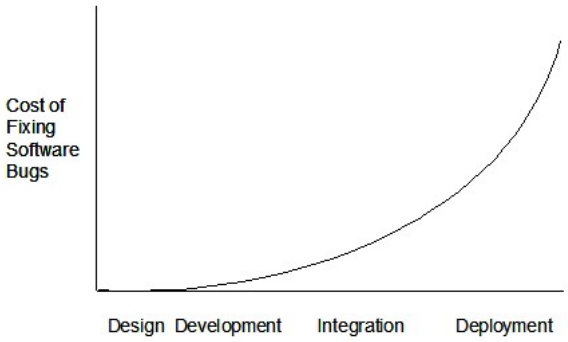
[L4P2]

Never Too Early to Test: an Introduction to Early Developer Testing

Developer testing

It seems logical to have the system built in its entirety before testing the system as a whole (also known as *system testing*). But what if a test case fails during system testing? Firstly, locating the cause of the failure is difficult due to a large search space; in a large system, the search space could be millions of lines of code, written by hundreds of developers! The failure may also be due to multiple inter-related bugs. Secondly, fixing the bug upon locating it could result in major rework, especially if the bug originated during the design or during requirements specification (i.e. a faulty design or faulty requirements). Furthermore, this process is costly and unpredictable. The whole team needs to work together to locate and fix bugs. One bug might 'hide' other bugs, which could emerge only after the first bug is fixed. Too many bugs found during system testing can lead to delivery delays. As illustrated by the graph on the right, the earlier a bug is found, the easier and cheaper to have it fixed. That is why developers need to start testing early, while the system is still under development. Such early testing done by developers is called developer testing.



Unit testing is one such form of early testing. Unit testing involves testing individual units (methods, classes, subsystems, …) and finding out whether each piece works correctly in isolation.

Automated unit testing

Unit testing may require *test drivers*. A test driver is a module written specifically for testing. A test driver's job is to invoke the SUT (Software Under Test) with test inputs. Test drivers are required for unit testing as most of the units do not have a UI. In the code example given below, PayrollTest is a test driver for the Payroll class that 'drives' the PayRoll class by sending it test inputs and printing out the output. However, it does not verify if the output is as expected.

```
public class PayrollTestDriver {
    public static void main(String[] args) {
        //test setup
        Payroll p = new Payroll();

        //test case 1
        p.setEmployees(new String[]{"E001", "E002"});
        print("Test 1 output " + p.totalSalary());

        //test case 2
        p.setEmployees(new String[]{"E001"});
        print("Test 2 output " + p.totalSalary());

        //more tests
        System.out.println("Testing completed");
    }
        ...
}
```

The PayrollTest class below not only drives the SUT Payroll class using test inputs, it also automatically verifies that output for each test input in as expected.

```
public class PayrollTestAtd {
    public static void main(String[] args) throws Exception {

        //test setup
        Payroll p = new Payroll();

        //test case 1
        p.setEmployees(new String[]{"E001", "E002"});
        // automatically verify the response
        if (p.totalSalary() != 6400) {
            throw new Error("case 1 failed ");
        }

        //test case 2
        p.setEmployees(new String[]{"E001"});
        if (p.totalSalary() != 2300) {
            throw new Error("case 2 failed ");
        }

        //more tests...

        System.out.println("All tests passed");
    }
}
```

JUnit is a tool for automated testing of Java programs. Similar tools are available for other languages.

Below is the code of an automated test for Payroll class, written using JUnit libraries.

```
public class PayrollTestJUnit {

    @Test
    public void testTotalSalary(){
        Payroll p = new Payroll();

        //test case 1
        p.setEmployees(new String[]{"E001", "E002"});
        assertEquals(p.totalSalary(), 6400);

        //test case 2
        p.setEmployees(new String[]{"E001"});
        assertEquals(p.totalSalary(), 2300);

        //more tests...
    }
}
```

Most modern IDEs come packaged with integrated support for testing tools. Figure 11 shows the JUnit output when running some JUnit tests using the Eclipse IDE.

Testability

Testability is an indication of how easy it is to test an SUT. As testability depends a lot on the design and implementation. We should try to increase the testability of our software when we design and implement them.

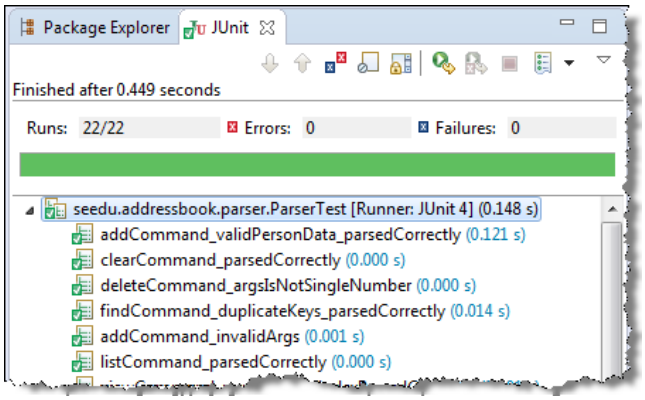


Figure 11. JUnit output on the Eclipse IDE

Test-Driven Development (TDD)

Typically, tests are written after writing the SUT. However, TDD advocates writing the tests *before* writing the SUT. In other words, first define the precise behavior of the SUT using test cases, and then write the SUT to match the specified behavior. While TDD has its fair share of detractors, there are many who consider it a good way to reduce defects. Note that TDD does not imply writing all the test cases first before writing functional code. Rather, proceed in small steps:

i. Decide what behavior to implement.
ii. Write test cases to test that behavior.
iii. Run those test cases and watch them fail.
iv. Implement the behavior.
v. Run the test case.
vi. Keep modifying the code and rerunning test cases until they all pass.
vii. Refactor code to improve quality.
viii. Repeat the cycle for each small unit of behavior that needs to be implemented.

Worked examples

[Q1]
Discuss advantages and disadvantages of developers testing their own code.

[A1]
Advantages:

- Can be done early (the earlier we find a bug, the cheaper it is to fix).
- Can be done at lower levels, for examples, at operation and class level (testers usually test the system at UI level).
- It is possible to do more thorough testing since developers know the expected external behavior as well as the internal structure of the component.
- It forces developers to take responsibility for their own work (they cannot claim that "testing is the job of the testers").

Disadvantages:

- Developer may unconsciously test only situations that he knows to work (i.e. test it too "gently").
- Developer may be blind to his own mistakes (if he did not consider a certain combination of input while writing code, he is likely to miss it again during testing).
- Developer may have misunderstood what the SUT is supposed to do in the first place.
- Developer may lack the testing expertise.

-- End of Handout --