JUnit

Satyendra Gurjar

February 25, 2015

Motivation

- JUnit was started in 2000[1]

[1] http://c2.com/cgi/wiki?TenYearsOfTestDrivenDevelopment

[2] http://blog.takipi.com/we-analyzed-30000-github-projects-here-are-the-top-100-libraries-in-java-js-ar

[3] http://junit.sourceforge.net/doc/testinfected/testing.htm

Motivation

- JUnit was started in 2000[1]
- A survey[2] performed in 2013 across 10,000 GitHub projects found that JUnit, along with slf4j-api, are the most popular libraries. Each library was used by 30.7% of projects.

[1] http://c2.com/cgi/wiki?TenYearsOfTestDrivenDevelopment

[2] http://blog.takipi.com/we-analyzed-30000-github-projects-here-are-the-top-100-libraries-in-java-js-ar

[3] http://junit.sourceforge.net/doc/testinfected/testing.htm

Motivation

- JUnit was started in 2000[1]
- A survey[2] performed in 2013 across 10,000 GitHub projects found that JUnit, along with slf4j-api, are the most popular libraries. Each library was used by 30.7% of projects.
- When I tried to use JUnit in 2005 and it didn't make any sense, most of the examples[3] of using JUnit were like this

[1] http://c2.com/cgi/wiki?TenYearsOfTestDrivenDevelopment

[2] http://blog.takipi.com/we-analyzed-30000-github-projects-here-are-the-top-100-libraries-in-java-js-ar

[3] http://junit.sourceforge.net/doc/testinfected/testing.htm

Actual example from JUnit 3.8 doc

```
public class Money {
    private int fAmount;
    private String fCurrency;

    public Money(int amount, String currency) {
        fAmount= amount; fCurrency= currency;
    }

    public int amount() {return fAmount;}

    public String currency() {return fCurrency;}

    public Money add(Money m) {
        return new Money (
          amount() + m.amount(), currency());
    }
}
```

And Test Case is

```
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");
        Money m14CHF= new Money(14, "CHF");
        Money expected= new Money(26, "CHF");
        Money result= m12CHF.add(m14CHF);
        Assert.assertTrue(expected.equals(result));
    }
}
```

- Real world applications are too complex to have such a simple testcase.

And Test Case is

```
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");
        Money m14CHF= new Money(14, "CHF");
        Money expected= new Money(26, "CHF");
        Money result= m12CHF.add(m14CHF);
        Assert.assertTrue(expected.equals(result));
    }
}
```

- Real world applications are too complex to have such a simple testcase.
- These examples failed to convey the point

So why its needed

1. provides testable specification of your code, for example

So why its needed

1. provides testable specification of your code, for example
   - what you expected a method to do when `null` is passed.

So why its needed

1. provides testable specification of your code, for example
    - what you expected a method to do when `null` is passed.
        - throw an exception,

So why its needed

1. provides testable specification of your code, for example
   - what you expected a method to do when `null` is passed.
     - throw an exception,
     - return error code etc.

So why its needed

1. provides testable specification of your code, for example
    - what you expected a method to do when `null` is passed.
        - throw an exception,
        - return error code etc.
    - what happens when invalid argument is passed

So why its needed

1. provides testable specification of your code, for example
    - what you expected a method to do when `null` is passed.
        - throw an exception,
        - return error code etc.
    - what happens when invalid argument is passed
    - what happens when a pre-condition before calling a method is not met

So why its needed

1. provides testable specification of your code, for example
   - what you expected a method to do when `null` is passed.
     - throw an exception,
     - return error code etc.
   - what happens when invalid argument is passed
   - what happens when a pre-condition before calling a method is not met
   - what happen when Database didnt respond, etc.

So why its needed

1. provides testable specification of your code, for example
    - what you expected a method to do when `null` is passed.
        - throw an exception,
        - return error code etc.
    - what happens when invalid argument is passed
    - what happens when a pre-condition before calling a method is not met
    - what happen when Database didnt respond, etc.

- we can document all this and but soon it will go out of sync and we would have no way of knowing current state of the code.

So why its needed (contd.)

2. Decoupling

[4] `http://c2.com/cgi/wiki?ExpensiveSetUpSmell`

So why its needed (contd.)

2. Decoupling
   - Is your code unit testable ?

[4] `http://c2.com/cgi/wiki?ExpensiveSetUpSmell`

So why its needed (contd.)

2. Decoupling
   - Is your code unit testable ?
   - Can we mock the dependecies and test a class in isolation ?

[4] `http://c2.com/cgi/wiki?ExpensiveSetUpSmell`

So why its needed (contd.)

2. Decoupling
    - Is your code unit testable ?
    - Can we mock the dependecies and test a class in isolation ?
    - from the Author of JUnit- “Reflect on your design practices. I have spent 8 years figuring out how to further decouple my objects to make them easier to test.”[4]

[4] http://c2.com/cgi/wiki?ExpensiveSetUpSmell

So why its needed (contd.)

2. Decoupling
   - Is your code unit testable ?
   - Can we mock the dependecies and test a class in isolation ?
   - from the Author of JUnit- “Reflect on your design practices. I have spent 8 years figuring out how to further decouple my objects to make them easier to test.”[4]
   - If we can’t test our code in isolation, there is something wrong with the design of our code it wont scale to changes and will lead to big “monolithic mess”.

[4] `http://c2.com/cgi/wiki?ExpensiveSetUpSmell`

The Monolithic Mess

```java
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?

The Monolithic Mess

```
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?
  - We need Database setup, namely

The Monolithic Mess

```
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?
  - We need Database setup, namely
    - for JDBC: User, Passwd, URL

The Monolithic Mess

```
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?
  - We need Database setup, namely
    - for JDBC: User, Passwd, URL
    - for DataSource: JNDI lookup, availability of App Server

The Monolithic Mess

```
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?
  - We need Database setup, namely
    - for JDBC: User, Passwd, URL
    - for DataSource: JNDI lookup, availability of App Server
- Issue:

The Monolithic Mess

```
class MyAPIHelper {
  Response getData(Request req) {
    // ...
    Account acct = DBHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

- How can we test `MyAPIHelper` ?
  - We need Database setup, namely
    - for JDBC: User, Passwd, URL
    - for DataSource: JNDI lookup, availability of App Server
- Issue:
  - `DBHelper.getAccount` is hardwired, we can't **mock** it and so we can't test our class in isolation.

Decoupling

- `static` method calls should be avoided, except for real utilities like `string.trimToNull` etc.

```
class MyAPIHelper {
                    // my default DBHelper
  DBHelper dbHelper = new DBHelper();

  // we can provide a new impl of DBHelper if needed.
  void setDbHelper(DBHelper dbh){dbHelper=dbh;}

  Response getData(Request req) {
    // ...
    Account acct = dbHelper.getAccount(req.getAcctId());
    // ...
    return new Response(acct);
  }
}
```

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`

[5] should be avoided

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`
- Which also makes our `MyAPIHelper` for extensible.

[5]should be avoided

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`
- Which also makes our `MyAPIHelper` for extensible.
- All the dependencies should be injected via

[5]should be avoided

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`
- Which also makes our `MyAPIHelper` for extensible.
- All the dependencies should be injected via
  - contructor or

[5] should be avoided

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`
- Which also makes our `MyAPIHelper` for extensible.
- All the dependencies should be injected via
  - contructor or
  - setters or

[5] should be avoided

Decoupling (contd.)

- Now we can provide a `mock` implementation of `DBHelper` in our unit test case of `MyAPIHelper`
- Which also makes our `MyAPIHelper` for extensible.
- All the dependencies should be injected via
  - contructor or
  - setters or
  - passed as argument to the function[5].

[5]should be avoided

Writing Testcases

- We use JUnit 4.x

Writing Testcases

- We use JUnit 4.x
- If you use maven, following needs to be added to `pom.xml`

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Writing Testcases

- We use JUnit 4.x
- If you use maven, following needs to be added to `pom.xml`

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- If you use `junit-4.x.jar` needs in the classpath.

Writing Testcases (contd.)

- Testcases must be included in the build script

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.
  - `TestNG` is a testing framework, inspired by `JUnit`, provides ways to run same testcase on multiple data.

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.
  - `TestNG` is a testing framework, inspired by `JUnit`, provides ways to run same testcase on multiple data.
- Create separate source dir for Test cases,

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.
  - `TestNG` is a testing framework, inspired by `JUnit`, provides ways to run same testcase on multiple data.
- Create separate source dir for Test cases,
  - `maven` does it by default, `src/test/java` dir

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.
  - `TestNG` is a testing framework, inspired by `JUnit`, provides ways to run same testcase on multiple data.
- Create separate source dir for Test cases,
  - `maven` does it by default, `src/test/java` dir
  - `ant` we need to create separate target for test compile and use different src dir than main src dir.

Writing Testcases (contd.)

- Testcases must be included in the build script
  - for maven, it included by default and `mvn package` runs all the test cases as part of build. we can run `mvn test` too
  - for ant, use `JUnit` task and `JUnitReport` task for generating test report.
- Testcases must not have any thing specific to developer's machine, no hardcoded paths to config file, use system properties.
- Test Data, can be included in the testcases itself.
  - `TestNG` is a testing framework, inspired by `JUnit`, provides ways to run same testcase on multiple data.
- Create separate source dir for Test cases,
  - `maven` does it by default, `src/test/java` dir
  - `ant` we need to create separate target for test compile and use different src dir than main src dir.
- Keep JEE component separate from business logic code. That is if we have EJB or WebService, write most of the code in Helper class.