wheelsoft

Piano di Qualifica

14 Marzo 2007

Sommario

Il documento identifica e dettaglia la strategia di verifica e validazione proposta per il collaudo del prodotto, incluse le eventuali prove con le azioni corrispondenti in ingresso e risultati attesi. Documento redatto secondo le Norme di Progetto v1.9.

Informazioni documento

Produzione	WheelSoft - wheelsoft@gmail.com
Redazione	Alberto De Bortoli - adeborto@studenti.math.unipd.it
	Matteo Borgato - mborgato@studenti.math.unipd.it
Approvazione	Stefano Ceschi Berrini - sceschib@studenti.math.unipd.it
	Alessio Rambaldi - arambald@studenti.math.unipd.it
	Silvio Daminato - sdaminat@studenti.math.unipd.it
	Matteo Borgato - mborgato@studenti.math.unipd.it
	Giulio Favotto - gfavotto@studenti.math.unipd.it
File	Piano_di_Qualifica_v2.3.pdf
Versione	2.3
Stato	Formale
Uso	Esterno
Distribuzione	Wheelsoft
	prof. Alessandro Sperduti
	prof. Renato Conte
	prof. Tullio Vardanega

Diario delle modifiche

2.3	14/03/07 - Correzioni varie, aggiunta informazioni sul funzionamento di SimpleTest e WebTester (G. Favotto);
2.2	11/03/07 - Aggiornamento delle sezioni *Logic Layer* e *Presentation Layer* con i dettagli dei test ed il resoconto delle metriche adottate, aggiornamento della sezione *Attività di verifica finali* (A. De Bortoli)
2.1	06/03/07 - Inserimento nella sezione *Resoconto delle attività di verifica*, dei resoconti dei test effettuati per il *Data Layer* e per il *Logic Layer* (A. De Bortoli, G. Favotto)
2.0	02/03/07 - Inizio stesura *Resoconto delle attività di verifica* (A. De Bortoli)
1.9	20/02/07 - Inserimento nella sezione *Attività di verifica sul codice* delle sezioni *Test di unità*, *Test d'integrazione* e *Metriche del prodotto* (G. Favotto)
1.8	18/02/07 - Riorganizzazione della struttura del documento con la ridenominazione della sezione *Gestione amministrativa della revisione* in *Gestione delle attività di verifca* e relativo aggiornamento, inserimento delle sezioni *Attività di verifica sulla documentazione* come ampliamento della sezione *Resoconto delle attività di verifica sui documenti* e della sezione *Attività di verifica sul codice* in cui vengono definiti i criteri di verifica da attuare suddivisi in tipologie e il resconto delle verifiche già effettuate (G. Favotto)
1.7	01/02/07 - Correzione imprecisioni (S. Daminato)
1.6	28/01/07 - Si è cercato di spiegare la strategia di verifica e validazione che Wheelsoft intende utilizzare per il progetto *SIAGAS*. Creata la tabella di criticità del software e spiegata inoltre la pianificazione del collaudo. (S. Ceschi Berrini)
1.5	23/01/07 - Modificate le sezioni di Tecniche di Analisi, Metodi di verifica e Resoconto delle attività di verifica. (S. Ceschi Berrini)
1.4	20/01/07 - Aggiornato documento in base al nuovo organigramma (S. Ceschi Berrini)
1.3	16/01/07 - Modifica della struttura del documento, aggiunte nuove sezioni, correzione errori del documento consegnato per la RR. (S. Ceschi Berrini)
1.2	11/12/06 - correzioni varie e approvazione
1.1	07/12/06 - prima stesura del documento

Indice

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è pianificare il processo relativo alla garanzia di qualità. Il processo relativo alla garanzia di qualità (Quality Assurance Process) deve assicurare che i prodotti e i processi relativi al ciclo di vita adottato siano conformi ai requisiti specificati e aderiscano ai piani prestabiliti. Il Piano di Qualifica dovrà inoltre identificare e dettagliare appropriatamente la strategia di verifica e validazione proposta dal fornitore per il collaudo del prodotto, incluse le eventuali prove, con le rispettive precondizioni e postcondizioni. Deve essere verificata la congruenza e la soddisfacibilità dei requisiti presenti nel documento di "Analisi dei Requisiti v2.0".

1.2 Scopo del prodotto

Per lo scopo del prodotto si faccia riferimento al documento "Analisi dei Requisiti v2.0".

1.3 Glossario

Il glossario è presente in un documento PDF chiamato "Glossario_v1.8.pdf", allegato alla documentazione per la Revisione dei Requisiti, che definisce e spiega i termini tecnici utilizzati nei documenti ufficiali al fine di eliminare ogni ambiguità relativa al linguaggio.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato "Sportello Informatico per Attivazione e Gestione di Attività di Stage", emesso dal committente prof. Alessandro Sperduti in data 07/11/06 e reperibile all'indirizzo:

  www.math.unipd.it/∼tullio/IS-1/2006/Progetti/SIAGAS.html

- Analisi dei Requisiti
- Norme di Progetto
- IEEE/EIA 12207.0-1996 - Software Life Cycle Processes
- IEEE Std 610.12-1990 - Standard Glossary of Software Engineering Terminology
- IEEE Std 730TM-2002 (revision of IEEE Std 730-1998) - Standard for Software Quality Assurance Plans
- IEEE Std 1012-1998 - Standard for Software Verification and Validation

1.4.2 Informativi

- Frederick P. Brooks: No Silver Bullet: Essence and Accidents of Software Engineering
- Guide to the SWEBOK (www.swebok.org)

2 Visione generale della strategia di verifica

2.1 Organizzazione, pianificazione strategica e temporale, responsabilità

Si assume che un processo ben pianificato ed amministrato porti al raggiungimento di un alto livello di qualità del prodotto. Le attività di verifica, necessarie per giungere al collaudo del sistema, saranno definite dal responsabile, il quale le suddividerà in moduli che verranno assegnati ai singoli verificatori; la pianificazione deve includere la totalità dei moduli che costituiscono il progetto e rispettare i tempi definiti dal responsabile. L'organizzazione delle strategie di verifica sarà principalmente basata sull'Analisi dei Requisiti. In questo modo, si otterrà la tracciabilità finalizzata a collegare i requisiti richiesti con le componenti e i relativi metodi che saranno individuati in dettaglio. Come detto in precedenza, responsabili dell'attività di verifica saranno i Verificatori ed il Responsabile di progetto.

Il processo di verifica deve essere istanziato ogni qual volta il prodotto di un processo subisca sostanziali cambiamenti. Tali cambiamenti devono essere necessariamente notificati nel registro delle modifiche relativo a quel prodotto. La segnalazione e la correzione di eventuali problemi o anomalie riscontrati durante processo di verifica dovranno essere trattati dal processo di risoluzione dei problemi.

La verifica inoltre si occuperà di assicurarsi che la definizione del requisito sia fatta in maniera non ambigua e la relativa trasposizione del componente ricopra tutte le caratteristiche specificate nel requisito. Sarà importante quindi il tracciamento, nel quale si associa ad ogni requisito uno o più componenti.

L'attività di verifica dovrà essere attuata in maniera costante ed al passo con l'avanzamento del prodotto, in modo da minimizzare i rischi di fallimento e/o discrepanze più o meno gravi.

Si è deciso di adottare un modello di ciclo di vita incrementale. La tipologia del sistema da sviluppare e la struttura dei processi di revisione si prestano bene ad un approccio che prevede necessità di adattamento ai cambiamenti: evoluzione dei problemi e tecnologie utilizzabili.

I processi di revisione adottati saranno di due tipi:

- revisione formale:
  - revisione dei requisiti (**RR**)
  - revisione di accettazione (**RA**)
- revisione informale:
  - revisione del progetto preliminare (**RPP**)
  - revisione del progetto definitivo (**RPD**)
  - revisione di qualifica (**RQ**)

2.2 Dettaglio delle revisioni

2.2.1 Revisione dei Requisiti

- **Prodotti in ingresso**: Capitolato d'Appalto, Analisi dei Requisiti, Piano di Progetto
- **Funzione**: accordarsi con il committente (cliente) per una descrizione il più possibile esaustiva del prodotto software atteso
- **Stato di uscita**: prodotto descritto

2.2.2 Revisione del Progetto Preliminare

- **Prodotti in ingresso**: Specifica Tecnica, aggiornamento del Piano di Qualifica
- **Funzione**: accertamento di realizzabilità, attivare la fase di realizzazione del prodotto
- **Stato di uscita**: prodotto specificato

2.2.3 Revisione del Progetto Definitivo

- **Prodotti in ingresso**: Definizione di Prodotto, aggiornamento del Piano di Qualifica
- **Funzione**: informare il committente riguardo le caratteristiche definitive del prodotto, attivare la fase di qualifica
- **Stato di uscita**: prodotto definito

2.2.4 Revisione di Qualifica

- **Prodotti in ingresso**: aggiornamento del Piano di Qualifica, inclusione della specifica delle prove di accettazione, versione preliminare del Manuale d'Uso
- **Funzione**: approvazione della campagna di verifica; attivazione della fase di accettazione
- **Stato di uscita**: prodotto qualificato

2.2.5 Revisione di Accettazione

- **Prodotti in ingresso**: versione definitiva del Piano di Qualifica, inclusione dell'esito delle prove di accettazione, versione definitiva del Manuale d'Uso
- **Funzione**: collaudo del sistema per Accettazione del prodotto da parte del committente
- **Stato di uscita**: prodotto accettato

2.3 Risorse necessarie e disponibili

La verifica dei moduli che compongono il prodotto *SIAGAS* impegna risorse umane e tecnologiche. L'amministratore di progetto eseguirà un'attività di supervisione e coordinamento di tali risorse durante la verifica. Questa procedura è necessaria per accertarsi che le risorse siano sufficienti ed efficienti per consentire una verifica di buon livello.

2.3.1 Risorse umane

Attualmente il gruppo prevede la seguente suddivisione dei ruoli:

- **1 Responsabile**
- **1 Amministratore**
- **3 Progettisti**
- **2 Verificatore**

Si faccia riferimento al piano di progetto per vedere la suddivisione dei ruoli nel tempo.

I verificatori avranno un ruolo chiave per il processo di qualità del prodotto in quanto dovranno cercare di scoprire errori ed anomalie sia nei documenti che nel codice. Per far questo dovranno controllare sintassi e semantica dei documenti e dovranno fare dei test per poter verificare il codice.

2.3.2 Risorse tecnologiche

Il gruppo dispone e si avvarrà delle seguenti risorse tecnologiche:

- **Un Server Web Apache**: per testare il progetto complessivo;
- LaTeX: per la stesura/correzione dei documenti;
- **Strumenti di validazione forniti dal W3C**: per aderire a standard affidabili;
- **Diversi tipi di Browser**: per estendere il dominio di utilizzo;
- **Editor testuali per le pagine php ed html**: per la stesura/correzione della *parte web*;
- **MySql**: per testare il corretto funzionamento del *SIAGASDB*.

2.4 Tecniche di Analisi

- **Analisi Statica**: viene svolta in maniera contestuale alla stesura del codice a partire dai requisiti di sistema. È necessaria al fine di individuare eventuali situazioni in cui il codice sia ridondante, la complessità non sia appropriata e vi siano problematiche riguardanti la progettazione "a tavolino". Bisogna esplicitare e analizzare le dipendenze tra ingressi e uscite che caratterizzano le unità di codice testate, e provare la correttezza del codice sorgente rispetto ai requisiti prefissati.

- **Analisi Dinamica**: verifica dinamicamente il comportamento di ogni singola unità di codice, sia indipendentemente (ogni singolo componente) sia nell'insieme (sul sistema). Deve cominciare prima della fine della fase di codifica e le sue esigenze devono essere considerate anche in fase di progettazione.

  Il progettista ne definirà le strategie di applicazione:

  - il Test di Unità:
    - *funzionale*: ha lo scopo di verificare ogni unità affinché soddisfi i requisiti previsti mediante l'impiego di un insieme di dati in ingresso che siano in grado di generare risultati attesi
    - *strutturale*: verifica la logica interna del codice, cioè sono effettuati controlli affinché ciascuna unità vada ad attivare ogni cammino di esecuzione possibile
  - Test di Integrazione: è applicato alle componenti e si serve della logica di integrazione funzionale per selezionare le funzioni da integrare, per ordinare le componenti per numero di dipendenze e per eseguire l'integrazione a partire dalle componenti con precedenza. Lo scopo finale è quello di individuare difetti di progettazione o carenze in fase di verifica. L'integrazione tra componenti avviene in modo incrementale e assemblando per primi i moduli che forniscono flusso di controllo e flusso dei dati, garantendo sempre un'integrazione che sia reversibile.
  - Test di Sistema: verifica del sistema per accertare che siano rispettati i requisiti iniziali.
  - Test di Accettazione: collaudo del sistema effettuato in presenza del committente.

2.5 Metodi di verifica

Le revisioni formali ed informali saranno lo strumento principale per la verifica. Queste serviranno per esaminare e convalidare documentazione, requisiti, codice, diagrammi e tecniche di verifica.

Le tecniche di verifica si divideranno in Analisi statica, Analisi dinamica e Tracciamento(atto a dimostrare completezza ed economicità dell'implementazione).

L'Analisi statica, come descritto dagli standard di certificazione, viene applicata combinando i seguenti metodi metodi:

- Flusso di controllo
- Flusso dei dati
- Flusso dell'informazione
- Esecuzione simbolica
- Verifica formale del codice
- Verifica di limite

- Uso dello stack
- Comportamento temporale
- Interferenza
- Codice oggetto

Inoltre la verifica dovrà includere attività come:

- Inspection : per rilevare la presenza di difetti tramite la lettura mirata del codice. Quest'attività è suddivisa in più fasi (pianificazione, definizione della lista di controllo, lettura del codice, correzione dei difetti). In ogni fase dovrà essere documentata l'attività svolta;
- Walkthrough : per rilevare la presenza di difetti tramite la lettura critica del codice. Quest'attività è suddivisa in più fasi (pianificazione, lettura del codice, discussione, correzione dei difetti). In ogni fase dovrà essere documentata l'attività svolta;

Wheelsoft si impegnerà a fare un'Analisi di questo tipo utilizzando i metodi necessari e più efficienti rispetto alle esigenze:

- Per la documentazione verrà fatta analisi sintattica e semantica del testo, oltre che controllo di possibili ambiguità nel linguaggio ed errori logici in schemi/modelli/diagrammi.
- Per quanto riguarda il codice, verranno combinati metodi di Analisi statica(come descritto sopra).

Inoltre, per fare test sui componenti singoli e sul sistema generale, verrà svolta Analisi dinamica utilizzando anche driver e stub.

Tutte le fasi dell'attività di verifica dovranno essere documentate (attività svolte, risultati e anomalie).

3 Gestione delle attività di verifica

3.1 Comunicazione e risoluzione di anomalie

Il Verificatore, per ogni modulo verificato, dovrà redigere un documento ad uso interno nel quale saranno specificati:

- modulo in analisi;
- stato del modulo;
- dati in ingresso;
- dati in uscita attesi;
- dati in uscita rilevati;
- esito del test ed eventuali anomalie riscontrate;

Nelle norme di progetto viene definita un modello di struttura per la relazione di un test case e l'inserimento di eventuali anomalie nel software di supporto.

Per ogni anomalia riscontrata si dovranno accertare le:

- possibili cause
- possibili soluzioni

Il Responsabile provvederà all'approvazione del documento e lo inoltrerà all'Amministratore, indicando il riferimento alla segnalazione di anomalia attivata. Verificatori e Programmatori designati raccoglieranno la segnalazione per provvedere alla sua gestione.

3.2 Procedure di controllo di qualità di processo

L'Amministratore ha a suo carico il controllo della gestione dell'attività di verifica, dell'emissione e distribuzione dei documenti durante i test e della gestione e risoluzione delle anomalie.

Ogni processo dovrà essere descritto in maniera dettagliata e non ambigua, in modo che sia semplice da controllare.

Dovranno essere indicati anche i livelli di criticità.

3.2.1 Livelli di criticità

Wheelsoft cercherà di attenersi allo standard IEEE 730-2002. La seguente tabella specifica le priorità rispetto la criticità del software:

Grado	Componente	Danni al prodotto
3	componente fondamentale rispetto alle funzionalità del sistema	prodotto inutilizzabile
2	componente importante rispetto alle funzionalità del sistema	importanti disagi causati al prodotto
1	componente influisce su funzionalità rilevanti del sistema	prodotto utilizzabile ed anomalia osservabile
0	componente influisce su funzionalità trascurabili	nessun disagio sostanziale causato al prodotto

3.2.2 Metriche di valutazione del processo

In termini di metriche relative ai processi di verifica Wheelsoft si propone di attenersi allo standard IEEE 1012:1998.

Dovranno essere analizzati vari fattori per valutare lo sviluppo e la qualità del prodotto. Questi saranno:

- Complessità del processo;
- Risorse richieste dal processo;
- Tempo richiesto dal processo;
- Eventuale gestione degli errori.

Nel caso in cui la qualità dell'output del processo sia accettabile, si cercherà di ridurre i costi senza diminuire la qualità.

4 Attività di verifica sulla documentazione

4.1 Pianificazione

Non potendo eseguire attività di dinamica sulla documentazione, la verifica sarà caratterizzata principalmente da attività di analisi statica:

- **Adesione di un documento alle norme di progetto**
- **Tracciamento**
  Il tracciamento in questa fase ricopre un ruolo determinante per quanto riguarda la correttezza di un documento, in quanto permette di verificare che un esigenza manifestata dal proponente venga formalmente descritta. Saranno pertanto necessarie le seguenti forme di tracciamento:
  - **Tracciamento requisiti-fonti**
    Il tracciamento requisiti-fonti viene descritto completamente nel documento di "Analisi Dei Requisiti v2.0" utilizzando dei codici di fonte e di requisito univoci in modo da poter associare ad ogni requisito una fonte documentata;
  - **Tracciamento requisiti-componenti**
    Il tracciamento requisiti-componenti viene descritto completamente nel documento "Specifica Tecnica v1.7"utilizzando dei codici di requisito e di componente univoci per poter associare ad ogni requisito uno o più componenti;
  - **Tracciamento requisiti-metodi**
    Il tracciamento requisiti-metodi viene descritto completamente nel documento "Definizione di Prodotto v1.6"utilizzando dei codici metodo univoci per poter associare ad ogni requisito uno o più metodi;

4.2 Resoconto

4.2.1 Verifiche tramite analisi

Verifiche attuate finora:

Revisione	Tecnica
Revisione Requisiti	In questa fase è stata attuata analisi statica secondo i criteri e le modalità previste, redigendo i documenti di verifica richiesti ed avviando le procedure per la correzione degli errori riscontrati
Revisione Progetto preliminare	Anche in questa fase è stata attuata analisi statica secondo i criteri e le modalità previste, redigendo i documenti di verifica richiesti ed avviando le procedure per la correzione degli errori riscontrati. Particolare attenzione è stata posta nella verifica dei diagrammi UML inseriti in cui sono stati riscontrate e corrette varie imperfezioni, dovute principalmente all'inesperienza nella redazione di tali modelli.

4.2.2 Verifiche tramite prove (test)

Al momento non sono state fatte verifiche tramite prove.

4.2.3 Esito delle revisioni

Revisioni fatte finora:

Tipo di revisione	Esito
Revisione dei Requisiti	• Documento "Analisi dei requisiti v2.0": A fini di tracciabilità occorre associare un identificatore alfanumerico a ciascun singolo requisito. Al momento invece i requisiti hanno identificatori numerici a intersezione non vuota. Buona qualità tecnica. Buona l'impostazione anche se l'analisi non è ancora sufficientemente approfondita. Alcuni termini sono generici (requisiti desiderabili: Interfaccia immediata? Efficienza del sistema ? ...). Manca tracciamento requisiti verso le fonti. Manca anche una specifica (almeno preliminare) della strategia di verifica dei requisiti; • Documento "Piano di Progetto": Errori e orrori terminologici; • Documento "Piano di Qualifica": nella versione attuale manca di profondità e di prospettiva strategica.
Revisione Progetto Preliminare	• Documento "Specifica Tecnica v1.6": Fig.3: manca il nome della relazione "stage-docente", la classe "Log" è isolata; • Documento "Piano di Progetto": Ottimo. Sarebbe utile una tabella riassuntiva dei costi preventivati a finire rivisti alla luce del consuntivoattuale; • Documento "Piano di Qualifica": Qualche miglioramento rispetto alla varsione RR. Le sezioni 2.4-5, pur impegnative, sono lacunose perchè non relazionano le intenzioni strategiche all'identificazione dell'infrastruttura di supporto necessaria per realizzarle sul piano tecnologico e metodologico.

5 Attività di verifica sul codice

5.1 Metriche

Al fine di monitorare la qualità del codice e quindi del prodotto finale verranno prese in considerazione le seguenti metriche quantitative.

5.1.1 Numero di parametri

Questa metrica si basa sul conteggio del numero di parametri di un metodo. Metodi con un valore alto di NdP potrebbero dover essere modificati al fine di ottenere un maggiore livello d'astrazione ed un aumento del grado di disaccoppiamemto. L'alto numero di parametri di un metodo, dunque, potrebbe significare una mancanza di coesione nelle funzionalità fornite dal metodo stesso (con conseguenti problemi relativi la manutenibilità).

5.1.2 Numero di livelli

Questa metrica rileva il massimo livello d'annidamento di ciascun metodo. Il livello d'annidamento di un metodo rappresenta la misura in cui il suo codice si ramifica, ed è determinato facendo riferimento al numero di istruzioni che impongono una divisione del flusso. Il NdL si calcola determinando il massimo tra tutti i livelli presenti in un metodo. Si assume che un valore di NdL alto comporti alcune problematiche relative alla comprensione e alla complessità del codice.

L'obiettivo è limitare il numero di livelli massimo a 6.

5.1.3 Complessità ciclomatica

La complessità ciclomatica misura il numero di cammini linearmente indipendenti che attraversano un metodo. Questa misurazione fornisce un singolo valore intero che può essere comparato con la complessità di altri programmi. Tale valore è, tra le altre cose, utilizzato per acquisire due tipi di informazioni. Innanzi tutto esprime il numero minimo di test da effettuare sul software per garantire un'adeguata *branch coverage*. In secondo luogo è utile, a partire dalla progettazione architetturale, per controllare l'affidabilità, la testabilità e l'amministrabilità del software. Ciascun metodo può essere rappresentato con un grafo ad albero. Il valore di complessità ciclomatica relativo a quel determinato metodo è espresso dalla formula che segue.

$$\mathbf{CC = E - N + P}$$

- CC = Complessità Ciclomatica
- E = Numero di archi
- N = Numero di nodi
- P = Numero di Componenti Connesse

Si assume che un alto valore di complessità ciclomatica non denota necessariamente un grosso rischio relativo alla parte di codice presa in esame. Esso

è, piuttosto, il sintomo che il codice in considerazione può non essere ben congegnato dal punto di vista logico (e che, sicuramente, sarà oggetto di un non trascurabile lavoro di prova).
Uno dei problemi che concerne l'analisi ciclomatica riguarda la stima del suo valore in relazione all'istruzione di tipo **switch**. La presenza di uno **switch** all'interno del codice, infatti, aumenta significativamente il valore di complessità ciclomatica. Verranno effettuate delle considerazioni per valutare le possibili trattazioni del problema.
L'obiettivo è portare il codice generato ad avere una complessità ciclomatica inferiore a 10.

5.1.4 Numero delle anomalie

Questa metrica si basa sul conteggio delle anomalie riscontrate. Tale metrica sarà disponibile in maniera aggiornata in ogni momento grazie al supporto fornito dall'applicazione di bugtracking indicata nelle norme. Importante sarà rilevare il momento in cui il numero delle nomalie riscontrate non risulterà più crescere ma sarà costante, in modo da non impiegare ulteriori risorse inutilmente.

5.1.5 Fan-in e Fan-out

Questa metrica dimostra l'utilità o la dipendenza di un componente rispetto ad altri. Verrano rilevati questi valori per i vari componenti del sistema. L'adozione di un software di supporto è in fase di valutazione, ma vista le relativa compattezza del sistema tale metrica potrebbe essere rilevata anche manualmente.

5.2 Pianificazione

La verifica sul codice si avvarrà delle forme di verifica sopra descritte, nelle forme di analisi statica e dinamica.

5.2.1 Analisi statica

Per realizzare l'analisi statica, come previsto dalle norme di progetto, si farà affidamento sulla funzionalità di segnalazione errori di PHP5. E' inoltre in fase di valutazione la possibilità di far affidamento su alcuni tools di recente sviluppo per ricevere ulteriore supporto all'analisi statica su linguaggi di scripting. Verrano comunque effettuate verifiche per rilevare la complessità in termini di annidamento e di branching.

5.2.2 Analisi dinamica

L'analisi dinamica prevede lo svolgimento di test su di un insieme finito di test case. Essendo il test una tipologia di verifica di tipo *negativo* con costi piuttosto elevati, è necessario trovare un compromesso fra il numero minimo di casi di prova necessari per dimostrare il corretto funzionamento di un componente e la quantità di risorse disponibili. Per questo motivo sarà quindi necessario effettuare le rilevazioni metriche descritte nella sezione 5.1 in modo da determinare con più precisione possibile la quantità e la modalità di test da effettuare.

- **Test di unità**

  Sono stati valutati vari strumenti di supporto per realizzare tali test; l'orientamento attuale prevede vengano utilizzati alcuni script java-php facenti parte dei pacchetti JWebUnit (estensione di JUnit) e SimpleTest in modo da verificare sia i componenti del livello base, il data layer, che i componenti logici PHP presenti nel logic layer ed infine i componenti HTML-CSS presenti nel presentation layer.
  Statement coverage e branch coverage saranno effettuati sulla base delle metriche rilevate.
  Segue la descrizione dei test per componente.

  - **Data layer**
    - **Descrizione componente**: Il Data layer contiene tutti i dati relativi al sistema SIAGAS.
    - **Strategia**: Oltre al controllo dell'adesione del documento alle norme di progetto relative alla creazione del Data layer, dovranno essere verificate le condizioni di sicurezza del database, la presenza dei vincoli di unicità delle chiavi, nonchè il corretto funzionamento di alcune query che testino i casi limite. Queste query sono in via di definizione.
  - **pear.MDB2**
    - **Descrizione componente**: *pear.MDB2* è il package che si occupa di eseguire query sul Data layer.
    - **Strategia**: Non verranno effettuati test strutturali in quanto MDB2 è un package esterno fornito come stabile.
  - **attori.Utente**
    - **Descrizione componente**: Questa componente è una classe astratta. Contiene solo metodi, comuni alle sottoclassi, senza parametri, che ritornano sempre lo stesso output (se lo stato del Data layer rimane invariato).
    - **Strategia**: Si dovrà utilizzare una qualsiasi sottoclasse derivata non astratta per testare i metodi tramite un *driver* che richiamerà i metodi definiti in *Utente*.
  - **attori.UtenteAutenticato**
    - **Descrizione componente**: Questa componente è una classe astratta. Contiene solo metodi, comuni alle sottoclassi, senza parametri, che ritornano sempre lo stesso output (se lo stato del Data layer rimane invariato).
    - **Tecnica**: Si dovrà utilizzare una qualsiasi sottoclasse derivata non astratta per testare i metodi tramite un *driver* che richiamerà i metodi definiti in *Utente*.
  - **attori.UtenteNonAutenticato**
    - **Descrizione componente**: Questa componente raccoglie i metodi per un utente non autenticato. I metodi di questa componente contengono costrutti di controllo per svolgere funzioni di accesso e di registrazione attraverso query sul Data layer.

    * **Strategia**: L'obiettivo è quello di ottenere un *branch coverage* dei costrutti di controllo realizzando un driver tramite il quale poter fornire tutti i valori significativi, accertando che i metodi si comportino come dovrebbero. Test funzionali con *valori legali* ed *illegali* verranno pertanto effettuati per ogni metodo con appositi script, mentre i test strutturali saranno effettuati già in fase di scrittura e se necessario saranno approfonditi al rilevamento di valori anomali delle metriche previste.

- **attori.Studente, attori.Amministratore, attori.Docente, attori.Proponente, attori.Super_utente**
    * **Descrizione componenti**: Queste componenti si comportano similmente in quanto rappresentano gli utenti del sistema, pertanto le tecniche di verifica saranno comuni.
    * **Strategia**: Test funzionali con *valori legali* ed *illegali* verranno pertanto effettuati per ogni metodo con appositi script, mentre i test strutturali saranno effettuati già in fase di scrittura e se necessario saranno approfonditi al rilevamento di valori anomali delle metriche previste.
- **attori.Sistema**
    * **Descrizione componente**: *Sistema* è la classe principale dell'applicazione; è realizzata come un singleton con i metodi che permettono ad esempio la generazione di documenti, accesso al Data layer (attraverso MDB2), l'autenticazione etc... Si presterà attenzione alla sicurezza nei metodi di questa classe, salvando i documenti creati in una cartella sicura e non permettendo l'esecuzione di query pericolose o l'inserimento di dati pericolosi (come codice javascript).
    * **Strategia**: È necessario assicurare che il singleton torni effettivamente un'istanza di sè stesso ad un'ipotetico chiamante. Verrà dunque realizzato uno *stub* tramite il quale ottenere l'istanza.
      In particolar modo si dovrà accertare che le richieste successive alla prima non creino una nuova istanza dell'oggetto in questione.
      Test funzionali con *valori legali* ed *illegali* verranno effettuati per ogni metodo con appositi script, mentre i test strutturali saranno effettuati già in fase di scrittura e se necessario saranno approfonditi al rilevamento di valori anomali delle metriche previste. La generazione dei documenti verrà testata manualmente. Gli automatismi verrano testati nella fase successiva.
- **Presentation layer**
    * **Descrizione componente**: Il presentation layer fornisce l'interfaccia grafica all'utente del sistema eseguendo le funzioni rese disponibili dal livello sottostante. E' pertanto composto da svariate componenti la cui conoscenza singola non risulta importante in quanto esse risultano omogenee anche in termini di esigenze di verifica.
    * **Strategia**: Verrà verificato che ogni pagina creata rispetti le norme di progetto esistenti, quindi le validazioni W3C in termini

di CSS, XHTML e WAI-AAA dovranno essere superate positivamente. Attraverso i tool selezionati per il controllo del codice html sarà verificato non siano presenti link scorretti o errori sintattici. I test funzionali e strutturali sul codice php saranno effettuati con appositi script utilizzando *valori legali* e *valori illegali*. Ogni cammino dovrà risultare coperto.

- **Test d'integrazione**

  La scelta architetturale relativa all'utilizzo di una struttura a tre strati promuove la scalabilità del sistema ed offre la possibilità di una facile sostituzione dei livelli interoperanti. Il passo successivo all'aver effettuato i test di unità è di, partendo dal basso (cioè dal data layer), sostituire ciascuno strato con *driver* creati ad hoc. Tale tipo di approccio è di carattere incrementale e s'avvicina alla strategia di tipo bottom-up.

  Vengono descritte ora le strategie di integrazione delle componenti:

  - **Integrazione Data layer, pear.MDB2**
    - **Descrizione contestuale**: MDB2 è un package che permette di connettersi ad un database astrando dal tipo di database per maggiore portabilità. MDB2 usa metodi per connettersi e per eseguire query.
    - **Strategia**: Saranno necessari test funzionali per verificare i casi estremi da eseguire con appositi script; test strutturali, essendo PEAR un package esterno, non saranno eseguiti.
    - **Aspettazione**: Si dovranno riscontrare gli esiti attesi
  - **Integrazione *attori.Sistema*, pear.MDB2**
    - **Descrizione contestuale**: *Sistema* è la classe che più interagisce con pear.MDB2, esegue sia query di interrogazione che di modifica. Le altre classi che eseguono query usano i metodi di *Sistema*, i quali ritornano l'informazione così come ritornata da pear.MDB2 a *Sistema*. Non ci sarà quindi necessità di eseguire test anche sull'intergrazione, per quanto riguarda l'esecuzione di query, tra le altre classi del package attori e la classe *Sistema* stessa.
    - **Strategia**: Su una tabella di prova verranno effettuate un numero di query-test sufficiente a garantire il buon funzionamento dei metodi (almeno una query per ogni categoria, dove per categoria s'intende *select*, *insert*, *update* e *delete*).
    - **Aspettazione**: La query richiesta viene effettuata senza errori con l'effetto desiderato.
  - **Integrazione Presentation layer**, *attori.UtenteNonAutenticato*
    - **Descrizione contestuale**: Per permettere agli utenti di autenticarsi nel sistema SIAGAS il presentation layer necessita di un'istanza di *Amministratore*, *Studente*, *Docente* o *Proponente*. Per ottenere quest'istanza il presentation layer invoca un metodo

su *UtenteNonAutenticato*. Il metodo avrà al suo interno dei costrutti di controllo che, in base ai parametri passati, torneranno al chiamante la corretta istanza dell'utente.

  - **Strategia**: Dovranno essere effettuati almeno tanti test quanti sono i possibili tipi di istanze che il metodo ritorna.
  - **Aspettazione**: Il metodo che ritorna l'istanza agisce correttamente.

- **Integrazione *attori.Sistema*, *attori***
  - **Componenti integrate**: *attori.UtenteNonAutenticato*, *attori.Studente*, *attori.Amministratore*, *attori.Proponente*, *attori.Docente*, *attori.Sistema*
  - **Descrizione contestuale**: Per poter accedere al database e per poter eseguire altre azioni importanti, le classi del package *attori* usano il singleton *Sistema*. Devono quindi poter ottenere un' istanza di *Sistema* usando un metodo statico.
  - **Strategia**: Si effettuerà una banale prova di funzionamento del metodo statico per accertarsi che la classe *Sistema* ritorni un'istanza di sé stessa.
  - **Aspettazione**: Il metodo statico che restituisce l'istanza funziona correttamente.

- **Integrazione *Presentation layer*, *attori***
  - **Descrizione contestuale**: Il presentation layer, per poter eseguire interrogazioni e per azioni quali generazione di password, generazione di notifiche ed allerte, deve usare metodi delle classi del package attori.
  - **Tecnica**: Si effettueranno test di funzionamento di questi metodi, con copertura completa dei cammini.
  - **Aspettazione**: I metodi si comportano secondo la definizione fatta.

5.3 Resoconto

Viene fornito il resoconto effettuato per i 3 livelli dell'architettura del sistema.

5.3.1 Presentation Layer

Analisi Statica

Il codice XHTML è stato scritto tramite Dreamweaver, editor fornito in versione di prova. Tramite esso è stato possibile effettuare in maniera continua il controllo di validazione e della sintassi.

Analisi Dinamica

Come pianificato, sono stati svolti test di validazione su tutte le pagine navigabili del sistema tramite lo strumento di validazione online del W3C reperibile all'indirizzo:

http://validator.w3.org/

Lo stesso processo di validazione è stato effettuato per i fogli di stile utilizzati tramite lo strumento reperibile all'indirizzo:

http://jigsaw.w3.org/css-validator/

Obiettivo del test	**Descrizione**	**Ingressi**	**Esito**
accertare l'adesione allo standard XHTML 1.1	utilizzo dello strumento di validazione W3C	tutte le pagine XHTML	negativo
accertare l'adesione allo standard CSS 2.0	utilizzo dello strumento di validazione W3C	fogli di stile CSS	positivo

Gli esiti si sono dimostrati positivi per quanto riguarda la validazione CSS, negativi per la validazione XHTML. Questi ultimi sono da intendersi come provvisori; i motivi della non validazione sono però noti e gli errori sono facilmente removibili; trattandosi del Presentation Layer (livello più alto dell'architettura) la correzione verrà effettuata come ultimo dettaglio della realizzazione.

Web Tester

Con l'utilizzo pianificato di Web Tester, funzionallità di Simpletest, strumento di supporto per la realizzazione di test, è stato poi realizzato il codice per la realizzazione di ulteriori tests. Esso mette a disposizione i seguenti metodi:

assertTitle($title)	Pass if title is an exact match
assertText($text)	Pass if matches visible and alt text
assertNoText($text)	Pass if doesn't match visible and alt text
assertPattern($pattern)	A Perl pattern match against the page content
assertNoPattern($pattern)	A Perl pattern match to not find content
assertLink($label)	Pass if a link with this text is present
assertNoLink($label)	Pass if no link with this text is present
assertLinkById($id)	Pass if a link with this id attribute is present
assertNoLinkById($id)	Pass if no link with this id attribute is present
assertField($name, $value)	Pass if an input tag with this name has this value
assertFieldById($id, $value)	Pass if an input tag with this id has this value
assertResponse($codes)	Pass if HTTP response matches this list
assertMime($types)	Pass if MIME type is in this list
assertAuthentication($protocol)	Pass if the current challenge is this protocol
assertNoAuthentication()	Pass if there is no current challenge
assertRealm($name)	Pass if the current challenge realm matches
assertHeader($header, $content)	Pass if a header was fetched matching this value
assertNoHeader($header)	Pass if a header was not fetched
assertCookie($name, $value)	Pass if there is currently a matching cookie
assertNoCookie($name)	Pass if there is currently no cookie of this name
getUrl()	The current location
getUrl()	The current location
get($url, $parameters)	Send a GET request with these parameters
post($url, $parameters)	Send a POST request with these parameters
head($url, $parameters)	Send a HEAD request without replacing the page content
retry()	Reload the last request
back()	Like the browser back button
forward()	Like the browser forward button
authenticate($name, $password)	Retry after a challenge
restart()	Restarts the browser as if a new session
getCookie($name)	Gets the cookie value for the current context
ageCookies($interval)	Ages current cookies prior to a restart
clearFrameFocus()	Go back to treating all frames as one page
clickSubmit($label)	Click the first button with this label
clickSubmitByName($name)	Click the button with this name attribute
clickSubmitById($id)	Click the button with this ID attribute
clickImage($label, $x, $y)	Click an input tag of type image by title or alt text
clickImageByName($name, $x, $y)	Click an input tag of type image by name
clickImageById($id, $x, $y)	Click an input tag of type image by ID attribute
submitFormById($id)	Submit a form without the submit value
clickLink($label, $index)	Click an anchor by the visible label text
clickLinkById($id)	Click an anchor by the ID attribute
getFrameFocus()	The name of the currently selected frame
setFrameFocusByIndex($choice)	Focus on a frame counting from 1
setFrameFocus($name)	Focus on a frame by name

Si mostra un semplice esempio di test effettuato con le funzionalità Web Tester

di Simpletest. I test da noi effettuati tramite i metodi di asserzione sopra elencati, permettono il rintracciamento di anomalie come la presenza di dead links, il caricamento di una pagina errata, l'assenza del titolo, l'errato contenuto della pagina, etc... Non si è ritenuto necessario effettuare altri tipi di test più specifici.

```
<?php
class TestOfSIAGASLinks extends WebTestCase {
  // percorso di base dal quale iniziare i tests
  public static $_base = 'http://wheelsoft.org/SIAGAS/';
  // si testa il corretto raggiungimento
  // delle pagine collegate alla home page del sito

  function testLinksHome(){
    $this->assertTrue($this->get(TestOfSIAGASLinks::
        $_base));

    $this->clickLink('Info Stage');
    $this->showRequest();
    if ($this->getUrl() != TestOfSIAGASLinks::$_base){
      $this->assertResponse(200);
      $this->assertText('tutor interno');
      }

    $this->clickLink('Documentazione');
    if ($this->getUrl() != TestOfSIAGASLinks::$_base){
      $this->assertResponse(200);
      $this->assertText('Visualizza il documento in
          formato');
      }

    $this->clickLink('Vetrina');
    if ($this->getUrl() != TestOfSIAGASLinks::$_base){
      $this->assertResponse(200);
       $this->assertText('Pagina n');
      }

  }

  function testHomepage() {
    $this->assertTrue($this->get(TestOfSIAGASLinks::
        $_base));
    //$this->showRequest();
    // si testa il corretto raggiungimento della home
        page del sito
    if ($this->assertResponse(200)){
    // si testa che la pagina ottenuta sia quella attesa
```

```
        $this->assertText('Benvenuto nel Sistema');
        // si testa che il titolo della pagina sia quello
            atteso
        $this->assertTitle(new PatternExpectation('/SIAGAS/')
            );
        TestOfSIAGASLinks::$_base = $this->getUrl();
        }
    }

}
?>
```

Web Tester mostra l'output dei test su browser come pagina HTML. Visivamente i risultati si presentano nella seguente maniera:

```
SIAGAS Presentation tests
-------------------------

GET /SIAGAS/Codice/www/index.php HTTP/1.0
Host: localhost
Connection: close
Cookie: PHPSESSID=246705ac7e647b817f0a784a93cf3266

OK
Test cases run: 1/1
    Passes: 15;
    Failures: 0;
    Exceptions: 0
```

Visualizzazione sui browser

Il sito è stato inoltre testato su 5 browser quali: Mozilla Firefox, Internet Explorer (versioni 6 o 7), Safari, Opera e Lynx (browser testuale). Tutti hanno visualizzato correttamente il sito. È stata riscontrata una sola eccezione per quanto riguarda Internet Explorer 6: la parte inferiore di ogni pagina causa una piccola imprecisione a livello di resa grafica causata dal fatto che Internet Explorer non supporta una determinata proprietà descritta nel foglio di stile. Sebbene il problema sia di rilevanza trascurabile è stato risolto.

Generazione documenti pdf

La corretta generazione dei documenti tramite FPDF è stata testata manualmente. Il formato dei documenti creati è PDF v1.3.

5.3.2 Logic Layer

Analisi Statica

L'analisi statica è stata effettuata in 2 momenti:

- durante la realizzazione tramite l'editor utilizzato verificando costantemente la correttenzza della sintassi;
- durante la fase di verifica attraverso il controllo dell'adesione del codice alle norme di progetto definite;

Analisi Dinamica

L'analisi dinamica è stata effettuata in 2 momenti:

- durante la realizzazione molti test con possibili valori attesi sono stati effettuati in fase di realizzazione, in maniera però non sempre sistematica, portando quindi all'immediata correzione delle anomalie riscontrate, dovute in particolare ad errori di sintassi o legate all'inesperienza nell'uso di librerie esterne;
- durante la fase di verifica attraverso la realizzazione di test come definito in fase di pianificazione; rispettando quindi le norme di verifica, si è proceduto eseguendo le misure stabilite; vengono qui riportati i risultati generali di tali misurazioni, il dettaglio è presente come commento all'interno del codice di ogni componente;
  - **Numero di Parametri**
    Come si può notare nell'istogramma in figura 1, il numero di parametri per metodo è stato mantenuto in generale molto basso rispettando l'obiettivo di non superare il numero massimo fissato a 6. È comunque presente una eccezione, la funzione *generaPassword()* della classe Sistema, la cui non adesione al vincolo è giustificata da scelte progettuali. L'istogramma illustra le rilevazioni effettuate.

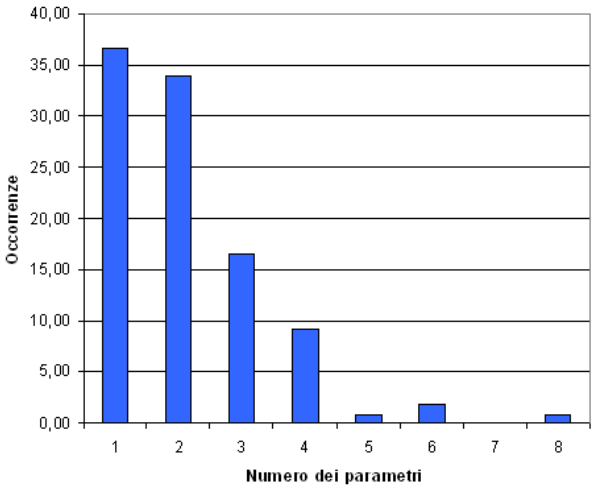

Figura 1: *numero di parametri in relazione al numero di metodi*

  - **Numero di Livelli**

Come si può notare nell'istogramma in figura 2, il numero di livelli per metodo è stato mantenuto in generale molto basso, ad eccezione di qualche caso particolare. L'istogramma illustra le rilevazioni effettuate.

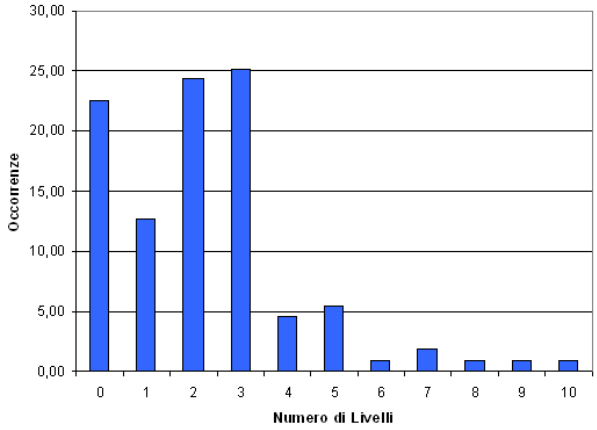

Figura 2: *numero di livelli in relazione al numero di metodi*

– **Complessità Ciclomatica**

Come si può notare nell'istogramma in figura 3, la complessità ciclomatica per metodo è stata mantenuta in generale molto bassa, ad eccezione di qualche caso particolare in cui la presenza di costrutti di tipo switch ha causato alcuni aumenti. Nel complesso, il vincolo di non superarare livelli di complessità ciclomatica pari a 10 è stato sempre rispettato. L'istogramma illustra le rilevazioni effettuate.

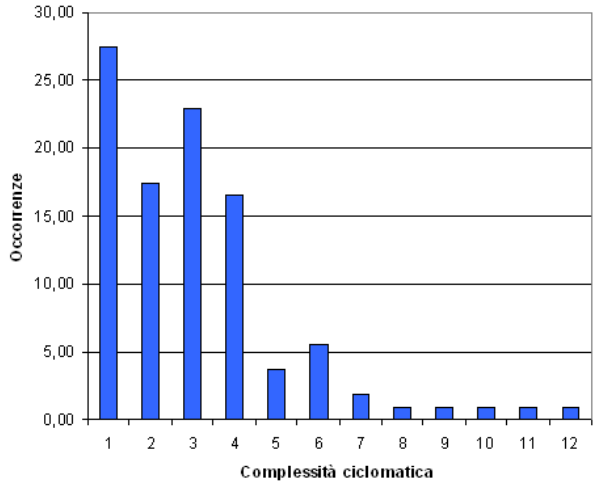

Figura 3: *complessità ciclomatiche in relazione al numero di metodi*

– **Anomalie riscontrate**
  Come si può notare nell'istogramma in figura 4, il numero di anomalie riscontrate si è mantenuto fin dall'inizio generalmente basso, diminuendo nel corso delle varie iterazioni di verifica. L'istogramma illustra le rilevazioni effettuate nelle varie iterazioni di verifica per componente: è stata registrata la media dei valori delle anomalie di un componente per iterazione.

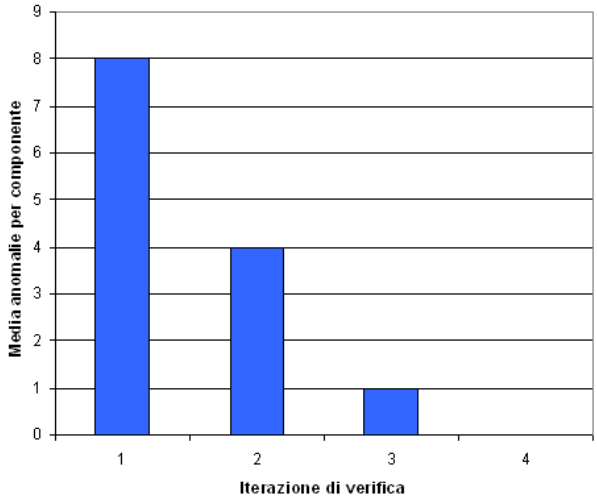

Figura 4: *media anomalie per componente in relazione alle iterazioni di verifica*

Esempio di metriche rilevate per il componente sistema:

Metrica	Media
Numero di parametri	0,75
Numero di livelli	2,3
Complessità ciclomatica	3,6
Fan-in/Fan-out	8/2
Anomalie	8

Con l'utilizzo pianificato di Simpletest, strumento di supporto per la realizzazione di test, è stato poi realizzato il codice per la realizzazione dei tests prestabiliti.

Per ogni componente è stato realizzato un numero di test adeguato rispetto alle metriche rilevate. Poiché il codice del prodotto è stato scritto in maniera congrua alle norme di codifica e svolge azioni semplici e sequenziali (raramente presenta annidamenti che possono causare aumenti indesiderati della complessità ciclomatica e/o del numero dei livelli), anche il codice che effettua test risulta essere piuttosto semplice.

Simpletest

Simpletest permette di verificare tramite asserzioni il corretto funzionamento del codice. Sono state create le cartelle *simpletest* e *tests* all'interno di **${WEB-ROOT}/../**. La cartella *simpletest* contiene il framework simpletest, mentre *tests* contiene file php che effettuano tests.
Sono state testate le componenti sinora realizzate con particolare attezione alla classe *Sistema* in quanto contenente la maggiorparte delle funzioni che le restanti classi utilizzano (connessione al SIAGASDB, generazione password, etc...). Il test delle componenti per cui la fase di realizzazione non è ancora completata verrà effettuato nella fase finale. Come pianificato, non sono stati effettuati test sul framework PEAR.MDB2 in quanto è un package esterno fornito come stabile.

Simpletest mette a disposizione i seguenti metodi:

assertTrue($x)	Fail if $x is false
assertFalse($x)	Fail if $x is true
assertNull($x)	Fail if $x is set
assertNotNull($x)	Fail if $x not set
assertIsA($x, $t)	Fail if $x is not the class or type $t
assertNotA($x, $t)	Fail if $x is of the class or type $t
assertEqual($x, $y)	Fail if $x == $y is false
assertNotEqual($x, $y)	Fail if $x == $y is true
assertWithinMargin($x, $y, $m)	Fail if abs($x - $y) ¡ $m is false
assertOutsideMargin($x, $y, $m)	Fail if abs($x - $y) ¡ $m is true
assertIdentical($x, $y)	Fail if $x == $y is false or a type mismatch
assertNotIdentical($x, $y)	Fail if $x == $y is true and types match
assertReference($x, $y)	Fail unless $x and $y are the same variable
assertClone($x, $y)	Fail unless $x and $y are identical copies
assertPattern($p, $x)	Fail unless the regex $p matches $x
assertNoPattern($p, $x)	Fail if the regex $p matches $x
expectError($x)	Swallows any upcoming matching error
assert($e)	Fail on failed expectation object $e
expectError($x)	Swallows any upcoming matching error
assert($e)	Fail on failed expectation object $e
setUp()	Runs this before each test method
tearDown()	Runs this after each test method
pass()	Sends a test pass
fail()	Sends a test failure
error()	Sends an exception event
signal($type, $payload)	Sends a user defined message to the test reporter
dump($var)	Does a formatted print_r() for quick and dirty debugging

Test funzionali vengono effettuati eseguendo un metodo con un set di valori in input e verificando che l'output fornito sia quello atteso. La cardinalità del set di valori in input dipende dalle metriche rilevate per il modulo in

esame.
Vengono qui forniti degli esempi di test effettuati sulla classe Sistema.

Si mostra di seguito un esempio di test effettuato sulla classe *Sistema*.

```
<?php
if (! defined('DIR_UP')) {
   define('DIR_UP', '../');
}
require_once(DIR_UP . 'simpletest/unit_tester.php');
require_once(DIR_UP . 'simpletest/reporter.php');
require_once(DIR_UP . 'attori/Sistema.class.php');

class TestOfSistema extends UnitTestCase {

    // controllo del funzionamento di getSistema()
   function TestOfgetSistema() {
      $riferimento = null;
      $this->assertNull($riferimento);
      $riferimento = Sistema::getSistema();
      $this->assertNotNull($riferimento);
   }

    // controllo del funzionamento di creaPassword()
   function TestOfcreaPassword() {
      $rif = null;
      $this->assertNull($rif);
      $rif = Sistema::creaPassword(7,1,1,1,1,1,0);
      $this->assertNotNull($rif);
   }

   // controllo del funionamento di valida()
   // in caso di stringhe diverse
   function TestOfValida() {
      $rif_sistema = Sistema::getSistema();
      $stringa_corretta = "stringa di prova";
      $stringa_errata = "stringa d<i pro>va";
      $this->assertNotEqual($stringa_corretta,
         $stringa_errata);
      $rif_stringa_corretta = $rif_sistema->valida("
         stringa di prova");
      $rif_stringa_errata = $rif_sistema->valida("
         stringa d<i pro>va");
      $this->assertNotEqual($rif_stringa_corretta,
         $rif_stringa_errata);
   }

   // controllo del funzionamento di backupTotale()
```

```
    // in caso di path inesistente
    function TestOfBackup() {
        $rif_sistema = Sistema::getSistema();
        $path_esistente = "/home/utente/";
        $path_inesistente = "/homee/utente/";
        $this->assertNotEqual($stringa_corretta,
            $stringa_errata);
        $rif_backup_effettuato = $rif_sistema->
            backupTotale($path_esistente);
        $rif_bakcup_non_effettuato = $rif_sistema->
            backupTotale($path_inesistente);
        $this->assertTrue($rif_bakcup_non_effettuato);
        $this->assertFalse($rif_bakcup_non_effettuato);
    }
}

$testSistema = &new TestOfSistema();
$testSistema->run(new HtmlReporter()); ?>
```

Simpletest mostra l'output dei test su browser come pagina HTML. Visivamente i risultati si presentano nella seguente maniera:

Log ClasseX

1/1 test cases complete: **22** passes, **0** fails and **0** exceptions.

Figura 5: *esempio di output di test con esito positivo*

Log ClasseX

Fail: Test -> Created before message at [/home/alberto/webroot/ST/tests/log_test.php line 16]
Fail: Test -> Created before message at [/home/alberto/webroot/ST/tests/log_test.php line 17]

1/1 test cases complete: **20** passes, **2** fails and **0** exceptions.

Figura 6: *esempio di output di test con esito negativo*

Tutti i test che inizialmente davano esito negativo, hanno dato esito positivo a fine fase di verifica.

Alcuni esempi di anomalie riscontrate sono i seguenti:

Metodo	**Classe**	**Anomalia riscontrata**	**Effetto indesiderato**
creaPassword()	Sistema	ramo condizionale mai raggiunto	impossibile creazione di password con caratteri speciali
restoreTotale()	Sistema	ritorno inaspettato della funzione	possibile cancellazione del database senza successivo ripristino
stageSenzaTutor()	Amministratore	ramo condizionale mai raggiunto	ritorna un insieme vuoto di Stage senza tutor

In seguito allo svolgimento dei test, parte dei metodi sono stati modificati per correggere le eventuali anomalie. Gli esiti dei vari test effettuati sui metodi della classe Sistema sono riportati nella seguente tabella. Il resoconto complessivo degli esiti di tutti i metodi sarà allegato a realizzazione e verifica completate.

Metodo testato	**Esito**
getSistema()	positivo
getMailGestore()	positivo
getSito()	positivo
valida()	positivo
connectMe()	positivo
query()	positivo
exec()	positivo
creaPassword()	positivo
setConfigurazione()	positivo
setSito()	positivo
backupTotale()	positivo
restoreTotale()	positivo

5.3.3 Data Layer

Analisi Statica

L'analisi statica è stata realizzata verificando la correttezza sintattica e l'aderenza alle norme del codice creato.
Le correzioni apportate sono risultate essere 12.

Analisi Dinamica

L'analisi dinamica è stata realizzata manualmente, tramite query con valori nei casi limite e tramite il package PEAR:MDB2.

6 Attività di verifica finali

A realizzazione ultimata saranno eseguiti tutti i test creati, con particolare attenzione alle ultime componenti realizzate, alla loro corretta integrazione nel sistema e alla gestione degli aspetti critici rilevati nella Definizione di Prodotto (concorrenza, sicurezza, backup).

6.1 Dettaglio della fase finale

Ai fini del collaudo, il prodotto verrà testato in due modalità:

- α Test (pre collaudo): Attività che verrà fatta internamente al fornitore, localmente e attraverso www.wheelsoft.org. Wheelsoft si impegna a completare i test di tutte le componenti secondo le modalità sopra descritte riportandone tutti gli esiti;
- β Test (collaudo) : Attività controllata e guidata dal committente basata sui test svolti nell'α Test.