

POLITECNICO DI TORINO

III Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

01BIF - Ingegneria del software II

Test e modifica di software prodotto da altri



DAVIDE BERTOLA MATR. 151404
LUCA ARDITO MATR. 151376

Aprile 2008

Indice

1	Introduzione	2
2	Fase 0: Setup software e checkout	3
3	Fase 1: Project Management	4
3.1	Polarion	4
3.2	Project Plan	5
4	Fase 2: Test d'accettazione (black box)	6
5	Fase 3: Test unitario, test integrazione e design review (white box)	9
6	Fase 4: Defects fixing	11
7	Fase 5: Post mortem analysis	12

1 Introduzione

Il software da testare è un termostato evoluto dotato di una GUI per mezzo della quale l'utente può monitorare la situazione delle varie stanze della casa e può impostare parametri quali la temperatura obiettivo, l'apertura/chiusura delle finestre ecc.

Il progetto è suddiviso in sei fasi:

- 0 Setup software e checkout (in eclipse)
- 1 Project Management
- 2 Test d'accettazione (black box)
- 3 Test unitario, test integrazione e design review (white box)
- 4 Defects fixing
- 5 Post mortem analysis

Per ogni fase sarà disponibile un documento che descrive dettagliatamente ciò che è stato eseguito.

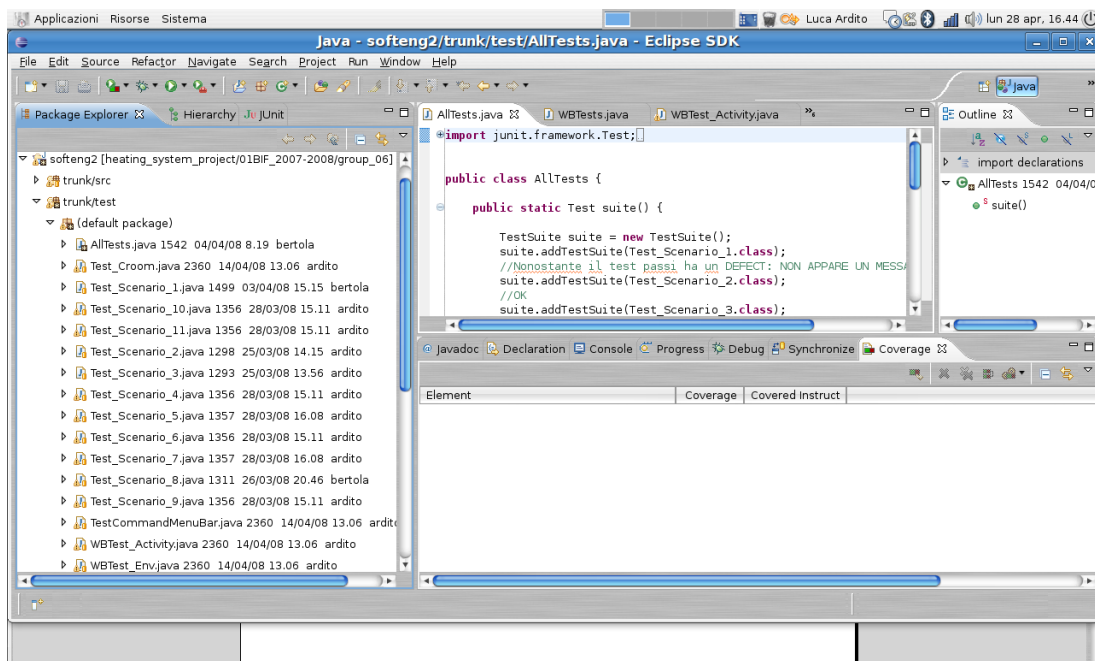
Ogni fase ha una data entro cui deve essere completata.

In una prima fase del progetto è stato reso disponibile solamente l'archivio jar contenente i file class. In questo modo è stato effettuato il test di accettazione (black box) dove per mezzo della gui si andavano a testare le varie caratteristiche del programma senza conoscere il codice sorgente.

Nella seconda fase è stato reso disponibile il codice sorgente ed è stato possibile effettuare il test white box e correggere alcuni degli errori presenti. Di conseguenza è stata stesa un'analisi post mortem nella quale si analizzano gli aspetti positivi e negativi e le esperienze collezionate durante la fase di progetto.

2 Fase 0: Setup software e checkout

Questa fase consiste nel configurare eclipse in modo tale da poter utilizzare subversion. E' stato fornito un apposito documento contenente le istruzioni per portare a termine questa prima fase. I file del progetto sono disponibili come normali file java ma si tratta di una working copy. Ogni volta che si vuole rendere disponibili anche agli altri partecipanti al progetto è sufficiente eseguire un commit. Quando si rende necessario aggiornare la propria working copy basta eseguire una update.



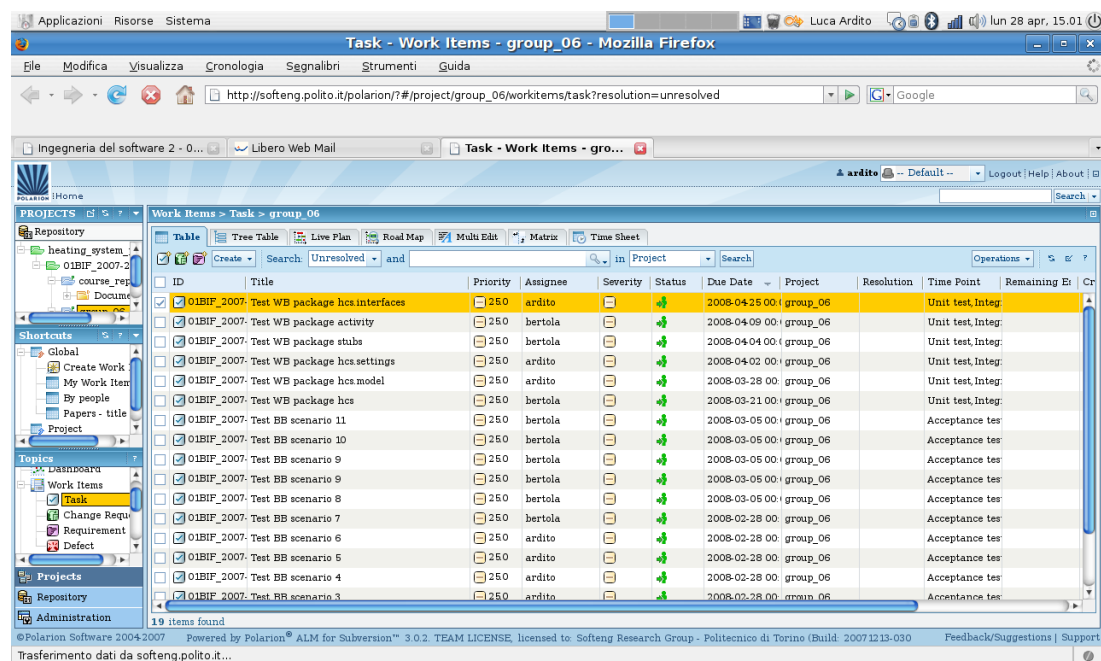
3 Fase 1: Project Management

Questa fase consiste nel gestire il progetto. Alla fine di questa fase dovranno essere generati i live plan su polarion ed un documento di project plan contenente una stima di tempi e costi

3.1 Polarion

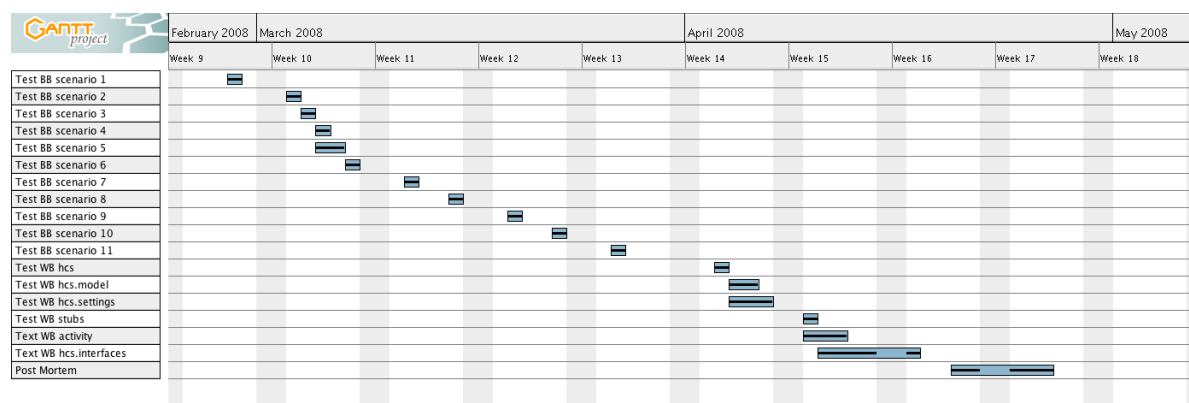
Polarion integra l'intero processo di sviluppo intorno al repository di Subversion aumentando ed espandendo le funzionalità di questo sistema di versioning ad altri domini (issue, time e change management, l'IT governance).

In base ai requirements forniti sono stati aperti assegnati dei task ai membri del gruppo e sono stati aperti dei defect. Alcuni defect sono stati fixati e per alcuni non è stato possibile. Tutte queste azioni sono state segnalate su polarion.



3.2 Project Plan

Il team di lavoro è composto da due persone. Il documento di sistema specifica tutti i requisiti contraddistinti da un nome univoco. Sono presenti inoltre undici possibili scenari d'uso in cui il sistema si può trovare. Ogni scenario deve soddisfare alcuni requisiti. I test d'accettazione devono essere eseguiti in questa direzione in quanto non si è a conoscenza del codice sorgente. Si assegnano ad Ardito i test relativi agli scenari 1 2 3 4 5 6 mentre si assegnano a Bertola i test relativi agli scenari 7 8 9 10 11. Per ogni scenario deve essere aperto un relativo task su polarion. Qualora il software non passasse qualche test deve essere aperto un relativo defect su polarion. La fase di accettazione deve essere conclusa entro il 16 Marzo 2008, la definizione del presente documento e i test white box devono essere completati entro il 13 Aprile 2008 mentre la fase di fix dei defect e l'analisi post mortem deve essere completata entro il 20 Aprile 2008. Il diagramma di Gantt è riportato di seguito.



4 Fase 2: Test d'accettazione (black box)

I test black box sono stati eseguiti utilizzando JFCTestCase.

Questa libreria permette di simulare l'utilizzo di una GUI automaticamente. In questo modo è stato possibile simulare l'utilizzo del programma e ottenere i valori leggendoli direttamente dall'interfaccia grafica. JFCTestCase è utilizzabile come JUnit; di conseguenza con delle assert si effettua un oracolo basandosi sui requisiti. Successivamente si verifica se il programma li rispetta. Come descritto nel project plan i test black box sono stati costruiti simulando gli scenari descritti nel documento di sistema. Di conseguenza ci sono 11 test che possono essere inseriti in una suite oppure possono essere eseguiti uno alla volta. Per ogni test non passato è stato aperto un defect su polarion. I defect aperti sono relativi ai requisiti: Temp-UR-F 11,Temp-UR-F 13,Temp-UR-F 14. Il sistema non fornisce un messaggio di errore se la standard inside time non è più piccola di almeno 4 gradi rispetto alla standard heating time. Viene lanciata un'eccezione che è visibile solamente in console e l'utente non è a conoscenza che i dati errati saranno scartati. L'eccezione inoltre non viene lanciata se viene inserita la standard inside time prima della standard heating time.

Il requisito Temp-UR-F 13 afferma che la Target temperature deve essere settata a standardAwayTemp ma in realtà si trova a 0.0.

Il requisito Temp-UR-F 14 afferma che il sistema deve chiudere la finestra ma ciò non accade.

Di seguito è riportato un esempio di test black box e precisamente il test per lo scenario 2.

```
import gui.Main;
import gui.MainFrame;
import hcs.model.Room;
import java.awt.Container;
import java.awt.Frame;
import javax.swing.JComponent;
import junit.extensions.jfcunit.JFCTestCase;
import junit.extensions.jfcunit.JFCTestHelper;
import junit.extensions.jfcunit.TestHelper;
import junit.extensions.jfcunit.eventdata.JFCEventManager;
import mystubs.CRoom;
import mystubs.Env;
import mystubs.PhysBoiler;
import stubs.PhysicalFactory;
import activity.SetClockActivity;
```

```

public class Test_Scenario_2 extends JFCTestCase {

    MainFrame frame;
    Frame manager = null;

    protected void setUp( ) throws Exception {
        super.setUp( );
        JFCEventManager.setDebug(true);
        JFCEventManager.setDebugType(JFCEventManager.DEBUG_OUTPUT);
        JFCEventManager.setRecording(true);
        Main.main((new String[3]));
        this.frame = Main.getMainFrame();
        setHelper( new JFCTestHelper( ) );
        SetClockActivity s = new SetClockActivity();
        s.setDelay(10);
    }

    public void testScenario2() throws InterruptedException{
        Env e = (Env) PhysicalFactory.getPhysicalEnvironment();
        e.setRain(true);
        e.setTemperature(21.0);
        PhysBoiler b = (PhysBoiler) PhysicalFactory.getPhysicalBoiler();
        b.setTargetTemperature(15.0);
        CRoom r = (CRoom)PhysicalFactory.getPhysicalRoom("living room", 0);
        r.setPresence(false);
        r.setTemperature(18.0);
        Thread.sleep(40);
        r.setPresence(true);
        Container c = frame.getContentPane();
        gui.RoomMonitor rm = (gui.RoomMonitor)
            getComponentByName(c, "living room RoomMonitor");
        Room _r = rm.getRoom();
        int standardInsideTime = _r.getSettings().getStandardInsideTime()*60*10;
        boolean changed=false;
        Thread.sleep(standardInsideTime/2);
        assertEquals(15.0,b.getTargetTemperature());
        r.setPresence(false);
        Thread.sleep(standardInsideTime + 100);
        if(b.getTargetTemperature()!=15.0)
            changed=true;
        assertFalse(changed);
    }
}

```

```
public static JComponent getComponentByName(Container container,String name) {
    for (int i = 0; i < container.getComponentCount(); i++) {
        if (name.equals(container.getComponent(i).getName())) {
            return (JComponent) container.getComponent(i);
        }
    }
    return null;
}

protected void tearDown() throws Exception{
    this.frame.setVisible(false);
    frame.dispose();
    frame = null;
    TestHelper.cleanUp( this );
    super.tearDown( );
}
}
```

5 Fase 3: Test unitario, test integrazione e design review (white box)

Questa fase prevede l'utilizzo del codice sorgente dell'applicazione.

La tecnica di integrazione usata è di tipo bottom up ovvero per ogni classe testata sono stati costruiti dei driver in grado di pilotarla. A causa delle tempistiche a disposizione non è stato possibile un test accurato di ogni classe.

Si utilizza il plug-in per Eclipse EclEmma che permette di monitorare la percentuale di codice coperta dai test. Non tutto il codice sorgente dell'applicazione è raggiungibile in quanto molti metodi del programma sono privati oppure protetti. Per ovviare a questo problema si renderebbe necessaria la reflection ma non sempre risulta efficace.

La copertura del codice è del 76% calcolando che la gui non è stata presa in considerazione al fine dei test.

Un'altra problematica relativa a questa fase è il file XML contenente i settaggi: il suo percorso non è un parametro del programma di conseguenza non è possibile eseguire test che passano una volta un file corretto ed una volta un file non valido. In questo modo si renderebbe possibile testare in una sola volta il codice che gestisce un file funzionante e il codice che gestisce un file non valido. Ciò può comunque essere verificato inserendo manualmente file non validi.

In aggiunta ai defect rilevati durante il test di accettazione si possono rilevare due defect che però non violano i requirement iniziali.

La classe relativa all'orologio di sistema accetta anche valori negativi e non vi è alcun controllo che lo impedisca.

Il file xml non è validato rispetto alla propria dtd. Potrebbero esserci dunque degli elementi inconsistenti senza che il programma possa rilevarli.

Di seguito è riportato un esempio di test WhiteBox e precisamente il test per la classe che gestisce l'orologio di sistema.

```
import junit.extensions.jfcunit.JFCTestCase;
import junit.extensions.jfcunit.JFCTestHelper;
import junit.extensions.jfcunit.TestHelper;
import junit.extensions.jfcunit.eventdata.JFCEventManager;
import activity.SetClockActivity;

public class WBTest_SetClockActivity extends JFCTestCase {
    protected void setUp( ) throws Exception {
        super.setUp( );
        JFCEventManager.setDebug(true);
        JFCEventManager.setDebugType(JFCEventManager.DEBUG_OUTPUT);
        JFCEventManager.setRecording(true);
    }

    public void testClock() throws InterruptedException{
```

```

        SetClockActivity sca = new SetClockActivity();
        sca.setDelay(10);
        assertEquals(sca.getDelay(),10);
        sca.start();
        sca.stop();
        sca.setDelay(10);
        sca.start();
        sca.start();
        sca.stop();
        sca.setDelay(0);
        assertEquals(sca.getDelay(),0);
        sca.stop();
        sca.stop();
        sca.setDelay(-1);
        assertEquals(sca.getDelay(),-1);
        sca.getDelay();
        sca.start();
        sca.stop();
    }
    protected void tearDown() throws Exception{
        TestHelper.cleanup( this );
        super.tearDown( );
    }
}

```

6 Fase 4: Defects fixing

Una volta stabilite le problematiche che riguardano l'applicazione si rende necessario, ove possibile, cercare di correggerle.

In questo caso non tutti i defect sono stati fixati. I defect relativi al file xml ad esempio, non hanno un'importanza tale da mettere in piedi un meccanismo di validazione di un file xml rispetto ad una propria dtd. Si preferisce assumere che l'utente ne inserisca uno valido.

Non sono stati risolti inoltre i problemi relativi alla target temperature. Da un'analisi del codice risulta che la target temperature del boiler o dell'heater assume solamente i valori 0.0 e 70.0 rispettivamente se la temperatura dell'ambiente è superiore a quella decisa dall'utente oppure se la temperatura dell'ambiente è inferiore a quella decisa dall'utente.

Di conseguenza non si fa riferimento alla `standardAwayTemp` o alla `standardPresenceTemp` se non per decidere se accendere o meno il boiler o l'heater in base alla strategy utilizzata.

Sono invece stati fixati i defect relativi alla chiusura automatica della finestra e all'eccezione dovuta ad una differenza di almeno 4 minuti tra `standardHeatingTime` e `standardInsideTime`. E' stata inoltre modificata la `InvalidTimeException` in modo da far visualizzare all'utente un messaggio di errore sottoforma di message box come richiesto dai requirements dello scenario 1. L'applicazione infatti, scatenava l'eccezione in console ma l'utente non era in grado di accorgersi che i dati non validi non sarebbero stati presi in considerazione dall'applicazione. Per quanto riguarda la classe `Room` e le modifiche relative alla finestra nel codice sorgente sono stati inseriti dei commenti che spiegano dettagliatamente ogni modifica che è stata effettuata.

7 Fase 5: Post mortem analysis

Non avendo mai avuto esperienze passate in ambito di test ogni dettaglio risulta essere importante.

Sono stati commessi alcuni errori di valutazione e non tutte le correzioni al codice sorgente sono state possibili.

La documentazione del sistema a volte risulta poco comprensibile e a volte si renderebbe necessario chiarire con i programmatori alcuni aspetti del software.

Sarebbe inoltre stato più efficiente testare l'applicativo durante la fase di programmazione e non alla fine. Comprendere la totalità del codice non è stato semplice e ha impiegato molto tempo data la dimensione del programma e il poco tempo a disposizione.

In ogni caso l'applicazione è stata migliorata e alcuni requisiti dopo questa fase di test sono stati rispettati.