato la correttezza.
- Se i metodi in questione andranno ad accedere al database in scrittura, si verificherà tramite le funzioni interne a Google App Engine la correttezza dei valori memorizzati.
- Nel caso di inconsistenze tra dati ottenuti ed attesi, si procederà all'individuazione ed alla modifica della componente che causa l'errore ed alla riesecuzione dei test di unità e di integrazione interni al modulo modificato. Successivamente si riprenderà dal punto 1.

TI2 - Test moduli `server.controller` – TI1

Partendo dal successo del test precedente ed una volta testate le funzionalità interne di `controller` e `businesslogic` a livello di unità, e di integrazione interna tra di loro (come fare questi test sarà definito in seguito al rilascio dell'architettura di dettaglio),

si procederà ad integrare il modulo `server.controller` e ad effettuare test di integrazione con le seguenti modalità:

- Separatamente per ogni tipo di controller implementato (`UserController`, `SignalerController`, ...) si andranno a richiamare singolarmente tutte le funzioni che vanno ad interagire con la `businesslogic` (isolando il modulo `controller` dai suoi utilizzatori tramite driver) e quindi col database (integrazione tra i due precedentemente testata).
- Nel caso di metodi che andranno a modificare lo stato del database, dati parametri in ingresso e risultati attesi stabiliti a priori, si verificherà tramite funzioni interne a Google App Engine la correttezza delle modifiche effettuate.
- Nel caso di metodi che vadano a recuperare dati dal database e ad elaborarli, dati parametri in ingresso e risultati attesi stabiliti a priori, si verificherà in modo automatizzato che il risultato dell'elaborazione ottenuto sia corretto.
- Nel caso di inconsistenze tra dati ottenuti ed attesi, si procederà all'individuazione ed alla modifica della componente che causa l'errore ed alla riesecuzione dei test di unità e di integrazione interni al modulo modificato. Successivamente si riprenderà dal punto 1.

TI3 - Test moduli `clientweb.presenter` – TI2

Una volta testate le funzionalità interne di `server.controller` e `clientweb.presenter` a livello di unità, e di integrazione interna tra di loro (come fare questi test sarà definito in seguito al rilascio dell'architettura di dettaglio), si procederà ad integrare questi due moduli tra di loro e ad effettuare test di integrazione con le seguenti modalità:

- Separatamente per ogni tipo di presenter, si andranno ad invocare le varie funzionalità che vanno ad interagire con il controller (isolando il modulo `presenter` dai suoi utilizzatori tramite driver).
- Nel caso di metodi che comportino modifica dello stato della base di dati (integrazione tra le componenti che consentono queste modifiche precedentemente testate), dati parametri in input definiti a priori, si andranno a verificare i risultati ottenuti tramite funzionalità interne a Google App Engine.
- Nel caso di metodi che non modifichino lo stato del database, ma si limitino a recuperare dati e/o effettuare operazioni su di essi, dati parametri in input e risultati attesi definiti a priori, si andrà a verificare la correttezza degli output in modo automatizzato.
- Nel caso di inconsistenze tra dati ottenuti ed attesi, si procederà all'individuazione ed alla modifica della componente che causa l'errore ed alla riesecuzione dei

test di unità e di integrazione interni al modulo modificato. Successivamente si riprenderà dal punto 1.

TI4 - Test moduli `clientweb.gui` – TI3

Una volta testate le funzionalità interne di `clientweb.gui` e `clientweb.presenter` a livello di unità, e di integrazione interna tra di loro (come fare questi test sarà definito in seguito al rilascio dell'architettura di dettaglio), si procederà ad integrare questi due moduli tra di loro e ad effettuare test di integrazione con le seguenti modalità:

- separatamente per ogni tipo di vista definita nel sistema, si procederà all'invocazione dei metodi in essa definiti tramite l'utilizzo di driver, isolando quindi gli input (in questo caso fittizi) dai rispettivi handlers;
- nel caso il metodo richiamato comporti una modifica dello stato della base di dati (integrazione tra le componenti che consentono queste modifiche precedentemente testate), dati parametri in input e risultati attesi definiti a priori, si andranno a verificare i risultati ottenuti tramite funzionalità interne a Google App Engine;
- nel caso di metodi che non modifichino lo stato del database, dati parametri in input e risultati attesi definiti a priori, si andrà a verificare la correttezza degli output in modo automatizzato;
- Nel caso di inconsistenze tra dati ottenuti ed attesi, si procederà all'individuazione ed alla modifica della componente che causa l'errore ed alla riesecuzione dei test di unità e di integrazione interni al modulo modificato. Successivamente si riprenderà dal punto 1.

A.1.4 Test di integrazione delle sotto-componenti

Di seguito saranno definiti i test di integrazione sulle sotto-componenti del sistema, classi e interfacce che implementano le funzionalità. Interfacce e classi dati non sono trattate singolarmente in questi paragrafi, ma in maniera coesa rispettando l'ordine logico delle funzionalità.

TIS1 Test Dao - Transfer Object

Testati i Dao e i Transferobjects a livello di unità, si procede con l'integrazione di questi. I test di integrazione sono eseguiti nelle seguenti modalità:

- per ogni tipo di Dao definito nel sistema, si procede all'invocazione dei metodi in esso definiti. L'utilizzo di driver in questo processo permette di isolare il Dao dal package `server.model.services`;

- definiti a priori parametri in input e risultati attesi in output, se il metodo richiamato modifica lo stato del database, si verificano i risultati ottenuti tramite funzionalità interne a Google App Engine;
- definiti a priori parametri in input e risultati attesi in output, se il metodo richiamato non modifica lo stato del database, si verificano i risultati ottenuti in modo automatizzato;
- se i dati ottenuti risultano inconsistenti rispetto ai dati attesi, si procede all'individuazione e alla modifica della componente che causa l'errore. Si rieseguo nuovamente i test di unità interni al modulo modificato. Si riprende infine dal primo punto.

TIS2 Transfer Object - Converter Testati i Converter e i Transferobject a livello di unità, si procede con l'integrazione di questi. I test di integrazione sono eseguiti nelle seguenti modalità:

- per ogni tipo di Converter definito nel sistema, si procede all'invocazione dei metodi in esso definiti. L'utilizzo di driver in questo processo permette di isolare il Converter dal package `server.model.dao`;
- definiti a priori dei parametri in input e dei risultati attesi in output, si verifica la correttezza della conversione in modo automatizzato;
- se i dati ottenuti risultano inconsistenti rispetto ai dati attesi, si procede all'individuazione e alla modifica della componente che causa l'errore. Si rieseguo nuovamente i test di unità interni al modulo modificato. Si riprende infine dal primo punto.

A.1.5 Test di unità

I test di unità sono programmati per tutte le classi del progetto che si occupano di soddisfare requisiti funzionali. Sono però escluse dai test le classi dei package che si occupano di gestire la parte grafica del sistema. I test che saranno effettuati sul codice sorgente saranno eseguiti tramite l'utilizzo di strumenti quali JUnit, EasyMock, e utilità di verifica di Spring, utilizzando particolarmente la classe `ReflectionTestUtils`. Affinché i test di unità siano utili e permettano di aumentare la qualità del prodotto, devono soddisfare alcune proprietà di seguito espote:

- Automatizzazione: devono poter essere eseguiti automaticamente. In ogni progetto deve essere disponibile un servizio di automazione che permetta agli sviluppatori di eseguire tutti o una parte dei test di unità in modo semplice. Tale proprietà viene rispettata in quanto gli strumenti utilizzati sono integrabili sia nell'ambiente di sviluppo che nel sistema di building automatico. Ad ogni sviluppatore

sarà quindi possibile eseguire in modo autonomo i test di suo interesse e, sistematicamente, verranno eseguiti tutti e notificati in modo adeguato al momento del building automatico.

- **Approfondimento** : devono essere esaustivi e accurati, devono verificare il comportamento di qualsiasi parte del progetto che potrebbe creare degli errori. Tale proprietà viene verificata tramite l'utilizzo di Cobertura il quale è in grado di calcolare la copertura dei test di unità cioè il numero di righe e possibili flussi di controllo che vengono effettivamente testati.
- **Indipendenza**: devono essere il più possibile indipendenti dall'ambiente di esecuzione dalle altre classi del progetto. Quando si scrive un test è consigliato verificare il comportamento di un singolo aspetto del progetto, in questo modo si riesce ad identificare univocamente un'errore. La proprietà di indipendenza è ottenuta mediante l'utilizzo della tecnica dei Mock Objects[g] nello specifico tramite la libreria EasyMock. Tale libreria permette di scrivere in modo facile degli oggetti, implementazioni di interfacce o istanziazioni di classi, ai quali è possibile assegnare un comportamento predefinito e indipendente dallo stato del sistema. Con l'uso di questa tecnica viene isolato il singolo metodo (o insieme di metodi) che si vuole testare dando un comportamento predefinito agli oggetti esterni con il quale interagisce. Quindi, se un test dovesse fallire, si è certi che l'errore risiede nel metodo che si sta testando e non in una componente di cui fa uso, avendo un comportamento noto.
- **Ininfluenza**: i test non devono influenzare il codice sorgente, pertanto non è accettabile modificare il codice sorgente per consentire o facilitare il testing (ad esempio aggiungere metodi getter e setter). A questo proposito è stata utilizzata la classe del framework Spring `ReflectionTestUtils`, che consente di impostare campi dati di una classe, anche privati, con una semplice chiamata di funzione.

A.2 Configurazione dell'ambiente di build

L'ambiente di build è stato automatizzato con un file di configurazione di Ant per la generazione di rapporti dettagliati nell'ambiente di build, che non coincide con l'ambiente di sviluppo. Dati i lunghi tempi di esecuzione, sia del processo di build, sia dell'esecuzione dei test e generazione dei relativi rapporti sull'esito del processo, si è ritenuto necessario separare in due flussi l'esecuzione della build. Ciò è possibile poiché la compilazione GWT, richiesta per la corretta esecuzione del software, non è necessaria per l'esecuzione dei test. Di seguito vengono riportati e spiegati alcuni estratti del file di configurazione "build.xml".

La prima sezione riguarda la configurazione e suddivisione delle directory e dei diversi classpaths necessari per le diverse esecuzioni della build. Sono stati definiti tre classpath: il primo, per la sola esecuzione della build del prodotto, comprende i file sorgenti del prodotto e le librerie esterne necessarie; il secondo riferisce il primo ed aggiunge le classi di test; il terzo riferisce al secondo e aggiunge le librerie necessarie alla generazione dei rapporti di Cobertura.

```
<property name="gwt.args" value="" />

<!-- Configure paths -->
<property name="src.dir" location="src" />
<property name="test.dir" location="test" />
<property name="lib.dir" location="war/WEB-INF/lib" />
<property name="build.dir" location="war/WEB-INF/classes" />
<property name="build.test.dir" location="test-classes" />
<property name="test.resources.dir" location="${test.dir}/resources" />
<property name="reports.dir" location="reports" />
<property name="cobertura.lib.dir" value="cobertura-lib" />
<property name="cobertura.report.dir"
    value="${reports.dir}/cobertura-results" />
<property name="instrumented.dir"
    value="${build.dir}/cobertura-instrument" />

<path id="project.classpath">
    <pathelement location="${build.dir}" />
    <fileset dir="${lib.dir}" >
        <include name="*.jar" />
    </fileset>
</path>

<path id="test.classpath">
    <path refid="project.classpath" />
    <pathelement location="${build.test.dir}" />
</path>

<path id="cobertura.classpath">
    <path refid="test.classpath" />
    <fileset dir="${cobertura.lib.dir}">
        <include name="**/*.jar" />
    </fileset>
</path>
<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />
```

Il seguente estratto compila i file sorgenti sia del prodotto sia dei test e li posiziona in cartelle definite.

```
<target name="prepare" description="Creazione directory di build">
    <mkdir dir="${build.dir}" />
    <mkdir dir="${build.test.dir}" />
</target>

<target name="javac" depends="prepare"
    description="Compile java source to bytecode">
    <javac srcdir="${src.dir}" includes="*" encoding="utf-8"
        destdir="${build.dir}"
        debug="true" debuglevel="lines,vars,source">
        <classpath refid="project.classpath" />
    </javac>
    <copy todir="${build.dir}">
        <fileset dir="src" excludes="**/*.java,**/jdoconfig.xml" />
    </copy>
</target>

<target name="javac-test" depends="javac"
    description="Compile test java source to bytecode">
    <javac srcdir="${test.dir}" includes="*" encoding="utf-8"
        destdir="${build.test.dir}" />
```

```
        debug="true" debuglevel="lines,vars,source">
        <classpath refid="test.classpath"/>
    </javac>
    <copy todir="${build.test.dir}">
        <fileset dir="test" excludes="**/*.java"/>
    </copy>
</target>
```

Il seguente estratto, segue la compilazione dei sorgenti, esegue la compilazione GWT, che compila in javascript le classi scritte in java.

```
<target name="gwtc" depends="javac"
        description="GWT compile to JavaScript (production mode)">
    <java failonerror="true" fork="true"
        classname="com.google.gwt.dev.Compiler">
        <classpath>
            <pathelement location="${src.dir}"/>
            <path refid="project.classpath"/>
        </classpath>
        <!-- add jvmarg -Xss16M or similar if you see a StackOverflowError -->
        <jvmarg value="-Xmx256M"/>
        <arg line="-war"/>
        <arg value="war"/>
        <!-- Additional arguments like -style PRETTY or -logLevel DEBUG -->
        <arg line="${gwt.args}"/>
        <arg value="it.donkeycode.rescueme.RescueMe"/>
    </java>
</target>
```

A seguire vi è la copia delle risorse esterne, necessarie per l'esecuzione dei test, e l'esecuzione dei test vera e propria.

```
<target name="copy_test_resources">
    <copy todir="${build.test.dir}" >
        <fileset dir="${test.resources.dir}" >
            <include name="**/*.*" />
            <exclude name="**/*.java" />
        </fileset>
    </copy>
</target>

<target name="test.dev" depends="copy_test_resources, javac-test"
        description="Run development mode tests">
    <mkdir dir="reports/htmlunit.dev" />
    <junit fork="yes" printsummary="yes" haltonfailure="yes">
        <jvmarg line="-Xmx256m" />
        <sysproperty key="${gwt.args}" value="-standardsMode -logLevel WARN" />
        <sysproperty key="java.awt.headless" value="true" />

        <classpath location="${instrumented.dir}" />
        <classpath refid="cobertura.classpath" />

        <batchtest todir="reports/htmlunit.dev" >
            <fileset dir="${build.test.dir}" >
                <include name="**/*Test.class" />
            </fileset>
        </batchtest>
        <formatter type="plain" />
        <formatter type="xml" />
    </junit>
</target>

<target name="test-execute"
        description="Run development tests">
    <antcall target="test.dev" />
</target>
```

Infine vi sono le istruzioni per generare i rapporti di Cobertura in formato html.

```
<target name="instrument" depends="javac">
    <delete file="cobertura.ser" />
    <delete dir="${instrumented.dir}" />
    <cobertura-instrument todir="${instrumented.dir}">
        <fileset dir="${build.dir}">
            <include name="**/*.class" />
            <exclude name="**/*Test.class" />
            <exclude name="**/clientweb/**" />
        </fileset>
    </cobertura-instrument>
</target>

<target name="coverage-check">
    <cobertura-check branchrate="34" totallinerate="80" />
</target>

<target name="coverage-report">
    <delete dir="${cobertura.report.dir}" />
    <mkdir dir="${cobertura.report.dir}" />
    <cobertura-report destdir="${cobertura.report.dir}">
        <fileset dir="${src.dir}">
            <include name="**/*.java" />
        </fileset>
        <fileset dir="${test.dir}">
            <include name="**/*.java" />
        </fileset>
    </cobertura-report>
</target>

<target name="coverage" depends="instrument, test-execute, coverage-report"
    description="instrument our self,
        run the tests and generate JUnit and coverage reports." />
```

B Resoconto delle attività di verifica

Di seguito sono riportate le tabelle di tracciamento requisito-componente e componente-requisito, le quali sono precedute da due tabelle che specificano le sigle dei componenti. Attraverso lo strumento ReTracker sarà inoltre possibile tracciare i test sulle componenti del sistema e quindi per ogni requisito i test che lo verificheranno.

B.1 Tracciamento

B.1.1 Packages naming

Nome package	Sigla
rescueme	RM
rescueme.clientweb	CW
rescueme.clientweb.gui	CW.G
rescueme.clientweb.gui.widget	CW.G.W
rescueme.clientweb.gui.widget.utils	CW.G.W.U
rescueme.clientweb.gui.widget.menu	CW.G.W.M
rescueme.clientweb.gui.widget.bodies	CW.G.W.B
rescueme.clientweb.gui.widget.bodies.authority	CW.G.W.B.A
rescueme.clientweb.gui.widget.bodies.population	CW.G.W.B.P
rescueme.clientweb.gui.widget.bodies.systemadmin	CW.G.W.B.S
rescueme.clientweb.presenter	CW.P
rescueme.clientandroid	CA
rescueme.clientandroid.gui	CA.G
rescueme.clientandroid.operations	CA.P
rescueme.clientandroid.operations	CA.I
rescueme.shared	SH
rescueme.shared.gwt	SH.G
rescueme.shared.businessobjects	SH.B
rescueme.shared.exceptions	SH.E
rescueme.server	SR
rescueme.server.controller	SR.C
rescueme.server.model	SR.M
rescueme.server.model.services	SR.M.S
rescueme.server.model.dao	SR.M.D
rescueme.server.model.dao.converter	SR.M.D.C
rescueme.server.model.dao.transferobjects	SR.M.D.T

B.1.2 Componenti - requisiti

Componente	Requisito
SH.O.GenericUserBO	FO4.1
SH.O.SignalerBO	FO3 FO4.2
SH.O.OperatorBO	FO6.1 FO6.9 FO8.1 FD3
SH.O.OperatorGroupBO	5.6.1 FO6.9
SH.O.AuthorityBO	FO8 FO8.2 FD3
SH.O.CatastropheBO	FO1.2 FO5.1.1 FO5.1.2 FO5.1.3 FO5.8 FD2 FD3
SH.O.CatastropheManagerBO	FO5.6
SH.O.ReportingBO	FO1 FO1.1 FO1.2 FO1.4 FO1.5 FO1.6 FO1.7 FO2.1 FO6.5 FO6.6 FO6.7 FO6.8 FO6.9 FO6.10 FO7 FD2 FD3

Componente	Requisito
SH.O.Priority	FO6.6
SH.O.Gravity	FO6.8
SH.O.Status	FO1.7
SH.O.CommentBO	FO2.1
	FO6.5
SH.O.HistoryBO	FO7
SR.M.D.SignalerDao	FO3
	FO4.2
	FO6.1
SR.M.D.OperatorDao	FO6.9
	FO8.1
	FD3
SR.M.D.OperatorGroupDao	FO5.6.1
	FO6.9
	FO8
SR.M.D.AuthorityDao	FO8.2
	FD3
	FO1.2
	FO5.1.1
	FO5.1.2
SR.M.D.CatastropheDao	FO5.1.3
	FO5.8
	FD2
	FD3
SR.M.D.CatastropheManagerDao	FO5.6

Componente	Requisito
SR.M.D.ReportingDao	FO1
	FO1.1
	FO1.2
	FO1.4
	FO1.5
	FO1.6
	FO1.7
	FO2.1
	FO6.5
	FO6.6
	FO6.7
	FO6.8
	FO6.9
	FO6.10
SR.M.D.CommentDao	FD2
	FD3
SR.M.D.HistoryDao	FO2.1
	FO6.5
SR.M.S.GenericUserActions	FO7
	FO3
SR.M.S.SignalerActions	FO4
	FO1
	FO1.3
	FO1.4
	FO1.5
	FO1.6
	FO1.7
	FO1.8
	FO4.2
	FO4.3

Componente	Requisito
SR.M.S.OperatorActions	FO1.7
	FO2
	FO6
	FO6.1
	FO6.7
	FO6.6
	FO6.8
	FO6.9
	FO6.9.1
	FO6.9.2
	FO6.10
	FO6.11
	FO7
	FD4
SR.M.S.AdminActions	FO5
	FO5.1
	FO5.1.1
	FO5.2
	FO5.3
	FO5.4
	FO5.5
	FO5.6
	FO5.6.1
	FO5.8
	FO8.1
	FD2
SR.M.S.SystemAdminActions	FO8
	FO8.2
	FD3
SR.M.S.CommonActions	FO2.1
	FO6.5
SR.C.UserController	FO3
	FO4

Componente	Requisito
SR.C.SignalerController	FO1
	FO1.3
	FO1.4
	FO1.5
	FO1.6
	FO1.7
	FO1.8
	FO4.2
	FO4.3
SR.C.OperatorController	FO1.7
	FO2
	FO6
	FO6.1
	FO6.7
	FO6.6
	FO6.8
	FO6.9
	FO6.9.1
	FO6.9.2
	FO6.10
	FO6.11
	FO7
	FD4
SR.C.AdminController	FO5
	FO5.1
	FO5.1.1
	FO5.2
	FO5.3
	FO5.4
	FO5.5
	FO5.6
	FO5.6.1
	FO5.8
	FO8.1
	FD2
SR.C.SystemAdminController	FO8
	FO8.2
	FD3

Componente	Requisito
SR.C.CommonController	FO2.1
	FO6.5
CW.P.UserPresenter	FO3
	FO4
CW.P.SignalerPresenter	FO1
	FO1.3
	FO1.4
	FO1.5
	FO1.6
	FO1.7
	FO1.8
	FO2.1
	FO4.2
	FO4.3
	FO1.7
	FO2
CW.P.OperatorPresenter	FO6
	FO6.1
	FO6.5
	FO6.6
	FO6.7
	FO6.8
	FO6.9
	FO6.9.1
	FO6.9.2
	FO6.10
	FO6.11
	FO7
	FD4

Componente	Requisito
CW.P.AdminPresenter	FO5
	FO5.1
	FO5.1.1
	FO5.2
	FO5.3
	FO5.4
	FO5.5
	FO5.6
	FO5.6.1
	FO5.8
	FO8.1
	FD2
CW.P.SystemAdminPresenter	FO8
	FO8.2
	FD3
CW.P.Initializer	FO4.1
CW.G.UserView	FO1
	IO1
	IO1.1
CW.G.SignalerView	FO1
	FO4.1
	IO1
	IO1.1
CW.G.OperatorView	FO2
	FO4.1
	FO6
	IO3
CW.G.AdminView	FO4.1
	FO5
	FO5.1
	IO3
CW.G.SystemAdminView	FO4.1
CW.G.W.P.ReportingForm	FO1
	FO1.1
	FO1.2
	FO1.6
	FO1.8

Componente	Requisito
	FO1
CW.G.W.P.UserReportingForm	FO1.1
	FO1.6
	FO1.8
CW.G.W.P.SignalerReportingForm	FO1
	FO1.1
	FO1.6
	FO3
CW.G.W.P.RegisterForm	FO3.1
	FO3.2
	FO4.2
CW.G.W.P.SignalerReportingDetails	FO2.1
CW.G.W.P.ChangeDataBody	FO4.3
CW.G.W.A.AuthorityReportingListBody	FO2
	FO6
	FO6.4
	FO1.7
	FO2
	FO6
	FO6.5
CW.G.W.A.AuthorityReportingDetailsBody	FO6.6
	FO6.7
	FO6.8
	FO6.9
	FO6.10
CW.G.W.A.OperatorDetails	FO5.5
CW.G.W.A.OperatorGroupDetails	FO5.5
	FO5.6.1
CW.G.W.A.OperatorCreation	FO5.3
CW.G.W.A.OperatorGroupCreation	FO5.4
CW.G.W.A.CatastropheDetails	FO5.2
	FO5.8
	FO5.1
CW.G.W.A.CatastropheCreation	FO5.1.2
	FO5.1.3
CW.G.W.A.CsvDownloaderBody	FD4
CW.G.W.A.AdminStatisticsWdg	FD2
CW.G.W.S.AuthorityCreationForm	FO8

Componente	Requisito
CW.G.W.S.AuthorityAdminCreationForm	FO8.1
CW.G.W.S.AuthorityDisableForm	FO8.2
CW.G.W.S.SystemAdminStatisticsWdg	FD3
CW.G.W.U.CustomMap	FO1.2.1
	FO1.6
	FO5.1.3
	FO6.10
CW.G.W.U.CatastropheListBox	FO1.2.1
CW.G.W.U.Login	FO4
CW.G.W.U.PieChart	FD2
	FD3
CW.G.W.U.BarChart	FD2
	FD3
CW.G.W.U.Login	FO4
CA.G.MainView	FO4.1
CA.G.LocalizationView	FO1.2
	FO1.2.1
	FO1.6
CA.G.GenericInfoView	FO1.1
CA.G.GravityView	FO1.1
CA.G.ReportingForwarding	FO1
	FO1.5
CA.G.ReportingListView	FO1.7
	FO2
CA.G.ReportingDetails	FO1.7
	FO2.1
CA.G.PreferencesView	FO1.8
	FO4
CA.G.RegistrationView	FO3
	FO3.1
	FO3.2
	FO4.2
CA.O.CatastropheOperations	FO1.2
CA.O.SubmitReportingOperations	FO1
CA.O.ListReportingOperations	FO2
CA.O.RegistrationOperations	FO3
CA.O.SingleReportingOperations	FO2.1

B.1.3 Requisiti - componenti

Componente	Requisito
FO1	SH.B.ReportingBO
	SR.M.D.ReportingDao
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CW.G.UserView
	CW.G.SignalerView
	CW.G.W.P.ReportingForm
	CW.G.W.P.UserReportingForm
	CW.G.W.P.SignalerReportingForm
	CA.G.ReportingForwarding
FO1.1	CA.O.SubmitReportingOperations
	SH.B.ReportingBO
	SR.M.D.ReportingDao
	CW.G.W.P.ReportingForm
	CW.G.W.P.UserReportingForm
	CW.G.W.P.SignalerReportingForm
	CA.G.GenericInfoView
FO1.2	CA.G.GravityView
	SH.B.ReportingBO
	SH.B.CatastropheBO
	SR.M.D.ReportingDao
	SR.M.D.CatastropheDao
	CW.G.W.B.P.ReportingForm
	CA.G.LocalizationView
FO1.2.1	CA.O.CatastropheOperations
	SH.B.CustomMap
	CW.G.W.U.CatastropheListBox
FO1.3	CA.G.LocalizationView
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
FO1.4	SH.B.ReportingBO
	SR.M.D.ReportingDao
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter

Componente	Requisito
FO1.5	SH.B.ReportingBO
	SR.M.D.ReportingDao
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CA.G.ReportingForwarding
FO1.6	SH.B.ReportingBO
	SR.M.D.ReportingDao
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CW.G.W.B.P.ReportingForm
	CW.G.W.B.P.UserReportingForm
	CW.G.W.B.P.SignalerReportingForm
	CW.G.W.U.CustomMap
FO1.7	CA.G.LocalizationView
	SH.B.ReportingBO
	SR.M.D.ReportingDao
	SH.B.Status
	SR.M.S.SignalerActions
	SR.M.S.OperatorActions
	SR.C.SignalerController
	SR.C.OperatorController
	CW.P.SignalerPresenter
	CW.P.OperatorPresenter
	CW.G.W.B.A.AuthorityReportingDetailsBody
FO1.8	CA.G.ReportingDetails
	CA.G.ReportingListView
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CW.G.W.B.P.ReportingForm
	CW.G.W.B.P.UserReportingForm
	CA.G.PreferencesView

Componente	Requisito
FO2	SR.M.S.OperatorActions
	SR.C.OperatorController
	SR.C.CommonController
	CW.P.OperatorPresenter
	CW.G.OperatorView
	CW.G.W.B.A.AuthorityReportingListBody
	CW.G.W.B.A.AuthorityReportingDetailsBody
	CA.G.ReportingListView
	CA.O.ListReportingOperations
FO2.1	SH.B.CommentBO
	SH.B.ReportingBO
	SR.M.D.CommentDao
	SR.M.D.ReportingDao
	SR.M.S.CommonActions
	CW.P.SignalerPresenter
	CW.G.W.B.P.SignalerReportingDetails
	CA.G.ReportingDetails
	CA.O.SingleReportingOperation
FO3	SH.B.SignalerBO
	SR.M.D.SignalerDao
	SR.M.S.UserActions
	SR.C.UserController
	CW.P.UserPresenter
	CW.G.W.B.P.RegisterForm
	CA.G.RegistrationView
	CA.O.RegistrationOperations
FO3.1	CW.G.W.B.P.RegisterForm
	CA.G.RegistrationView
FO3.2	CW.G.W.B.P.RegisterForm
	CA.G.RegistrationView
FO4	SR.M.S.UserActions
	SR.C.UserController
	CW.P.UserPresenter
	CW.G.W.U.Login
	CA.G.PreferencesView

Componente	Requisito
FO4.1	CW.P.Initializer
	SH.B.GenericUserBO
	CW.G.SignalerView
	CW.G.OperatorView
	CW.G.AdminView
	CW.G.SystemAdminView
	CA.G.MainView
FO4.2	SH.B.SignalerBO
	SR.M.D.SignalerDao
	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CW.G.W.B.P.RegisterForm
	CA.G.RegistrationView
FO4.3	SR.M.S.SignalerActions
	SR.C.SignalerController
	CW.P.SignalerPresenter
	CW.G.W.B.P.ChangeDataBody
FO5	SR.M.S.AdminActions
	SR.C.AdminController
	CW.P.AdminPresenter
	CW.G.AdminView
FO5.1	SR.M.S.AdminActions
	SR.C.AdminController
	CW.P.AdminPresenter
	CW.G.AdminView
	CW.G.W.B.A.CatastropheCreation
FO5.1.1	SH.B.CatastropheBO
	SR.M.D.CatastropheDao
	SR.M.S.AdminActions
	SR.C.AdminController
	CW.P.AdminPresenter
FO5.1.2	SH.B.CatastropheBO
	SR.M.D.CatastropheDao
	CW.G.W.B.A.CatastropheCreation

Componente	Requisito
FO5.1.3	SH.B.CatastropheBO SR.M.D.CatastropheDao CW.G.W.B.A.CatastropheCreation CW.G.W.U.CustomMap
FO5.2	CW.G.W.B.A.CatastropheDetails SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter
FO5.3	SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter CW.G.W.B.A.OperatorCreation
FO5.4	SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter CW.G.W.B.A.OperatorGroupCreation
FO5.5	SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter CW.G.W.B.A.OperatorDetails CW.G.W.B.A.OperatorGroupDetails
FO5.6	SH.B.CatastropheManagerBO SR.M.D.CatastropheManagerDao SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter
FO5.6.1	SH.B.OperatorGroupBO SR.M.D.OperatorGroupDao SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter CW.G.W.B.A.OperatorGroupDetails
FO5.8	SH.B.CatastropheBO SR.M.D.CatastropheDao SR.M.S.AdminActions SR.C.AdminController CW.P.AdminPresenter CW.G.W.B.A.CatastropheDetails

Componente	Requisito
FO6	SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.OperatorView CW.G.W.B.A.AuthorityReportingListBody CW.G.W.B.A.AuthorityReportingDetailsBody
FO6.1	SH.B.OperatorBO SR.M.D.OperatorDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter
FO6.4	CW.G.W.B.A.AuthorityReportingListBody
FO6.5	SH.B.CommentBO SH.B.ReportingBO SR.M.D.CommentDao SR.M.D.ReportingDao SR.M.S.CommonActions SR.C.CommonController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportingDetailsBody
FO6.6	SH.B.ReportingBO SH.B.Priority SR.M.D.ReportingDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportingDetailsBody
FO6.7	SH.B.ReportingBO SR.M.D.ReportingDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportingDetailsBody

Componente	Requisito
FO6.8	SH.B.ReportingBO SH.B.Gravity SR.M.D.ReportingDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportitngDetailsBody
FO6.9	SH.B.ReportingBO SH.B.OperatorBO SH.B.OperatorGroupBO SR.M.D.ReportingDao SR.M.D.OperatorDao SR.M.D.OperatorGroupDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportingDetailsBody
FO6.9.1	SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter
FO6.9.2	SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter
FO6.10	SH.B.ReportingBO SR.M.D.ReportingDao SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter CW.G.W.B.A.AuthorityReportitngDetailsBody CW.G.W.U.CustomMap
FO6.11	SR.M.S.OperatorActions SR.C.OperatorController CW.P.OperatorPresenter

Componente	Requisito
FO7	SH.B.ReportingBO
	SH.B.HistoryBO
	SR.M.D.HistoryDao
	SR.M.S.OperatorActions
	SR.C.OperatorController
	CW.P.OperatorPresenter
FO8	SH.B.AuthorityBO
	SR.M.D.AuthorityDao
	SR.M.S.SystemAdminActions
	SR.C.SystemAdminController
	CW.P.SystemAdminPresenter
	CW.G.W.B.S.AuthorityCreationForm
FO8.1	SH.B.OperatorBO
	SR.M.D.OperatorDao
	SR.M.S.AdminActions
	CW.P.AdminPresenter
	CW.G.W.B.S.AuthorityAdminCreationForm
FO8.2	SH.B.AuthorityBO
	SR.M.D.AuthorityDao
	SR.M.S.SystemAdminActions
	SR.C.SystemAdminController
	CW.P.SystemAdminPresenter
	CW.G.W.B.S.AuthorityDisableForm
FD2	SH.B.ReportingBO
	SH.B.CatastropheBO
	SR.M.D.ReportingDao
	SR.M.D.CatastropheDao
	SR.M.S.AdminActions
	SR.C.AdminController
	CW.P.AdminPresenter
	CW.G.W.B.A.AdminStatisticsWdg
	CW.G.W.U.PieChart
	CW.G.W.U.BarChart

Componente	Requisito
FD3	SH.B.AuthorityBO
	SH.B.OperatorBO
	SH.B.CatastropheBO
	SH.B.ReportingBO
	SR.M.D.AuthorityDao
	SR.M.D.OperatorDao
	SR.M.D.CatastropheDao
	SR.M.D.ReportingDao
	SR.M.S.SystemAdminActions
	SR.C.SystemAdminController
	CW.P.SystemAdminPresenter
	CW.G.W.B.S.SystemAdminStatisticsWdg
	CW.G.W.U.PieChart
	CW.G.W.U.BarChart
FD4	SR.M.S.OperatorActions
	SR.C.OperatorController
	CW.P.OperatorPresenter
	CW.G.W.B.A.CsvDownloaderBody
IO1	CW.G.UserView
	CW.G.SignalerView
IO1.1	CW.G.UserView
	CW.G.SignalerView
IO3	CW.G.OperatorView
	CW.G.AdminView

B.2 Esito della campagna di analisi dinamica

B.2.1 β -testing e test di accettabilità

In questo paragrafo si presentano i risultati ottenuti dai β -tester, sia sull'utilizzo del sistema, sia sui test di accettabilità. È stato fornito un modulo ai tester per ottenere un feedback preciso sui punti di progetto considerati importanti o critici, nonché le proprie impressioni di utilizzo ed eventuali bug riscontrati. I risultati ottenuti hanno permesso di individuare i punti più critici del sistema e agire esattamente dove gli utenti riscontravano problemi. Dieci persone si sono rese disponibili per effettuare il β -testing.

Test	Bug rilevati	Semplicità (%)	Tempo apprendimento (min)
TA1	13	90	2
TA2	18	80	14
TA3	9	70	28
TA4	1	90	7
TA5	1	100	1
TA6	0	100	1

Ad ogni feedback ricevuto è stata associata una issue e, una volta risolta, è stato fornito ai tester una nuova versione dell'applicazione in modo da ottenere sempre dei feedback aggiornati allo stato del progetto. Possiamo osservare che le funzionalità richieste sono state completate, e l'apprendimento delle funzionalità dell'applicazione ha richiesto un tempo ragionevole agli utenti.

Test	Stato
TA1	Completato
TA2	Completato
TA3	Completato
TA4	Completato
TA5	Completato
TA6	Completato

Come si può osservare dalla precedente tabella i test di accettabilità risultano completamente coperti, ossia tutte le funzionalità obbligatorie sono state implementate.

B.2.2 Test effettuati

Osserviamo dettagliatamente i risultati ottenuti dei test effettuati. Si osservi che ad ogni Testsuite corrispondono più test. Per ragioni di spazio il suffisso `it.donkeycode.rescueme` viene omissso nel nome dei packages.

Test naming

Testsuite	Codice
.server.controller.AndroidRestServiceTest	Testsuite 1
.server.controller.CommonControllerTest	Testsuite 2
.server.controller.OperatorControllerTest	Testsuite 3
.server.model.dao.AuthorityDaoTest	Testsuite 4
.server.model.dao.CatastropheDaoTest	Testsuite 5
.server.model.dao.CatastropheManagerDaoTest	Testsuite 6
.server.model.dao.CommentDaoTest	Testsuite 7
.server.model.dao.converter.AuthorityConverterTest	Testsuite 8
.server.model.dao.converter.CatastropheConverterTest	Testsuite 9
.server.model.dao.converter.CatastropheManagerConverterTest	Testsuite 10
.server.model.dao.converter.CommentConverterTest	Testsuite 11
.server.model.dao.converter.HistoryConverterTest	Testsuite 12
.server.model.dao.converter.ImageConverterTest	Testsuite 13
.server.model.dao.converter.OperatorConverterTest	Testsuite 14
.server.model.dao.converter.OperatorGroupConverterTest	Testsuite 15
.server.model.dao.converter.ReportingConverterTest	Testsuite 16
.server.model.dao.converter.SignalerConverterTest	Testsuite 17
.server.model.dao.converter.SystemAdminConverterTest	Testsuite 18
.server.model.dao.CustomImageDaoTest	Testsuite 19
.server.model.dao.HistoryDaoTest	Testsuite 20
.server.model.dao.OperatorDaoTest	Testsuite 21
.server.model.dao.OperatorGroupDaoTest	Testsuite 22
.server.model.dao.ReportingDaoTest	Testsuite 23
.server.model.dao.SignalerDaoTest	Testsuite 24
.server.model.dao.SystemAdminDaoTest	Testsuite 25
.server.model.services.AdminActionsTest	Testsuite 26
.server.model.services.GenericUserActionsTest	Testsuite 27
.server.model.services.OperatorActionsTest	Testsuite 28
.server.model.services.SystemAdminActionsTest	Testsuite 29
.servicesdaointegration.OperatorActionsDaoIntegrationTest	Testsuite 30
.shared.businessobjects.BusinessObjects	Testsuite 31
resources.BuildersTest	Testsuite 32
resources.FileParserTest	Testsuite 33

Test results

Testsuite	Test eseguiti	Fallimenti	Errori	Tempo impiegato
Testsuite 1	16	0	0	0,839 sec
Testsuite 2	2	0	0	0,540 sec
Testsuite 3	2	0	0	0,477 sec
Testsuite 4	5	0	0	2,375 sec
Testsuite 5	5	0	0	1,241 sec
Testsuite 6	5	0	0	1,275 sec
Testsuite 7	5	0	0	1,212 sec
Testsuite 8	2	0	0	0,319 sec
Testsuite 9	2	0	0	0,312 sec
Testsuite 10	2	0	0	0,323 sec
Testsuite 11	2	0	0	0,326 sec
Testsuite 12	2	0	0	0,334 sec
Testsuite 13	2	0	0	0,319 sec
Testsuite 14	2	0	0	0,332 sec
Testsuite 15	2	0	0	0,336 sec
Testsuite 16	2	0	0	0,500 sec
Testsuite 17	2	0	0	0,341 sec
Testsuite 18	2	0	0	0,316 sec
Testsuite 19	5	0	0	1,309 sec
Testsuite 20	5	0	0	1,294 sec
Testsuite 21	5	0	0	1,241 sec
Testsuite 22	5	0	0	1,233 sec
Testsuite 23	5	0	0	1,253 sec
Testsuite 24	5	0	0	1,316 sec
Testsuite 25	5	0	0	1,280 sec
Testsuite 26	55	0	0	0,822 sec
Testsuite 27	4	0	0	0,298 sec
Testsuite 28	23	0	0	0,638 sec
Testsuite 29	8	0	0	0,507 sec
Testsuite 30	1	0	0	1,369 sec
Testsuite 31	5	0	0	0,199 sec
Testsuite 32	8	0	0	0,352 sec
Testsuite 33	1	0	0	0,231 sec

B.2.3 Test di integrazione delle componenti principali

In questo paragrafo si presenta un esempio di un test di integrazione effettuato tra la classe `OperatorActions` e il package `server.model.dao`.

Test T11: OperatorActions - Dao

```
public class OperatorActionsDaoIntegrationTest {

    private final LocalServiceTestHelper helper =
        new LocalServiceTestHelper(new LocalDatastoreServiceTestConfig());
    OperatorActions operatorActions;
    IDao<HistoryBO> historyDao;
    IDao<ReportingBO> reportingDao;
    IDao<CommentBO> commentDao;
    HistoryConverter historyConverter;
    ReportingConverter reportingConverter;
    CommentConverter commentConverter;
    TestFileParser parser;
    ObjectBuilder objectBuilder;
    List<String> inputs;
    List<ReportingBO> reporting;

    @Before
    public void setUp() {
        helper.setUp();
        try {
            parser = new TestFileParser(
                "test-classes/resources/DaoTestFile.txt");
            inputs = parser.getDataFromFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
        objectBuilder = new ObjectBuilder(inputs);
        operatorActions = new OperatorActions();
        historyDao = new HistoryDao();
        reportingDao = new ReportingDao();
        commentDao = new CommentDao();
        historyConverter = new HistoryConverter();
        reportingConverter = new ReportingConverter();
        commentConverter = new CommentConverter();
        reporting = new ArrayList<ReportingBO>();
        reporting = objectBuilder.constructReportingBO();
        ReflectionTestUtils.setField(
            reportingDao, "reportingConverter", reportingConverter);
        for (int i=0;i<reporting.size();i++) {
            reportingDao.create(reporting.get(i));
        }
    }

    @After
    public void tearDown() {
        helper.tearDown();
    }

    @Test
    public void updateReportingTest() {
        ReflectionTestUtils.setField(
            historyDao, "historyConverter", historyConverter);
        ReflectionTestUtils.setField(
            operatorActions, "historyDao", historyDao);
        ReflectionTestUtils.setField(
            operatorActions, "reportingDao", reportingDao);
        ReflectionTestUtils.setField(
            reportingConverter, "commentDao", commentDao);
        ReflectionTestUtils.setField(
            commentDao, "commentConverter", commentConverter);

        ReportingBO updated = new ReportingBO();
        updated.setKey(reportingDao.read(
            "author =="+reporting.get(0).getAuthor()+"", ""
        ).get(0).getKey());

        updated.setAuthor(reporting.get(0).getAuthor()+"_updated");
    }
}
```

```

        updated.setPosition(new CustomGeoPt(12.4f, 46.4f));
        updated.setPriority(new Priority(ePriority.Bassa));
        updated.setGravity(new Gravity(eGravity.Bassa));
        updated.setStatus(new Status(eState.Conclusa));

        assertTrue(operatorActions.updateReporting(updated));

        updated = reportingDao.read(
            "author == '"+updated.getAuthor()+"'", "").get(0);

        assertEquals(updated.getAuthor(),
            reporting.get(0).getAuthor() + ".updated");
    }
}

```

È interessante osservare come la complessità nell'effettuare i test di integrazione aumenti in maniera sensibile rispetto ai test di unità. Questo perché per effettuare dei test significativi, è necessario coprire il maggior numero di diramazioni del codice. All'aumentare delle classi da testare e della complessità di queste, ottenere una buona branch coverage diventa sempre più complesso.

B.2.4 Test di integrazione delle sotto-componenti

In questo paragrafo si presenta un esempio di test di integrazione delle classi `AuthorityDao` e `AuthorityConverter`.

Test TIS1: AuthorityDao - AuthorityConverter

```

public class AuthorityDaoTest {

    private final LocalServiceTestHelper helper =
        new LocalServiceTestHelper(new LocalDatastoreServiceTestConfig());
    private ObjectBuilder objectBuilder;
    private TestFileParser parser;
    private AuthorityDao dao;
    private List<String> inputs=new ArrayList<String>();
    private List<AuthorityTO> to;
    private List<AuthorityBO> bo;
    private AuthorityConverter authorityConverter;
    String filter;
    String ordering;

    @Before
    public void setUp() {

        try {
            helper.setUp();
            DatastoreService ds = DatastoreServiceFactory.getDatastoreService();
            parser = new TestFileParser("test-classes/resources/DaoTestFile.txt");
            inputs = parser.getDataFromFile();
            objectBuilder = new ObjectBuilder(inputs);
            bo = new ArrayList<AuthorityBO>();
            authorityConverter = new AuthorityConverter();
            to = objectBuilder.constructAuthorityTO();
            for(int i=0;i<to.size(); i++) {
                Entity temp = new Entity("AuthorityTO");
                to.get(i).setKey(temp.getKey());
                temp.setProperty("name", to.get(i).getName());
                temp.setProperty("active", to.get(i).isActive());
                ds.put(temp);
                bo.add(authorityConverter.toBO(to.get(i)));
            }
            dao = new AuthorityDao();
            ReflectionTestUtils.setField(dao, "authorityConverter", authorityConverter);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    @After
    public void tearDown() {
        helper.tearDown();
    }

    @Test
    public void readByIdTest(){
        for(int i=0; i<to.size(); i++) {
            AuthorityBO result = dao.readById(to.get(i).getKey());
            assertEquals(result.getName(), to.get(i).getName());
            assertEquals(result.isActive(), to.get(i).isActive());
        }
    }

    @Test
    public void readTest(){
        filter = "name==" + to.get(0).getName() + " ";
        ordering = "";
        List<AuthorityBO> results = dao.read(filter, ordering);
        AuthorityBO result = results.get(0);
        assertEquals(result.getName(), to.get(0).getName());
        assertEquals(result.isActive(), to.get(0).isActive());
    }

    @Test
    public void updateTest(){
        String newName = "nuovoNome";
        filter = "name==" + newName + " ";
        ordering = "";
        bo.get(0).setName(newName);
        bo.get(0).setActive(false);
        dao.update(bo.get(0));
        List<AuthorityBO> results = dao.read(filter, ordering);
        AuthorityBO result = results.get(0);
        assertEquals(result.getName(), newName);
        assertEquals(result.isActive(), false);
    }

    @Test
    public void deleteTest(){
        filter = "name==" + bo.get(0).getName() + " ";
        ordering = "";
        dao.delete(bo.get(0).getKey());
        List<AuthorityBO> results = dao.read(filter, ordering);
        assertEquals(results.size(), 0);
    }

    @Test
    public void createTest(){
        AuthorityBO auth = new AuthorityBO();
        auth.setName("authority");
        auth.setActive(false);
        filter = "name==" + auth.getName() + " ";
        ordering = "";
        dao.create(auth);
        List<AuthorityBO> results = dao.read(filter, ordering);
        assertEquals(results.size(), 1);
        assertEquals(results.get(0).getName(), auth.getName());
    }
}

```

B.2.5 Test di unità

In questo paragrafo si presenta un esempio di test di unità effettuato per la classe `SystemAdminActions`, con uso della libreria `EasyMock` attraverso il framework `JUnit`.

Test TU1: Classe `SystemAdminActions`

```

public class SystemAdminActionsTest {
    SystemAdminActions systemAdminActions;

```



```

TestFileParser parser;
ObjectBuilder objectBuilder;
List<String> inputs;
List<SignalerBO> signaler;
List<OperatorBO> operator;
List<ReportingBO> reporting;
List<OperatorGroupBO> operatorGroup;
List<CatastropheBO> catastrophes;
List<AuthorityBO> authorities;
private IDao<GenericUserBO> systemAdminDaoMock;
private AuthorityDao authorityDaoMock;
private IDao<CatastropheBO> catastropheDaoMock;
private IDao<OperatorGroupBO> operatorGroupDaoMock;
private IDao<OperatorBO> operatorDaoMock;
private IDao<SignalerBO> signalerDaoMock;

@Before
public void setUp() {
    try {
        parser = new TestFileParser("test-classes/resources/DaoTestFile.txt");
        inputs = parser.getDataFromFile();
    } catch (Exception e) {
        LogFile.getInstance().addLog("File non Trovato");
    }
    objectBuilder = new ObjectBuilder(inputs);
    systemAdminActions = new SystemAdminActions();
    signaler = new ArrayList<SignalerBO>();
    operator = new ArrayList<OperatorBO>();
    reporting = new ArrayList<ReportingBO>();
    operatorGroup = new ArrayList<OperatorGroupBO>();
    signaler = objectBuilder.constructSignalerBO();
    operator = objectBuilder.constructOperatorBO();
    reporting = objectBuilder.constructReportingBO();
    catastrophes = objectBuilder.constructCatastropheBO();
    operatorGroup = objectBuilder.constructOperatorGroupBO();
    authorities = objectBuilder.constructAuthorityBO();
    systemAdminDaoMock = EasyMock.createMock(SystemAdminDao.class);
    authorityDaoMock = EasyMock.createMock(AuthorityDao.class);
    catastropheDaoMock = EasyMock.createMock(CatastropheDao.class);
    operatorGroupDaoMock = EasyMock.createMock(OperatorGroupDao.class);
    operatorDaoMock = EasyMock.createMock(OperatorDao.class);
    signalerDaoMock = EasyMock.createMock(SignalerDao.class);
}

@After
public void tearDown() {
}

/*
 * Conclude correttamente
 */
@Test
public void addAuthorityTest1() {
    String arg1 = (String)EasyMock.anyObject();
    String arg2 = (String)EasyMock.anyObject();
    EasyMock.expect(
        authorityDaoMock.read(arg1, arg2)).andReturn(new ArrayList<AuthorityBO>());
    EasyMock.expect(authorityDaoMock.create(authorities.get(0))).andReturn(true);
    EasyMock.replay(authorityDaoMock);
    ReflectionTestUtils.setField(
        systemAdminActions, "authorityDao", authorityDaoMock);
    try {
        assertTrue(systemAdminActions.addAuthority(authorities.get(0)));
        EasyMock.verify(authorityDaoMock);
    } catch (AlreadyExistentException e) {
        assertTrue(false);
    }
}

/*
 * Lancia eccezione
 */
@Test
public void addAuthorityTest2() {
    EasyMock.expect(
        authorityDaoMock.read((String)EasyMock.anyObject(),

```

```

        (String) EasyMock.anyObject()).andReturn(authorities);
EasyMock.replay(authorityDaoMock);
ReflectionTestUtils.setField(systemAdminActions, "authorityDao", authorityDaoMock);
try {
    assertTrue(!systemAdminActions.addAuthority(authorities.get(0)));
    EasyMock.verify(authorityDaoMock);

} catch (AlreadyExistsException e) {
    assertTrue(true);
}
}

/*
 * Conclude correttamente
 */
@Test
public void disableAuthorityTest() {
    EasyMock.expect(authorityDaoMock.readById(
        (String) EasyMock.anyObject()).andReturn(authorities.get(0)));
    EasyMock.expect(authorityDaoMock.update(
        (AuthorityBO) EasyMock.anyObject()).andReturn(true);
    EasyMock.expect(
        operatorDaoMock.read((String) EasyMock.anyObject(),
            (String) EasyMock.anyObject()).andReturn(operator);
    EasyMock.expect(
        operatorDaoMock.update((OperatorBO) EasyMock.anyObject()).andReturn(true);
    EasyMock.expect(
        operatorGroupDaoMock.read((String) EasyMock.anyObject(),
            (String) EasyMock.anyObject()).andReturn(operatorGroup);
    EasyMock.expect(
        operatorGroupDaoMock.update(
            (OperatorGroupBO) EasyMock.anyObject()).andReturn(true);
    EasyMock.expect(
        catastropheDaoMock.read((String) EasyMock.anyObject(),
            (String) EasyMock.anyObject()).andReturn(catastrophes);
    EasyMock.expect(
        catastropheDaoMock.update(
            (CatastropheBO) EasyMock.anyObject()).andReturn(true);
    EasyMock.replay(authorityDaoMock);
    EasyMock.replay(operatorDaoMock);
    EasyMock.replay(operatorGroupDaoMock);
    EasyMock.replay(catastropheDaoMock);
    ReflectionTestUtils.setField(
        systemAdminActions, "authorityDao", authorityDaoMock);
    ReflectionTestUtils.setField(
        systemAdminActions, "operatorDao", operatorDaoMock);
    ReflectionTestUtils.setField(
        systemAdminActions, "operatorGroupDao", operatorGroupDaoMock);
    ReflectionTestUtils.setField(
        systemAdminActions, "catastropheDao", catastropheDaoMock);

    assertTrue(systemAdminActions.disableAuthority(authorities.get(0).getName()));
    EasyMock.verify(authorityDaoMock);
    EasyMock.verify(operatorDaoMock);
    EasyMock.verify(operatorGroupDaoMock);
    EasyMock.verify(catastropheDaoMock);
}

/*
 * Deve concludere correttamente
 */
@Test
public void systemSetupTest() {
    EasyMock.expect(
        signalerDaoMock.read("username=='anonymous'", "")
            .andReturn(new ArrayList<SignalerBO>());
    EasyMock.expect(
        signalerDaoMock.create((SignalerBO) EasyMock.anyObject())
            .andReturn(true);
    EasyMock.expect(
        systemAdminDaoMock.create((GenericUserBO) EasyMock.anyObject())
            .andReturn(true);
    EasyMock.expect(
        systemAdminDaoMock.read((String) EasyMock.anyObject(),
            (String) EasyMock.anyObject()).andReturn(new ArrayList<GenericUserBO>());
    EasyMock.replay(signalerDaoMock);

```

```
EasyMock.replay(systemAdminDaoMock);
ReflectionTestUtils.setField(systemAdminActions,
    "signalerDao", signalerDaoMock);
ReflectionTestUtils.setField(systemAdminActions,
    "systemAdminDao", systemAdminDaoMock);
assertTrue(systemAdminActions.systemSetup());
EasyMock.verify(systemAdminDaoMock);
EasyMock.verify(signalerDaoMock);
}
}
```

L'annotazione `@Before` indica il metodo che deve essere eseguito prima dell'invocazione di ogni metodo di test, rappresentato dall'annotazione `@Test`. In questo modo, ogni test è indipendente dagli altri in quanto vengono allocate ogni volta nuove risorse su cui vengono eseguiti i test. Nell'esempio precedente le invocazioni dei metodi avranno il seguente ordine:

1. `public void setUp();`
2. `public void addAuthorityTest1();`
3. `public void setUp();`
4. `public void addAuthorityTest2();`
5. `public void setUp();`
6. `public void disableAuthorityTest();`
7. `public void setUp();`
8. `public void systemSetupTest();`

I metodi precedenti che sono usati nell'esempio, di seguito sono descritti in dettaglio:

- `public void setUp()`: effettua il parse del file testuale contenente la definizione degli oggetti da creare per questo test tramite le chiamate `parser.newTestFileParser`, `parser.getDataFromFile`. Successivamente costruisce gli oggetti contenuti nella stringa di testo ottenuta (opportunamente formattata) tramite le chiamate alla classe `ObjectBuilder` (classe sviluppata internamente per scopi di testing, opportunamente testata prima di utilizzarla per avere la certezza del suo corretto funzionamento). Una volta ottenuti gli oggetti utili al test, si procede con la creazione dei mock objects, tramite le chiamate al metodo `EasyMock.createMock(tipo_della_classe)`;
- `public void addAuthorityTest1()`: imposta i parametri attesi e i risultati che i mock objects devono ritornare. Successivamente imposta il campo dati `private SystemAdminActions.authorityDao` tramite la chiamata alla classe di

utilità `ReflectionsTestUtils.setField()` ed infine va ad effettuare la chiamata al metodo `SystemAdminActions.addAuthority()`, verificando che non siano lanciate eccezioni;

- `public void addAuthorityTest2()`: imposta i parametri attesi e i risultati che i mock objects devono ritornare. Successivamente imposta il campo dati privato `SystemAdminActions.authorityDao` tramite la chiamata alla classe di utilità `ReflectionsTestUtils.setField()` ed infine va ad effettuare la chiamata al metodo `SystemAdminActions.addAuthority()`, verificando che venga correttamente lanciata un'eccezione di tipo `AlreadyExistentException`;
- `public void disableAuthorityTest()`: imposta i parametri attesi e i risultati che i mock objects devono restituire. Successivamente va ad impostare i vari campi privati necessari al test della classe `SystemAdminActions` tramite chiamate alla classe `ReflectionsTestUtils.setField()` ed infine va ad effettuare la chiamata al metodo `SystemAdminActions.disableAuthority()`, verificando che concluda correttamente;
- `public void systemSetupTest()`: imposta i parametri attesi e i risultati che i mock objects devono restituire. Successivamente va ad impostare i vari campi privati necessari al test della classe `SystemAdminActions` tramite chiamate alla classe `ReflectionsTestUtils.setField()` ed infine va ad effettuare la chiamata al metodo `SystemAdminActions.systemSetup()`, verificando che concluda correttamente.

JUnit offre molti metodi per fare asserzioni come:

- `assertEquals`: verifica che due valori siano uguali. Esistono diverse ridefinizioni del metodo per testare oggetti di tipo diverso (p.es stringhe, interi, array, ecc...);
- `assertTrue`: verifica che la valutazione della condizione data in input ritorni un valore affermativo. Esiste anche il duale `assertFalse`;
- `assertNotNull`: verifica che un riferimento non sia nullo cioè si riferisca effettivamente ad un oggetto. Per simulare il comportamento degli oggetti tramite EasyMock bisogna seguire la seguente serie di passi:
 - Istanziamento di un mock object[g] riferito ad una particolare classe o interfaccia tramite il metodo `EasyMock.createMock(NomeClasse.class)`;
 - Registrazione del comportamento del mock object: dovranno esser indicati i valori ritornati dai vari metodi dell'oggetto e il numero di volte che questi dovranno essere invocati durante il test. Il metodo `expect(mockobj.metodo()).setRitorno(valore).occorrenze(n)` assolve

questo compito. Esistono svariate possibilità per indicare i valori di ritorno e i parametri in entrata;

- Attivare il mock object, tramite il metodo `replay(oggetto_mock)`;
- Verificare che il comportamento del mock object sia stato rispettato, tramite il metodo `verify(oggetto_mock)`. In caso di comportamento errato dell'oggetto, verrà sollevata un'eccezione che identifica il problema.

B.3 Copertura del codice sorgente

Nel seguente capitolo è presente una panoramica riguardante la copertura del codice rispetto sia alle linee presenti nei sorgenti, sia alle possibili ramificazioni dovute a salti condizionali del codice. Nella verifica della copertura del codice sono esclusi i package riguardanti la grafica. I riferimenti sui livelli minimi di copertura da assicurare per lo sviluppo del progetto sono descritti nel capitolo 8.6 nel documento Norme-di-progetto-4.0.0.pdf. I risultati sono stati ottenuti grazie allo script del processo di build e il tool Cobertura.

Informazioni più dettagliate riguardanti la copertura del codice del progetto è disponibile all'indirizzo: <http://donkeycode2010.altervista.org/coberturaresults>.

Packages:

A livello di progetto risulta raggiunta la copertura preventivata. Questo permette di garantire la qualità sulla parte di funzionalità del codice. Osserviamo il grafico rappresentante la copertura dei vari package:

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	124	73% 2287/3130	68% 367/534	1,589
it.donkeycode.rescueme.server	3	23% 8/34	N/A	1,182
it.donkeycode.rescueme.server.controller	15	62% 333/532	52% 49/94	1,955
it.donkeycode.rescueme.server.model.dao	14	69% 496/716	97% 45/46	3,145
it.donkeycode.rescueme.server.model.dao.converter	13	98% 250/253	89% 25/28	1,56
it.donkeycode.rescueme.server.model.dao.transferobjects	12	95% 233/244	N/A	1
it.donkeycode.rescueme.server.model.services	13	69% 561/804	63% 192/302	2,447
it.donkeycode.rescueme.shared.businessobjects	30	74% 383/517	87% 56/64	1,255
it.donkeycode.rescueme.shared.exceptions	11	76% 23/30	N/A	1
it.donkeycode.rescueme.shared.gwt	13	N/A	N/A	1

Figura 4: Copertura dei packages

Di seguito si presentano in dettaglio per ogni package i risultati di copertura ottenuti. Si noti che il package `server` contiene le classi `LogFile` e `ServerConst`. Sono stati omessi i risultati di questo package poiché le classi contenute servono rispettivamente per il de-bug e come contenitore di informazioni esterne al programma, quali indirizzo del server e indirizzo mail del gruppo di sviluppo.

Package server.controller:

La maggior parte delle classi di questo package non è testata, questo è imputabile al fatto che nella maggior parte dei casi le componenti interne si limitano a effettuare invocazioni di metodi delle classe del package `server.services`. Si osservi che le classe `AndroidRestService` è testata in modo approfondito, essendo l'unica classe con una complessità tale da richiedere tale sforzo. Il package risulta comunque avere una copertura sulle linee di codice del 62 %, mentre la copertura sulle ramificazioni è al 52%. La complessità ciclomatica media è 1.965.

Coverage Report - it.donkeycode.rescueme.server.controller


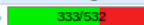
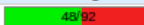

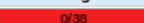
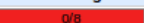


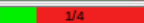
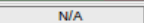
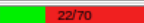
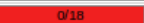
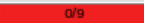
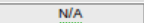
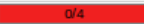
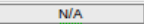

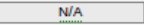

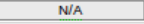


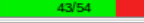
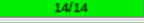



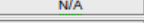



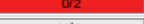
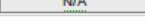
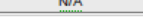
Package 	# Classes	Line Coverage	Branch Coverage	Complexity
it.donkeycode.rescueme.server.controller	15	62%  333/532	52%  48/92	1,965
Classes in this Package 		Line Coverage	Branch Coverage	Complexity
AdminController		0%  0/38	0%  0/8	1,143
AndroidRestService		89%  267/297	73%  34/46	5,545
BaseRemoteService		25%  1/4	N/A  N/A	1
CommonController		31%  22/70	0%  0/18	2,75
ConfirmRegistrationService		0%  0/9	N/A  N/A	2
GwtAuthenticationFailureHandler		0%  0/4	N/A  N/A	1
GwtAuthenticationSuccessHandler		0%  0/4	N/A  N/A	1
GwtLogoutSuccessHandler		0%  0/4	N/A  N/A	1
InitSystem		0%  0/8	0%  0/2	2
OperatorController		79%  43/54	100%  14/14	1,583
RescueMeUserDetailsService		0%  0/12	0%  0/2	5
SignalerController		0%  0/6	N/A  N/A	1
SystemAdminController		0%  0/11	N/A  N/A	1
UserController		0%  0/11	0%  0/2	1,4
package-info		N/A  N/A	N/A  N/A	0

Figura 5: Copertura package `server.controller`

Package server.model.dao:

Il package `server.model.dao` ha copertura sulle linee di codice del 69%, mentre la copertura sulle ramificazioni è al 97%. La complessità ciclomatica media è 3.145.

Coverage Report - it.donkeycode.rescueme.server.model.dao

Package /	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.server.model.dao	14	69%	496/716	97%	45/46	3,145
it.donkeycode.rescueme.server.model.dao.converter	13	98%	250/253	89%	25/28	1,56
it.donkeycode.rescueme.server.model.dao.transferobjects	12	95%	233/244	N/A	N/A	1
Classes in this Package /		Line Coverage		Branch Coverage		Complexity
AuthorityDao		66%	40/60	100%	4/4	3,4
CatastropheDao		69%	46/66	83%	5/6	3,6
CatastropheManagerDao		69%	45/65	100%	4/4	3,4
CommentDao		67%	42/62	100%	4/4	3,4
HistoryDao		72%	49/68	100%	4/4	3,4
IDao		N/A	N/A	N/A	N/A	1
ImageDao		67%	41/61	100%	4/4	3,4
OperatorDao		68%	44/64	100%	4/4	3,4
OperatorGroupDao		69%	45/65	100%	4/4	3,4
PMF		80%	4/5	N/A	N/A	1
ReportingDao		73%	54/73	100%	4/4	3,4
SignalerDao		68%	44/64	100%	4/4	3,4
SystemAdminDao		66%	40/60	100%	4/4	3,4
package-info		N/A	N/A	N/A	N/A	0

Figura 6: Copertura package `server.model.dao`

Package server.model.dao.converter:

Il package `server.model.dao.converter` ha copertura sulle linee di codice del 98%, mentre la copertura sulle ramificazioni è al 89%. La complessità ciclomatica media è 1.560.

Coverage Report - it.donkeycode.rescueme.server.model.dao.converter

Package /	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.server.model.dao.converter	13	98%	250/253	89%	25/28	1,56
Classes in this Package /		Line Coverage		Branch Coverage		Complexity
AuthorityConverter		100%	13/13	100%	2/2	1,5
CatastropheConverter		100%	29/29	100%	2/2	1,5
CatastropheManagerConverter		100%	13/13	100%	2/2	1,5
CommentConverter		100%	19/19	100%	2/2	1,5
HistoryConverter		100%	31/31	100%	2/2	1,5
IConverter		N/A	N/A	N/A	N/A	1
ImageConverter		100%	18/18	100%	2/2	1,5
OperatorConverter		92%	25/27	66%	4/6	2,5
OperatorGroupConverter		100%	17/17	100%	2/2	1,5
ReportingConverter		97%	46/47	75%	3/4	1,667
SignalerConverter		100%	24/24	100%	2/2	1,5
SystemAdminConverter		100%	19/19	100%	2/2	1,5
package-info		N/A	N/A	N/A	N/A	0

Figura 7: Copertura package `server.model.dao.converter`

Package server.model.dao.transferobjects:

Il package `server.model.dao.transferobjects` ha copertura sulle linee di codice del 95%, mentre non presenta copertura sulle ramificazioni. Questo perché le classi al suo interno non posseggono più di una sola sequenza di esecuzione.

Coverage Report - it.donkeycode.rescueme.server.model.dao.transferobjects

Package [↗]	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.server.model.dao.transferobjects	12	95%	<div><div>233/244</div></div>	N/A	N/A	1
Classes in this Package [↗]		Line Coverage		Branch Coverage		Complexity
AuthorityTO		90%	<div><div>10/11</div></div>	N/A	N/A	1
CatastropheManagerTO		90%	<div><div>10/11</div></div>	N/A	N/A	1
CatastropheTO		96%	<div><div>28/29</div></div>	N/A	N/A	1
CommentTO		95%	<div><div>19/20</div></div>	N/A	N/A	1
CustomImageTO		94%	<div><div>18/17</div></div>	N/A	N/A	1
HistoryTO		96%	<div><div>31/32</div></div>	N/A	N/A	1
OperatorGroupTO		94%	<div><div>16/17</div></div>	N/A	N/A	1
OperatorTO		96%	<div><div>25/26</div></div>	N/A	N/A	1
ReportingTO		97%	<div><div>43/44</div></div>	N/A	N/A	1
SignalerTO		96%	<div><div>25/26</div></div>	N/A	N/A	1
SystemAdminTO		90%	<div><div>10/11</div></div>	N/A	N/A	1
package-info		N/A	N/A	N/A	N/A	0

Figura 8: Copertura package `server.model.dao.transferobjects`

Package server.model.services:

Il package `server.model.services` ha copertura sulle linee di codice del 69%, mentre la copertura sulle ramificazioni è al 63%. La complessità ciclomatica media è 2.447.

Coverage Report - it.donkeycode.rescueme.server.model.services



Package 	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.server.model.services	13	69%	<div><div></div></div> 561/804	63%	<div><div></div></div> 192/302	2,447
Classes in this Package 	Line Coverage		Branch Coverage		Complexity	
AdminActions	80%	<div><div></div></div> 245/305	59%	<div><div></div></div> 83/140	3,794	
CommonActions	0%	<div><div></div></div> 0/22	0%	<div><div></div></div> 0/8	3,25	
GenericUserActions	29%	<div><div></div></div> 16/54	50%	<div><div></div></div> 5/10	6,5	
IAdminActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
ICommonActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
IGenericUserActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
IOperatorActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
ISignalerActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
ISystemAdminActions	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	1	
OperatorActions	88%	<div><div></div></div> 213/241	78%	<div><div></div></div> 80/102	4,211	
SignalerActions	0%	<div><div></div></div> 0/57	0%	<div><div></div></div> 0/8	2,143	
SystemAdminActions	69%	<div><div></div></div> 57/125	70%	<div><div></div></div> 24/34	3	
package-info	N/A	<div><div></div></div> N/A	N/A	<div><div></div></div> N/A	0	

Figura 9: Package `server.model.services`

Package server.businessobjects:

Il package `server.businessobjects` ha copertura sulle linee di codice dell'74%, mentre la copertura sulle ramificazioni è all'87%. La complessità ciclomatica media è 1.255.

Coverage Report - it.donkeycode.rescueme.shared.businessobjects

Package /	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.shared.businessobjects	30	74%	383/517	87%	56/64	1,255
Classes in this Package /		Line Coverage		Branch Coverage		Complexity
AdminStatistics		0%	0/57	N/A	N/A	1
AuthorityBO		100%	10/10	N/A	N/A	1
CatastropheBO		100%	26/28	N/A	N/A	1
CatastropheManagerBO		100%	10/10	N/A	N/A	1
CatastropheWithPositions		71%	5/7	N/A	N/A	1
CommentBO		100%	19/19	N/A	N/A	1
CompleteHistory		80%	8/10	N/A	N/A	1
CustomGeoPt		61%	8/13	N/A	N/A	1
CustomImageBO		100%	16/16	N/A	N/A	1
Filters		78%	15/19	91%	11/12	5
Filters\$eType		66%	2/3	N/A	N/A	5
GenericUserBO		100%	20/20	N/A	N/A	1
Gravity		74%	23/31	85%	12/14	2,4
Gravity\$eGravity		100%	2/2	N/A	N/A	2,4
HistoryBO		100%	43/43	N/A	N/A	1
OperatorBO		100%	10/10	N/A	N/A	1
OperatorGroupBO		100%	16/16	N/A	N/A	1
OperatorWithGroup		71%	5/7	N/A	N/A	1
OperatorsAndGroups		100%	9/9	N/A	N/A	1
Priority		71%	23/32	85%	12/14	2,4
Priority\$ePriority		50%	1/2	N/A	N/A	2,4
ReportingBO		100%	46/46	N/A	N/A	1
ReportingWithReferences		80%	8/10	N/A	N/A	1
SignalerBO		100%	13/13	N/A	N/A	1
Status		70%	24/34	81%	13/16	2,6
Status\$eState		50%	1/2	N/A	N/A	2,6
SystemAdminStatistics		0%	0/25	N/A	N/A	1
UserType		80%	16/20	100%	8/8	2
UserType\$eTypes		66%	2/3	N/A	N/A	2
package-info		N/A	N/A	N/A	N/A	0

Figura 10: Package `server.businessobjects`

Package server.exception:

Il package `server.exception` ha copertura sulle linee di codice dell'76%, mentre non presenta copertura sulle ramificazioni. Questo perché le classi al suo interno non posseggono più di una sola sequenza di esecuzione.

Coverage Report - it.donkeycode.rescueme.shared.exceptions



Package 	# Classes	Line Coverage		Branch Coverage		Complexity
it.donkeycode.rescueme.shared.exceptions	11	76%	<div><div></div></div> 23/30	N/A	N/A	1
Classes in this Package 	Line Coverage		Branch Coverage		Complexity	
AlreadyExistentException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
CatastropheNotDefinedException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
DataAccessException	0%	<div><div></div></div> 0/3	N/A	N/A	1	
IllegalModificationException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
KeyNotFoundException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
OperatorNotDefinedException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
RescueMeException	66%	<div><div></div></div> 2/3	N/A	N/A	1	
ResourceNotFoundException	0%	<div><div></div></div> 0/3	N/A	N/A	1	
UsernameAlreadyExistentException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
UsernameNotFoundException	100%	<div><div></div></div> 3/3	N/A	N/A	1	
package-info	N/A	N/A	N/A	N/A	0	

Figura 11: Package `server.exceptions`

B.4 Esito della campagna di analisi statica

L'analisi statica effettuata sul codice ha mostrato come i valori soglia definiti nelle norme di progetto siano stati rispettati. Vengono di seguito presentati alcuni risultati sull'analisi statica. Informazioni più dettagliate riguardanti le misurazioni effettuate sono disponibili ai seguenti indirizzi:

- RescueMe - <http://donkeycode2010.altervista.org/rescuememetriche>
- RescueApp - <http://donkeycode2010.altervista.org/rescueappmetriche>

B.4.1 Metriche RescueMe

Osserviamo i risultati ottenuti sull'applicazione RescueMe, in dettaglio osserviamo i package `clientweb`, `shared` e `server`. Tutti i package rientrano nelle soglie definite nelle norme di progetto.

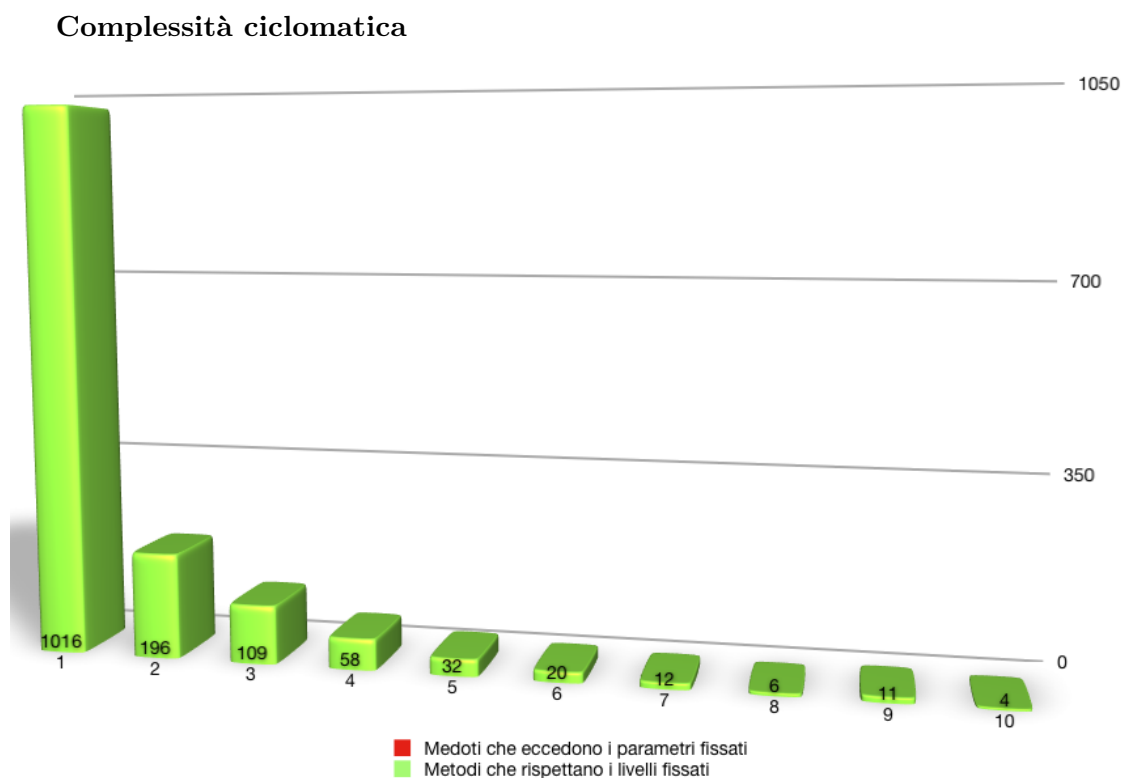


Figura 12: Complessità ciclomatica RescueMe

Come si può osservare dalla figura precedente la complessità ciclomatica dei metodi di RescueMe rientra nei limiti fissati nelle norme di progetto. La maggior parte dei metodi hanno complessità ciclomatica di 1.

Annidamento del codice

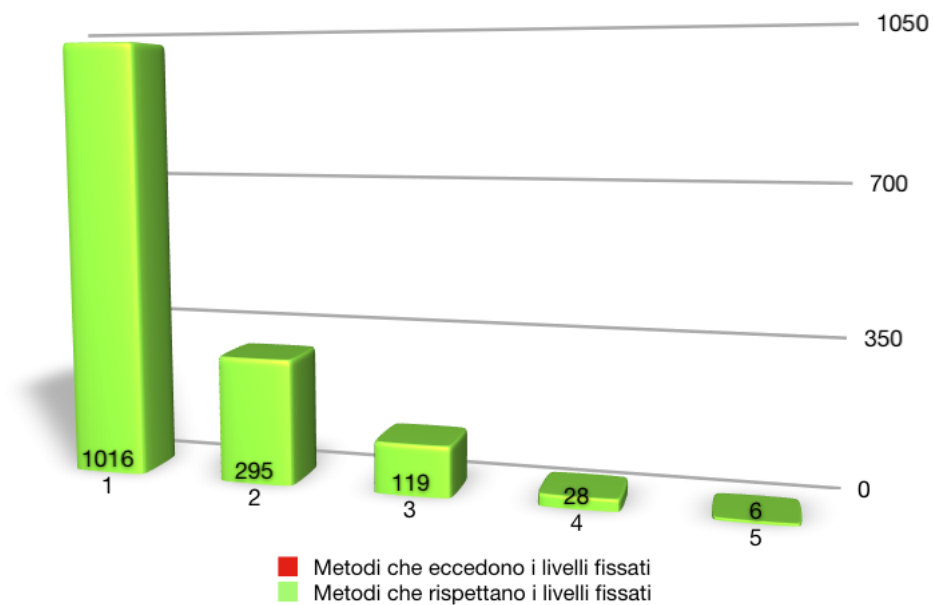


Figura 13: Annidamento del codice RescueMe

Come si può osservare dalla figura precedente il livello di annidamento del codice di RescueMe rientra nei limiti fissati nelle norme di progetto raggiungendo il livello massimo di 5. La maggior parte del codice si trova a livello di annidamento pari a 1.

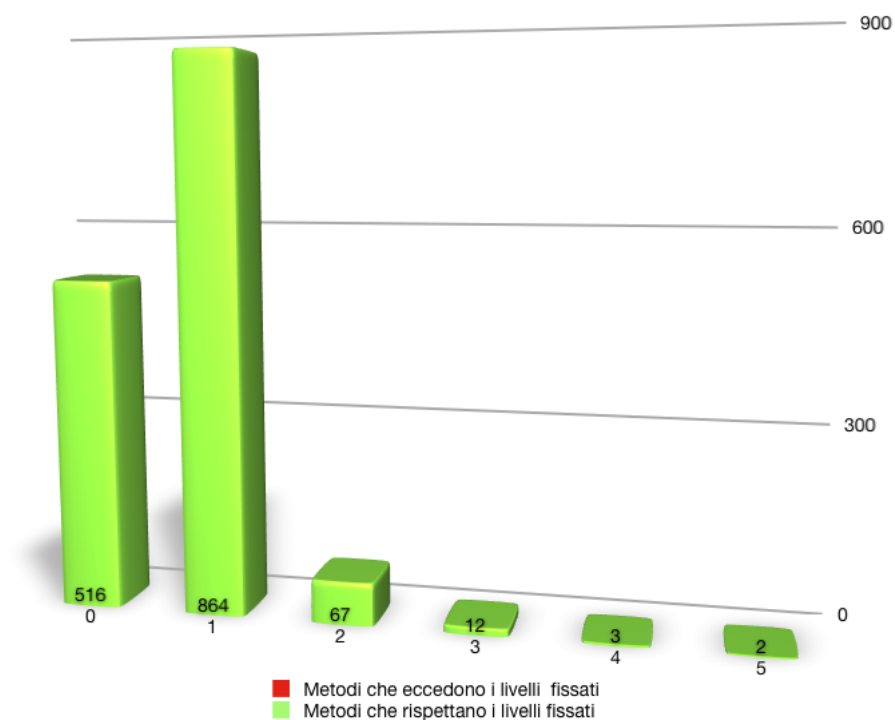
Numero di parametri per metodo

Figura 14: Numero di parametri per metodo RescueMe

Come si può osservare dalla figura precedente il numero di parametri per metodo di RescueMe rientra nei limiti fissati nelle norme di progetto raggiungendo il livello massimo di 5. La maggior parte dei metodi hanno un numero di parametri pari a 0 o a 1.

B.4.2 Metriche RescueApp

Osserviamo ora le misurazioni effettuate sull'applicazione RescueApp, i tre package `clientandroid.gui`, `clientandroid.operations` e `clientandroid.information` rispettano le metriche stabilite nelle norme.

Complessità ciclomatica

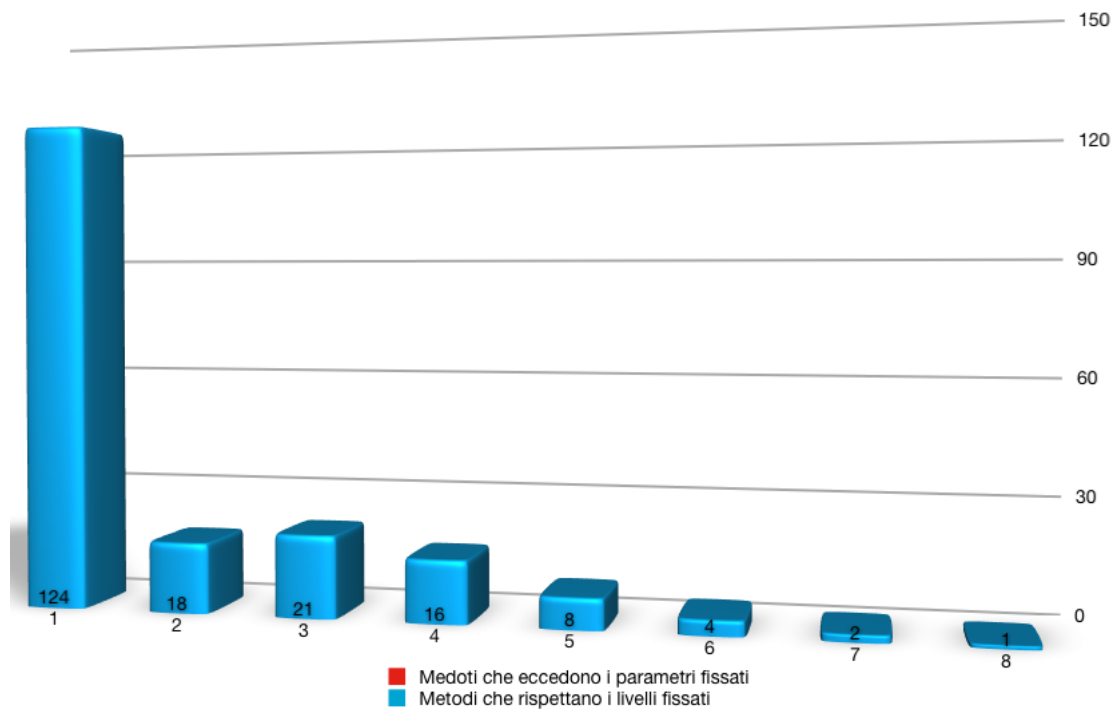


Figura 15: Complessità ciclomatica RescueApp

Come si può osservare dalla figura precedente, la complessità ciclomatica raggiunge un valore massimo di otto, rispetta quindi le norme di progetto. La maggior parte dei metodi risulta avere complessità pari a 1.

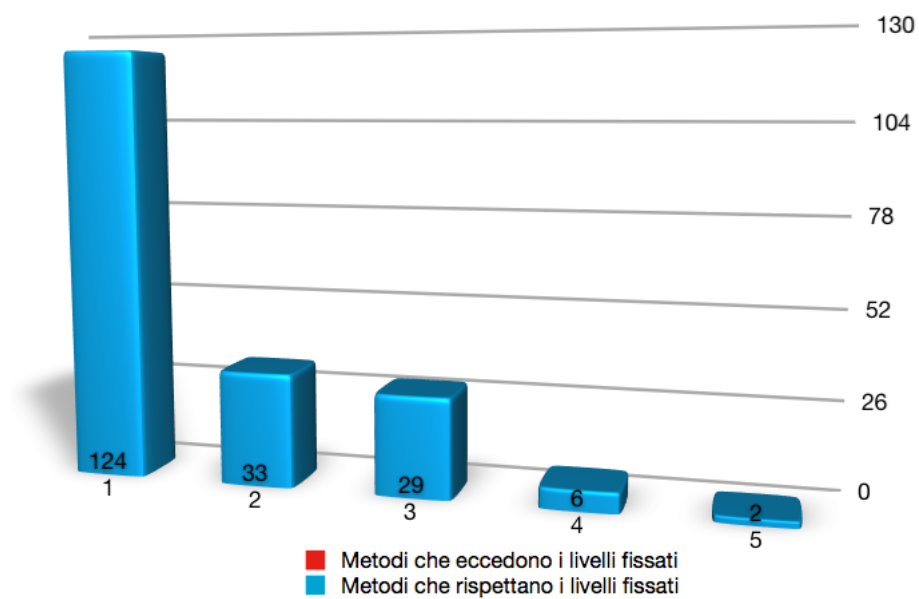
Annidamento del codice

Figura 16: Annidamento del codice RescueApp

Come si può osservare dalla figura precedente, il codice raggiunge un valore massimo di annidamento di cinque, rispetta quindi le norme di progetto pur raggiungendo il valore di soglia. La maggior parte del codice risulta avere complessità pari a 1.

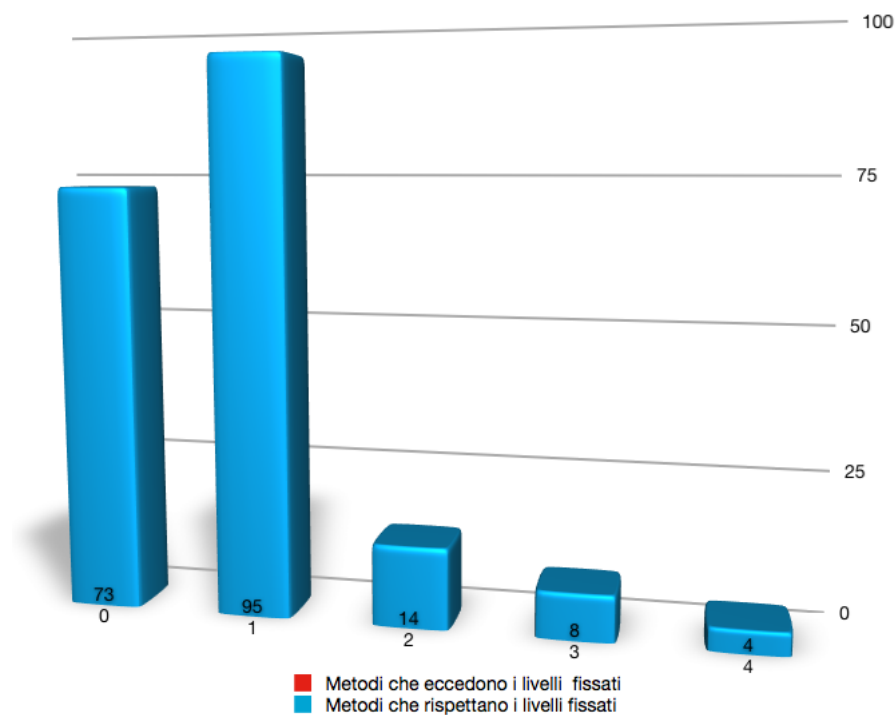
Numero di parametri per metodo

Figura 17: Numero di parametri per metodo RescueApp

Come si può osservare dalla figura precedente, i metodi raggiungono un valore massimo di parametri passati di 4, rispetta quindi le norme di progetto. La maggior parte dei metodi hanno o un solo parametro, o nessun parametro in ingresso.