COMP6700/2140 Test Driven Development

Steve Blackburn and Josh Milthorpe

Research School of Computer Science, ANU

31 March 2017

Test Driven Development (TDD)

TDD "red, green, refactor"

1. Create test that defines new requirements
2. Ensure test fails
3. Write code to support new requirement
4. Run tests to ensure code is correct
5. Then refactor and improve
6. Repeat

Key element of extreme programming

JUnit

Unit testing for Java

- Developed by Kent Beck
  – Father of extreme programming movement
- Integrated into IntelliJ
- Useful for:
  – TDD (Test driven development)
  – Bug isolation and regression testing
    - Precisely identify the bug with a unit test

JUnit

Unit testing for Java

- Developed by Kent Beck
  – Father of extreme programming movement
- Integrated into IntelliJ
- Useful for:
  – TDD (Test driven development)
  – Bug isolation and regression testing
    - Precisely identify the bug with a unit test
    - Use test to ensure that the bug is not reintroduced

JUnit (2)

- Methods marked with `@Test` will be tested
- When JUnit is called on a class, all tests are run and a report is generated.
  A failed test does not stop execution of subsequent tests.
- A rich set of *annotations* can be used to configure the testing environment, including:
  `@Test`, `@Ignore`, `@Before`, `@BeforeClass`, `@After`, `@AfterClass`
- *Assertion methods* test for expected behaviour:

```
assertEquals("Zhang San", user.name);
assertAll("name",
    () -> assertEquals("Li", address.getSurname()),
    () -> assertEquals("Si", address.getGivenName())
);
assertThrows(ArithmeticException.class, () -> {
    int x = 1 / 0;
});
```

Further Reading

- JUnit User Guide
- JUnit FAQ
- Wikipedia Test Driven Development
- Hortsmann *Core Java for the Impatient*, Ch.11 (Annotations)