Button Led System
process report

Zanotti Andrea

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
andrea.zanotti9@studio.unibo.it

1 Introduction

2 Vision

Non esiste codice senza progetto, non esiste progetto senza analisi del problema e non esiste problema senza requisiti.
Non c'è codice senza test.
Button e Led sono metafore per input e output.
Mantenere disaccopiati logica realizzativa del progetto e testing.
Individuare diverse tipologie di test da applicare in diversi momenti della realizzazione:

- Unit testing (singolo componente, white box)
- Integration Testing (di interazione, white box)
- Functional Testing (sull'intero sistema, black box)
- Stress Load Testing (performance, black box)
- User Acceptance Testing (feedback utente, black box)

Utilizzare approccio top-down in fase di progettazione e bottom-up in fase di implementazione (zooming).
Zooming: sviluppo del progetto senza curarsi delle tecnologie che verranno utilizzate per implementarlo.
Il Button Led System deve essere una applicazione compatibile con IoT.
Rendere l'applicazione Button Led System funzionante a prescindere dalla posizione, dalla tecnologia d'implementazione e dalla tecnologia utilizzata per la comunicazione (ICT).
Realizzare il progetto seguendo le linee guida del framework SCRUM.
Sempre nell'ambito delle linee guida SCRUM, all'interno dello sprint backlog, devono essere individuate funzionalità utili all'intera software house, che verranno (nello sprint successivo) fattorizzate e rese disponibili come libreria. Realizzazione di una entità esterna che esegue la configurazione dell'intero Button Led System in modo autonomo e configurabile, in modo da non delegare ai componenti il compito della configurazione.
Abbattere i costi per un prodotto di qualità: raggiunti i requisiti minimi di funzionalità continuare a investire risorse nel progetto per aumentarne la qualità e abbatterne i costi.

3 Goals

Implementare ogni componente del ButtonLedSystem (button, led e controller) su piattaforma:

- Android
- Raspberry
- Arduino
- Java SE

4 Requirements

Design and build a ButtonLed software system in which a Led is turned on and off each time a Button is pressed (by an human user). The system should run on a single support, e.g. a conventional PC, a Raspberry Pi or Arduino.

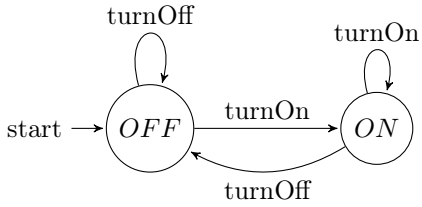
5 Requirement analysis

5.1 Use cases

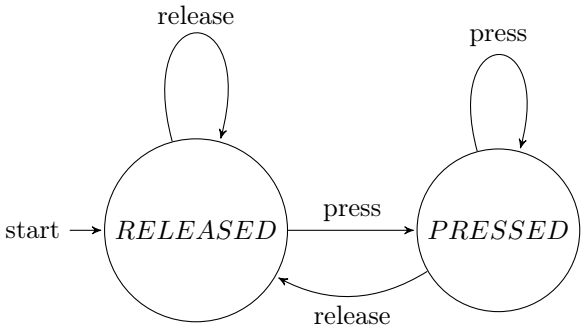
5.2 Scenarios

5.3 (Domain)model

LED: Struttura: entità atomica, passiva.
Interazione: espone le operazioni void turnOn, void turnOff e boolean getState. Queste operazioni sono anche metodi nell'accezione OOP.
Comportamento: alla creazione un LED è spento. L'esecuzione dell'operazione turnOn accende il Led, l'esecuzione dell'operazione turnOff lo spegne. L'operazione getState ritorna lo stato del Led (false →spento, true →acceso).
Viene esplicitato dal seguente automa a stati finiti:



BUTTON:
Struttura: entità atomica, attiva.
Interazione: una sola operazione per conoscere lo stato, boolean getState.
Comportamento: alla creazione un button è released.
Viene esplicitato dal seguente automa a stati finiti:



Button e Led non sono correlati →il button non ha alcuna conoscenza del led e viceversa, per realizzare il sistema è necessaria un'entità coordinatrice.

5.4 Test plan

LED:
Il test plan (di tipo unit) deve verificare che inizialmente getState ritorni il valore false. Dopo aver eseguito l'operazione turnOn, getState deve restituire il valore true.

```
package it.unibo.ing.iss.buttonledJava.tests;

import static org.junit.Assert.*;
import it.unibo.ing.iss.buttonledJava.impl.JavaLed;
import it.unibo.ing.iss.buttonledJava.interfaces.leds.ILed;

import org.junit.Before;
import org.junit.Test;

public class LedTest {
        private ILed led = new JavaLed();
        @Before
        public void setUp() throws Exception {
                led = new JavaLed();
        }

        @Test
        public void test() {
                assertFalse(led.isOn());
                led.turnOn();
                assertTrue(led.isOn());
                led.turnOff();
                assertFalse(led.isOn());
```

```
        }

}
```

BUTTON:
É necessaria una nuova entità (DebuggableButton) per poter modificare lo stato del button, tale entità esporra le operazioni, void press, void release.

```
package it.unibo.ing.iss.buttonledJava.tests;

import static org.junit.Assert.*;
import it.unibo.ing.iss.buttonledJava.impl.JavaButton;
import it.unibo.ing.iss.buttonledJava.interfaces.buttons.IDebuggableButton;

import org.junit.Test;

public class ButtonTest {
        @Test
        public void basicTest() {
                IDebuggableButton bt = new JavaButton();
                assertFalse(bt.isPressed());
                bt.press();
                assertTrue(bt.isPressed());
                bt.release();
                assertFalse(bt.isPressed());
                bt.release();
                assertFalse(bt.isPressed());
        }

        @Test
        public void advancedTest(){
                IDebuggableButton bt = new JavaButton();
                bt.press();
                bt.press();
                bt.press();
                assertTrue(bt.isPressed());
                bt.release();
                assertFalse(bt.isPressed());
                bt.release();
                assertFalse(bt.isPressed());
                bt.press();
                assertTrue(bt.isPressed());
        }

}
```

BUTTONLEDSYSTEM:

```
package it.unibo.ing.iss.buttonledJava.tests;

import static org.junit.Assert.*;
import it.unibo.ing.iss.buttonledJava.impl.JavaButtonLed;
import it.unibo.ing.iss.buttonledJava.interfaces.IButtonLed;

import org.junit.Before;
import org.junit.Test;

public class ButtonLedTest {

	@Before
	public void setUp() throws Exception {

	}

	@Test
	public void repeatedPushFromOff() {
		IButtonLed btLed = new JavaButtonLed(false,false);
		btLed.pushButton();
		assertTrue(btLed.getLedState());
		btLed.pushButton();
		assertTrue(btLed.getLedState());
	}

	@Test
	public void repeatedPushFromOn() {
		IButtonLed btLed = new JavaButtonLed(true, false);
		btLed.pushButton();
		assertFalse(btLed.getLedState());
		btLed.pushButton();
		assertFalse(btLed.getLedState());
	}

	@Test
	public void repeatedPushFromPressed() {
		IButtonLed btLed = new JavaButtonLed(false,true);
		btLed.pushButton();
		assertFalse(btLed.getLedState());
		btLed.pushButton();
		assertFalse(btLed.getLedState());
	}
```

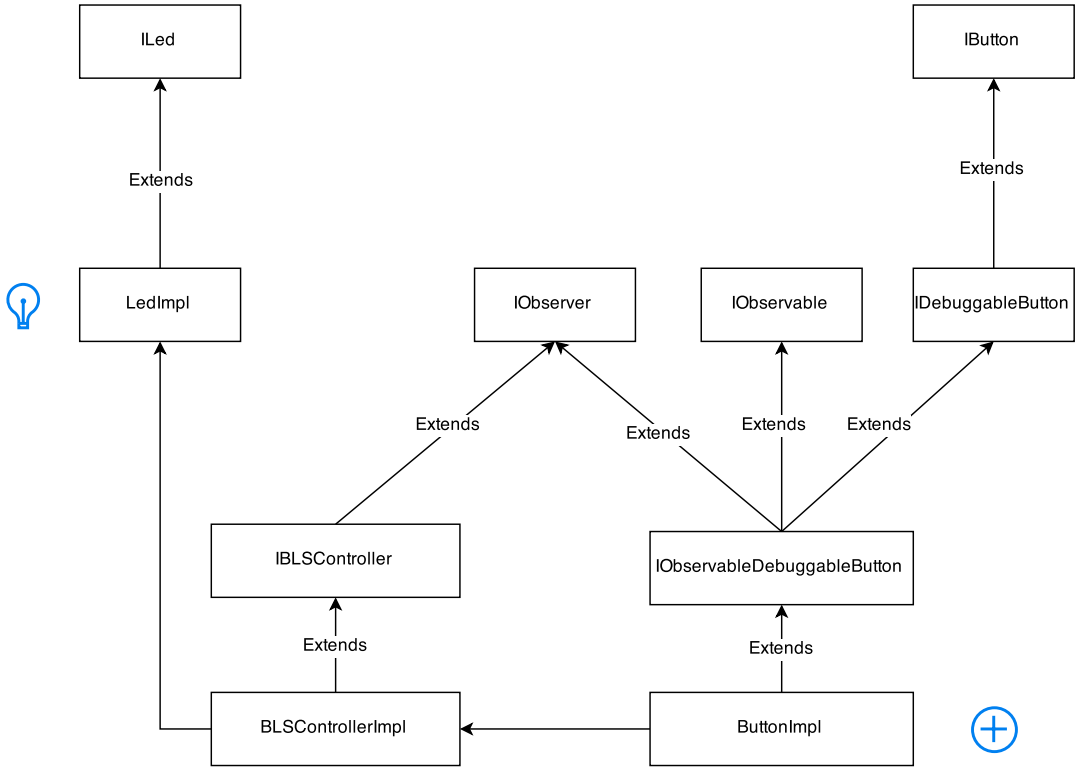
```
    @Test
    public void repeatedPushFromUnpressed() {
            IButtonLed btLed = new JavaButtonLed(false,false);
            btLed.pushButton();
            assertTrue(btLed.getLedState());
            btLed.pushButton();
            assertTrue(btLed.getLedState());
    }

    @Test
    public void repeatedPushFromOnAndUnpressed() {
            IButtonLed btLed = new JavaButtonLed(true, false);
            btLed.pushButton();
            assertFalse(btLed.getLedState());
            btLed.pushButton();
            assertFalse(btLed.getLedState());
    }

    @Test
    public void repeatedPushFromOnAndPressed() {
            IButtonLed btLed = new JavaButtonLed(true, true);
            btLed.pushButton();
            assertTrue(btLed.getLedState());
            btLed.pushButton();
            assertTrue(btLed.getLedState());
    }

    @Test
    public void repeatedPushFromOffAndUnpressed() {
            IButtonLed btLed = new JavaButtonLed(false, false);
            btLed.pushButton();
            assertTrue(btLed.getLedState());
            btLed.pushButton();
            assertTrue(btLed.getLedState());
    }

    @Test
    public void repeatedPushFromOffAndPressed() {
            IButtonLed btLed = new JavaButtonLed(false, true);
            btLed.pushButton();
            assertFalse(btLed.getLedState());
            btLed.pushButton();
            assertFalse(btLed.getLedState());
    }
}
```

6 Problem analysis

Per risolvere il problema di scalabilità e mantenere le entità disaccoppiate faremo utilizzo del pattern `Observer`: il Button espone un'interfaccia tramite la quale le entità possono registrarsi.
Dividiamo i componenti dell'architettura in Attivi e Passivi.
I componenti Passivi generalmente generano eventi, i componenti Attivi possono ricevere comandi. Per la realizzazione di tale funzionalità verrà utilizzato il pattern `Command`.

6.1 Logic architecture

L'architettura logica è esplicitata in figura.



6.2 Abstraction gap

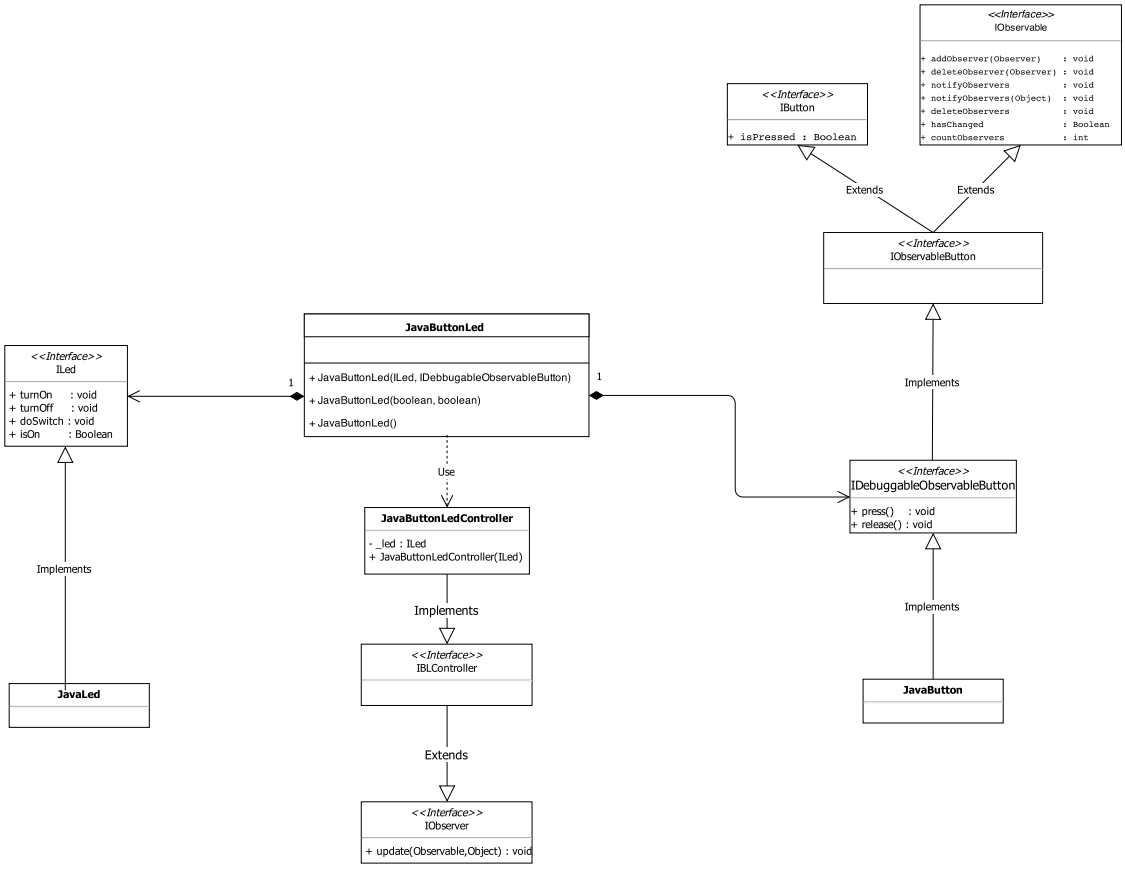
6.3 Risk analysis

Nell'interfaccia del Button, oltre al metodo accessor `isPressed` deve essere presente anche il metodo `addObserver`, ciò è risultato necessario dai colloqui con il committente (durante l'analisi dei requisiti).

7 Work plan

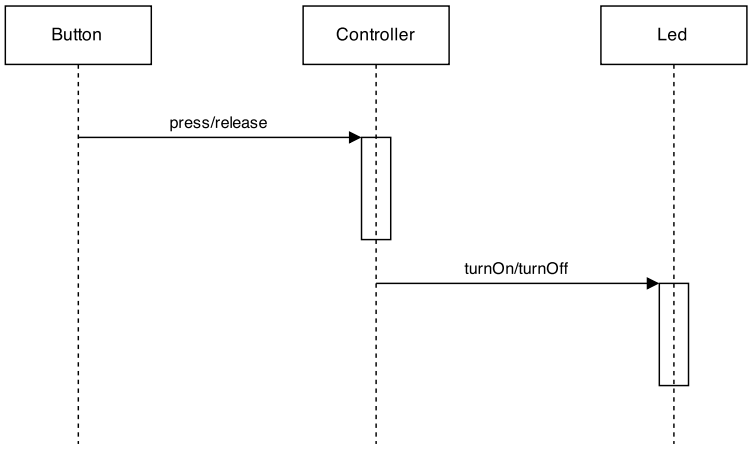
8 Project

L'architettura progettuale è esplicitata in figura.
`https://github.com/iBelliDiISS/ButtonLedJava`



8.1 Structure

8.2 Interaction



8.3 Behavior

9 Implementation

10 Testing

11 Deployment

12 Maintenance

A Codice

B Information about the author

Photo of the Author