Προγραμματισμός II
Πρακτικές ελέγχου, επιθεώρησης και συνεχούς ολοκλήρωσης

Διομήδης Σπινέλλης
Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας
Οικονομικό Πανεπιστήμιο Αθηνών

dds@aueb.gr
http://www.dmst.aueb.gr/dds
@CoolSWEng

2023-11-29

Σφάλματα στον κώδικα

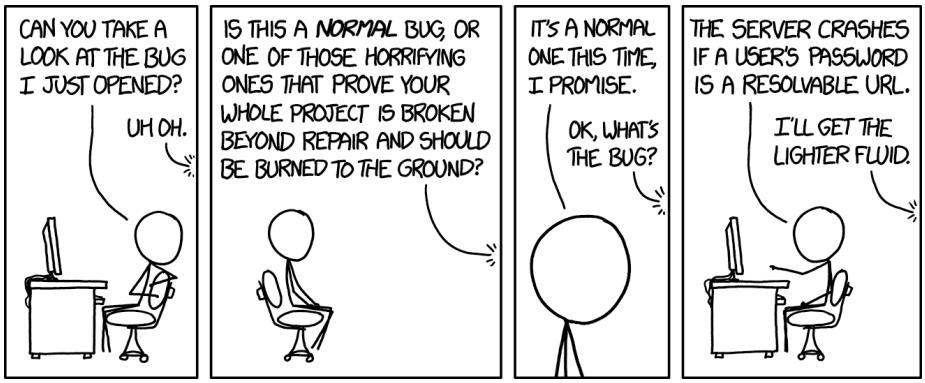

Figure 1: Σφάλματα στον κώδικα

Περιεχόμενα

- Δοκιμές μονάδας
- Επιθεώρηση κώδικα
- Συνεχής ολοκλήρωση μέσω GitHub Actions

Δοκιμές μονάδας

- Στο μικρότερο ελεγχόμενο άρθρωμα
- Αυτόνομες
- Ανεξάρτητες μεταξύ τους
- Αναπτύσσονται πριν/μαζί με τον κώδικα

- Εκτελούνται αυτόματα
- Κατά τη δόμηση
- Κατά τη (συνεχή) ολοκλήρωση
- Στόχος η πλήρης κάλυψη του κώδικα

Πλεονεκτήματα δοκιμών μονάδας

- Έγκαιρη διάγνωση προβλημάτων
- Διευκολύνουν την αναδόμηση του κώδικα
- Διευκολύνουν την ολοκλήρωση
- Παρέχουν τεκμηρίωση
- Επιβάλλουν αρθρωτό σχεδιασμό (αλλά δυσκολεύουν την ενθυλάκωση)

Παράδειγμα δοκιμών μονάδας

```
package gr.aueb.dmst.dds.example;

import gr.aueb.dmst.dds.example.LinkedList;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class TestLinkedList {

    private LinkedList<Integer> list;
    private final int firstElement = 4;
    private final int missingElement = 42;

    @Before
    public void setUp() {
        list = new LinkedList<Integer>(firstElement);
    }

    @Test
    public void testGetSize() {
     Assert.assertEquals("failure - wrong size", list.size(), 1);
        list = list.add(1);
     Assert.assertEquals("failure - wrong size", list.size(), 2);
    }

    @Test
    public void testContains() {
     Assert.assertTrue("failure - does not contain first element",
                list.contains(firstElement));
```

```
 Assert.assertFalse("failure - contains missing element",
            list.contains(missingElement));
}

@Test
public void testGetHead() {
 Assert.assertEquals("failure - wrong head", list.getHead(),
            (Integer)firstElement);
}

@Test
public void testGetTail() {
    final int secondElement = 1000;
    list = list.add(secondElement).getTail();
 Assert.assertEquals("failure - wrong size", list.size(), 1);
 Assert.assertEquals("failure - wrong head", list.getHead(),
            (Integer)firstElement);
 Assert.assertTrue("failure - does not contain first element",
            list.contains(firstElement));
 Assert.assertFalse("failure - contains second element",
            list.contains(secondElement));
}

@Test
public void testContructor() {
 Assert.assertEquals("failure - wrong size", list.size(), 1);
 Assert.assertEquals("failure - wrong head", list.getHead(),
            (Integer)firstElement);
 Assert.assertTrue("failure - does not contain first element",
            list.contains(firstElement));
 Assert.assertFalse("failure - contains missing element",
            list.contains(missingElement));
}

@Test
public void testAdd() {
    final int secondElement = 1000;
    list = list.add(secondElement);
 Assert.assertEquals("failure - wrong size", list.size(), 2);
 Assert.assertEquals("failure - wrong head", list.getHead(),
            (Integer)secondElement);
 Assert.assertTrue("failure - does not contain first element",
            list.contains(firstElement));
 Assert.assertTrue("failure - does not contain second element",
            list.contains(secondElement));
 Assert.assertFalse("failure - contains missing element",
```

```
                list.contains(missingElement));
    }

    @Test
    public void testToString() {
     Assert.assertEquals("failure - wrong to String", list.toString(),
                ((Integer)firstElement).toString());
        final int secondElement = 1000;
        list = list.add(secondElement);
     Assert.assertEquals("failure - wrong to String", list.toString(),
              ((Integer)secondElement).toString() + " -> " +
                ((Integer)firstElement).toString());
    }


    @After
    public void tearDown() {
        /* Not really needed */
        list = null;
    }
}
```

Επιθεώρηση κώδικα

- Δεύτερο ζευγάρι μάτια
- Απαραίτητη πριν την τελική ενσωμάτωση του κώδικα
- Εξετάζει
  - Πληρότητα
  - Σφάλματα
  - Ποιότητα

Ροή εργασίας

- Ο Γιάννης έχει υλοποιήσει τη λειτουργία Α
- Ο Γιάννης επιλέγει τη Μαρία για να επιθεωρήσει τον κώδικά του
- Ζητά από τη Μαρία να επιθεωρήσει τον κώδικα
- Η Μαρία επισημαίνει τμήματα που χρειάζονται βελτίωση
- Ο Γιάννης κάνει τις απαραίτητες διορθώσεις
- Ο Γιάννης καταχωρίζει τον κώδικα και ενημερώνει τη Μαρία
- Η Μαρία δίνει LGTM ή επισημαίνει πρόσθετες βελτιώσεις

Υποστήριξη από εργαλεία

- GitHub μέσω pull request
  - Υλοποίηση σε ξεχωριστό αποθετήριο και κλάδο

  - Καταχώρηση
  - Δημιουργία pull request και επιλογή επιθεωρητή
  - Επιθεώρηση με σχόλια ανά γραμμή
  - Αλλαγές
  - git rebase -i

Συνεχής ολοκλήρωση

- Μετά από κάθε καταχώρηση
- Δόμηση
- Έλεγχοι μονάδας
- Έλεγχοι ολοκλήρωσης
- Πρόσθετοι έλεγχοι
  - Μορφοποίηση
  - Κάλυψη
  - Διάπτυξη (deployment)

Παράδειγμα: αστοχία CI

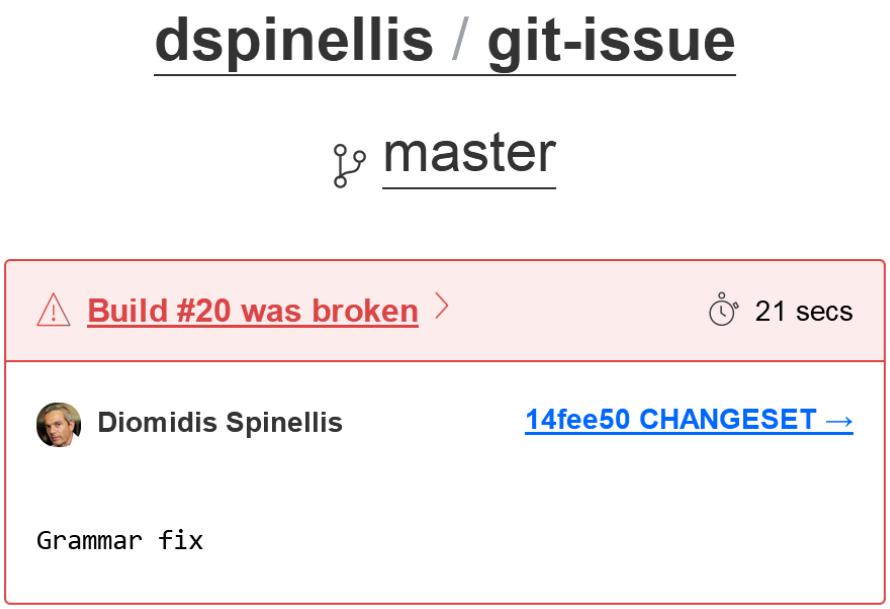

Figure 2: Μήνυμα αστοχίας κατά τη συνεχή ολοκλήρωση

Figure 3: Μήνυμα συνεχιζόμενης αστοχίας κατά τη συνεχή ολοκλήρωση

Παράδειγμα: αστοχία CI (2)

Παράδειγμα: διόρθωση CI

Υλοποίηση GitHub Actions

- Δημιουργούμε στο αποθετήριο ένα αρχείο YML κάτω από το .github/workflows με το εξής περιεχόμενο:

```
name: Java CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - name: Build with Maven
```

Figure 4: Μήνυμα διορθωμένης αστοχίας κατά τη συνεχή ολοκλήρωση

```
run: mvn -B package --file pom.xml
```

Πρόσθετοι πόροι

- Συνεργασία με pull request
- Επιθεώρηση κώδικα μέσω pull request
- GitHub Actions
- Kent Beck: Test Desiderata