Unit Testing Testing
Plan

• Java
  • jUnit (test cases)
• Emma (coverage)
• C
  • Check (test cases)
  • Splint (static checking)
JUnit

• Testing harness for Java

• Enables quick and simple test creation, execution, and evaluation

• Tests can be scattered in code or organized into separate suites

• [http://www.junit.org/](http://www.junit.org/)
JUnit: Anatomy of a test case

Annotation

@Test public void testAdd(){
  int i = SimpleProg.add_nums(5,4);
  assertEquals(9,i);
}

Test Function Definition

Function Name

Performs the actual test. Up until this point we are only making sure the application does not crash.
jUnit: Anatomy of a test case

Lots of other asserts

assertEquals  assertFalse  assertNot_Null  assertNotSame
assertNullassertSameassertTruelfailfailNotEquals failNotSame
failSame

@Test  public  void testAdd(){
    int  i  =  SimpleProg.add_nums(5,4);
    assertEquals(9,i);
}

Performs the actual test. Up until this point we are only making sure the application does not crash.
import org.junit.*;
import static org.junit.Assert.*;
public class testSimpleProg{
  @Before public void setUp() {
    //Set up the environment
    System.out.println("SETTING UP");
  }
  @After public void tearDown() {
    //Clean up the environment
    System.out.println("CLEANING UP");
  }
  @Test public void testAdd(){
    int i = SimpleProg.add_nums(5,4);
    assertEquals(9,i);
  }
  @Test public void testAddNot(){
    int i = SimpleProg.add_nums(5,5);
    assertTrue("I is wrong",i!=9);
  }
}
Running jUnit

• Compiling with jUnit
  • javac -cp junit-4.5.jar:. *.java

• Running with jUnit
  • java -cp junit-4.5.jar:. org.junit.runner.JUnitCore testSimpleProg
Emma: Code Coverage

• Emma is a free code coverage tool for Java
• Code can be instrumented using Emma or executed by the emma runner
What is Coverage?

- An indication of how much we are testing
- Having good code coverage does not mean we have good tests
What is Coverage?

Types of Coverage

- Class
- Function
- Block
- Statement
- Branch
- Path

An indication of how much we are testing. Having good code coverage does not mean we have good tests.
Running Emma

```
java -cp emma.jar:junit-4.5.jar:. emmarun -sp . \
-ix +SimpleProg -r html -cp junit-4.5.jar:. \norg.junit.runner.JUnitCore testSimpleProg
```
What does Emma get us?
Unit Testing C: Check

• Check is a unit testing framework, similar to jUnit

• Idea is the same:
  • Make test cases
  • Compile them in
  • User a test runner to execute them

Anatomy of a Check Unit Test

START_TEST (test_add)
{
    int i = addNums(5,4);
    fail_unless(i==9, "i not set correctly");
}
END_TEST

Performs the actual test. Up until this point we are only making sure the application does not crash.
# Anatomy of a Test Suite

**Included Headers**

```c
#include <stdlib.h>
#include <check.h>
```

**Test Cases**

```c
START_TEST (test_add)
{
    int i = addNums(5,4);
    fail_unless(i==9, "i not set correctly");
}
END_TEST
```

```c
Suite *money_suite (void)
{
    Suite *s = suite_create ("Simple Check");
    /* Core test case */
    TCase *tc_core = tcase_create ("Core");
    tcase_add_test (tc_core, test_add);
    suite_add_tcase (s, tc_core);
    return s;
}
```

**Test Runner**

```c
int main (void)
{
    int number_failed;
    Suite *s = money_suite ();
    SRunner *sr = srunner_create (s);
    srunner_run_all (sr, CK_NORMAL);
    number_failed = srunner_ntests_failed (sr);
    srunner_free (sr);
    return (number_failed == 0) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```
Running Check

• Compiling with Check
  • gcc -o test testHello.c hello.c \\ check-0.9.5/src/*.o -Icheck-0.9.5/src

• Run the application as normal
  • test

Output:

Running suite(s): Simple Check
100%: Checks: 1, Failures: 0, Errors: 0
Static Checking with Splint

• Splint checks individual source files before compilation

• Can find a large variety of bugs and point developers to problem areas

• Has problems with finding false positives and thus making complex code to work with Splint can be difficult

• Users can annotate their code to help Splint to make certain assumptions

• http://www.splint.org/
Sample Code

```c
01: char reflect(char *s){
02:   if(s == NULL){return '\0';}
03:   return *s;
04: }
05:
06: int main(){
07:   int c;
08:   int d = c+1;
09:   char s;
10:   char *tmp = malloc(sizeof(char)*10);
11:   s = reflect(tmp);
12:   strcpy(tmp, "Hello");
13:   printf("Value of C: %d\n", d);
14:   printf("Value of tmp: %s\n", tmp);
15:   return 0;
16: }
```
Sample Splint Output 1

splint hello2.c
Splint 3.1.1 --- 07 Dec 2007
hello2.c: (in function main)
hello2.c:8:11: Variable c used before definition
  An rvalue is used that may not be initialized to a value on some execution path. (Use -usedef to inhibit warning)
hello2.c:11:16: Possibly null storage tmp passed as non-null param:
  reflect (tmp)
  A possibly null pointer is passed as a parameter corresponding to a formal parameter with no /*@null@*/ annotation. If NULL may be used for this parameter, add a /*@null@*/ annotation to the function parameter declaration. (Use -nullpass to inhibit warning)
hello2.c:10:17: Storage tmp may become null
hello2.c:11:16: Passed storage tmp not completely defined (*tmp is undefined):
  reflect (tmp)
  Storage derivable from a parameter, return value or global is not defined. Use /*@out@*/ to denote passed or returned storage which need not be defined. (Use -comppdef to inhibit warning)
hello2.c:10:41: Storage *tmp allocated

..........
hello2.c:15:12: Fresh storage tmp not released before return
  A memory leak has been detected. Storage allocated locally is not released
  before the last reference to it is lost. (Use -mustfreenew to inhibit
  warning)
hello2.c:10:41: Fresh storage tmp created
hello2.c:1:6: Function exported but not used outside hello2: reflect
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
hello2.c:4:1: Definition of reflect

Finished checking --- 5 code warnings
zsh: exit 1 splint hello2.c
Questions