Software Testing
Trying Hard to Make Software Fail

Ludovic Courtès
Service d’expérimentation & développement

7 June 2011
Introduction

Practicalities
- Running Tests
- What to Test?
- When Write Tests?

Beyond
Why Test?

- to find & isolate bugs
- to make sure bugs get fixed
- to make sure bugs don't come back

"Faults in Linux: Ten Years Later", Palix et. al, 2010
http://faultlinux.lip6.fr/

Figure 5: Faults in Linux 2.6.0 to 2.6.33

“Faults in Linux: Ten Years Later”, Palix et. al, 2010
http://faultlinux.lip6.fr/
Why Test?

- to find & isolate bugs
- to make sure bugs get fixed
- to make sure bugs don't come back

"Faults in Linux: Ten Years Later", Palix et. al, 2010
http://faultlinux.lip6.fr/

Figure 5: Faults in Linux 2.6.0 to 2.6.33

"Faults in Linux: Ten Years Later", Palix et. al, 2010
http://faultlinux.lip6.fr/
Why Test?

- to find & isolate bugs

"Faults in Linux: Ten Years Later", Palix et. al, 2010
http://faultlinux.lip6.fr/
Why Test?

- to find & isolate bugs
- to make sure bugs get fixed

"Faults in Linux: Ten Years Later", Palix et. al, 2010

http://faultlinux.lip6.fr/
Why Test?

- to find & isolate bugs
- to make sure bugs get fixed
- to make sure bugs don’t come back
1 Introduction

2 Practicalities
- Running Tests
- What to Test?
- When Write Tests?

3 Beyond
“Determining success or failure of tests must be an automatic process.”
— B. Meyer, Principle #4
1 Introduction

2 Practicalities
   - Running Tests
   - What to Test?
   - When Write Tests?

3 Beyond
Automake Rules

configure.ac

AM_INIT_AUTOMAKE([ \ color-tests \ parallel-tests])

Makefile.am

TESTS = foo bar baz
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \
color-tests \nparallel-tests])

Makefile.am

TESTS = foo bar baz

cHECK_PROGRAMS = $(TESTS)
Automake Rules

http://gnu.org/software/automake/

**configure.ac**

```plaintext
AM_INIT_AUTOMAKE([ \
  color-tests \ 
  parallel-tests])
```

**Makefile.am**

```plaintext
TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \ 
  valgrind -q \ 
  --leak-check=full \ 
  --error-exitcode=1
```
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \n  color-tests \n  parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \n  valgrind -q \n  --leak-check=full \n  --error-exitcode=1

foo.c

#include <stdlib.h>

int main ()
{
    if (it_works ())
        return EXIT_SUCCESS;
    else
        return EXIT_FAILURE;
}
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \n    color-tests \n    parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \n    valgrind -q \n    --leak-check=full \n    --error-exitcode=1

foo.c

#include <stdlib.h>

int main ()
{
    #ifdef CAN_RUN_TEST
        if (it_works ())
            return EXIT_SUCCESS;
        else
            return EXIT_FAILURE;
    #else
        /* Skip this test. */
        return 77;
    #endif
}

$ make check
CCLD foo
CC bar.o
CC lib.o
CCLD bar
CCLD baz
PASS: foo
FAIL: bar
SKIP: baz
=====================
1 of 3 tests failed
(1 test was not run)
See test-suite.log
=====================
$ make recheck
FAIL: bar
=====================
1 of 1 test failed
=====================
$ make recheck
CCLD bar
PASS: bar
===================== 1 test passed
=====================
Automake Rules

http://gnu.org/software/automake/

$ make check

configure.ac

AM_INIT_AUTOMAKE([ \
    color-tests \n    parallel-tests])

Makefile.am

TESTS = foo bar baz

cHECK_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \n    valgrind -q \n    --leak-check=full \n    --error-exitcode=1

$ make check

CCLD foo
CC bar.o
CC lib.o
CCLD bar
CCLD baz
PASS: foo
FAIL: bar
SKIP: baz

===================== 1 of 3 tests failed
(1 test was not run)
See test-suite.log

$ make recheck

FAIL: bar

===================== 1 of 1 test failed

$ make vi bar.c

$ make recheck

CCLD bar
PASS: bar

===================== 1 test passed

$ make check-html
Automake Rules

configure.ac

AM_INIT_AUTOMAKE([ \
    color-tests \
    parallel-tests])

Makefile.am

TESTS = foo bar baz

cHECK_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \ 
    valgrind -q \ 
    --leak-check=full \ 
    --error-exitcode=1

$ make check
  CCLD foo
  CC bar.o
  CC lib.o
  CCLD bar
  CCLD baz
  PASS: foo
  FAIL: bar
  SKIP: baz

=====================  
1 of 3 tests failed  
(1 test was not run)  
See test-suite.log

=====================  

$ make recheck
  FAIL: bar

=====================  
1 of 1 test failed  

=====================  

$ make recheck
  CCLD bar
  PASS: bar

=====================  
1 test passed  

=====================  

$ make check-html
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \color-tests \parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \valgrind -q \--leak-check=full \--error-exitcode=1

$ make recheck
FAIL: bar

================================
1 of 1 test failed

================================
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \
  color-tests \
  parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \
  valgrind -q \
  --leak-check=full \
  --error-exitcode=1

$ make recheck
FAIL: bar

$ vi bar.c

$ make recheck

$ make recheck

FAIL: bar

$ make recheck

1 of 1 test failed

$ make recheck

1 test passed

$ make check-html
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \
  color-tests \n  parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \ 
  valgrind -q \ 
  --leak-check=full \ 
  --error-exitcode=1

$ make recheck
FAIL: bar
=================================
1 of 1 test failed
=================================

$ vi bar.c

$ make recheck
CCLD bar
PASS: bar
=================================
1 test passed
=================================

$ make check-html
Automake Rules

http://gnu.org/software/automake/

configure.ac

AM_INIT_AUTOMAKE([ \n    color-tests \n    parallel-tests])

Makefile.am

TESTS = foo bar baz

check_PROGRAMS = $(TESTS)

TESTS_ENVIRONMENT = \n    valgrind -q \n    --leak-check=full \n    --error-exitcode=1

$ make recheck
FAIL: bar

$ vi bar.c

$ make recheck
CCLD bar
PASS: bar
1 test passed

$ make check-html
CMake Lists (?)
http://cmake.org/

CMakeLists.txt

cmake_minimum_required (VERSION 2.8)
project (sed)

enable_testing ()

add_test (my-test1 \${EXECUTABLE_OUTPUT_PATH}/my-test1)
add_test (my-test2 \${EXECUTABLE_OUTPUT_PATH}/my-test2)
CMake
http://cmake.org/

CMakeLists.txt

`cmake_minimum_required (VERSION 2.8)`
`project (sed)`

`add_executable (my-test1 my-test1.c)`
`add_executable (my-test2 my-test2.c)`

`enable_testing ()`

`add_test (my-test1 ${EXECUTABLE_OUTPUT_PATH}/my-test1)`
`add_test (my-test2 ${EXECUTABLE_OUTPUT_PATH}/my-test2)`

$ cmake .
$ make
$ make test

Running tests...
Test project /home/ludo/src/cmake/t1
Start 1: my-test1 .......... Passed 0.00 sec
Start 2: my-test2 ..........***Failed 0.00 sec
50% tests passed, 1 tests failed out of 2
Total Test time (real) = 0.03 sec
The following tests FAILED:
2 - my-test2 (Failed)
$ cmake .
-- The C compiler identification is GNU
...

$ make
[ 50%] Building C object .../my-test1.c.o
Linking C executable my-test1
...
CMake
http://cmake.org/

$ make test
Running tests...
Test project /home/ludo/src/cmake/t1
  Start 1: my-test1
  1/2 Test #1: my-test1 .......... Passed 0.00 sec
  Start 2: my-test2
  2/2 Test #2: my-test2 ..........***Failed 0.00 sec

50% tests passed, 1 tests failed out of 2

Total Test time (real) = 0.03 sec

The following tests FAILED:
  2 - my-test2 (Failed)
Check
http://check.sf.net/

#include <check.h>
#include <stdlib.h>
#include <string.h>

START_TEST (test_concat)
{
    char dst[123];

    strcpy (dst, "hel");
    strcat (dst, "lo");
    fail_if (strcmp (dst, "hello"),
             "strcat is broken");
}

END_TEST

Suite *make_test_suite (void)
{
    Suite *s = suite_create ("strings");
    TCase *tc = tcase_create ("concat");
    tcase_add_test (tc, test_concat);
    suite_add_tcase (s, tc);
    return s;
}

int main ()
{
    int nrfailed;
    Suite *s = make_test_suite ();
    SRunner *sr = srunner_create (s);
    srunner_run_all (sr, CK_ENV);
    nrfailed = srunner_ntests_failed (sr);
    srunner_free (sr);
    return (nrfailed == 0) ? EXIT_SUCCESS : EXIT_FAILURE;
}

$ gcc my-test-suite.c \
   -lcheck \
   -o test $
$ ./test
Running suite(s): strings
100%: Checks: 1, Failures: 0, Errors: 0

$ CK_VERBOSITY=verbose ./test-suite
Running suite(s): strings
50%: Checks: 2, Failures: 0, Errors: 1
    test.c:20:P:concat:test_concat:0: Passed
    test.c:12:E:segfault:test:0: (after this point) Received signal 11 (Segmentation fault)

$ CK_FORK=no ./test-suite
Running suite(s): strings
Segmentation fault
Suite *make_test_suite (void) {
    Suite *s = suite_create ("strings");
    TCase *tc = tcase_create ("concat");
    tcase_add_test (tc, test_concat);
    suite_add_tcase (s, tc);
    return s;
}

int main () {
    int nrfailed;
    Suite *s = make_test_suite ();
    Srunner *sr = srunner_create (s);
    srunner_run_all (sr, CK_ENV);
    nrfailed = srunner_ntests_failed (sr);
    srunner_free (sr);
    return (nrfailed == 0) \
        ? EXIT_SUCCESS : EXIT_FAILURE;
Check

http://check.sf.net/

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

START_TEST (test_concat)
{
    char dst[123];

    strcpy (dst, "hel");
    strcat (dst, "lo");
    fail_if (strcmp (dst, "hello"), "strcat is broken");
}

END_TEST
```

```bash
$ gcc my-test-suite.c -lcheck -o test
$ ./test
Running suite(s): strings
100%: Checks: 1, Failures: 0, Errors: 0

$ CK_VERBOSITY=verbose ./test-suite
Running suite(s): strings
50%: Checks: 2, Failures: 0, Errors: 1
test.c:20:P:concat:testconcat:0: Passed
test.c:12:E:segfault:testsegfault:0: (after this point) Received signal 11 (Segmentation fault)

$ CK_FORK=no ./test-suite
Running suite(s): strings
Segmentation fault
```
START_TEST (test_concat) {
    char dst[123];

    strcpy (dst, "hel");
    strcat (dst, "lo");
    fail_if (strcmp (dst, "hello"), "strcat is broken");
}
END_TEST
#include <check.h>
#include <stdlib.h>
#include <string.h>

START_TEST (test_segfault)
{
    static const char rostr[] = "hi!";
    strcpy (rostr, "hey!");
    fail ("never reached");
}
END_TEST

$ CK_VERBOSITY=verbose ./test-suite

Running suite(s): strings
50%: Checks: 2, Failures: 0,
test.c:20:P:concat:test_concat

test.c:12:E:segfault:test_segfault
#include <check.h>
#include <stdlib.h>
#include <string.h>

START_TEST (test_segfault)
{
    static const char rostr[] = "hi!";
    strcpy (rostr, "hey!");
    fail ("never reached");
}
END_TEST

$ CK_VERBOSITY=verbose ./test-suite
Running suite(s): strings
50%: Checks: 2, Failures: 0, Errors: 1
test.c:20:P:concat:test_concat:0: Passed

test.c:12:E:segfault:test_segfault:0: (after this point) Received signal 11 (Segmentation fault)

$ CK_FORK=no ./test-suite
Running suite(s): strings
Segmentation fault
In Other Languages

- **Java** → JUnit → http://www.junit.org/
- **C++** → CPPUnit → http://cppunit.sf.net/
- **Scheme** → SRFI-64 → http://srfi.schemers.org/srfi-64/
- ...

1 Introduction

2 Practicalities
   - Running Tests
   - What to Test?
   - When Write Tests?

3 Beyond
Test Granularity

- **Unit testing** is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. In object-oriented programming a unit is usually a method. Unit tests are created by programmers or occasionally by white box testers during the development process.
Test Granularity

- **Unit testing** is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. In object-oriented programming a unit is usually a method. Unit tests are created by programmers or occasionally by white box testers during the development process.

- **Integration testing** (sometimes called Integration and Testing, abbreviated ”I&T”) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
Test Granularity

- **Unit testing** is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. In object-oriented programming a unit is usually a method. Unit tests are created by programmers or occasionally by white box testers during the development process.

- **Integration testing** (sometimes called Integration and Testing, abbreviated "I&T") is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

- **System testing** of software or hardware is testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.
**Test Granularity**

- **Unit testing** is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. In object-oriented programming a unit is usually a method. Unit tests are created by programmers or occasionally by white box testers during the development process.

- **Integration testing** (sometimes called Integration and Testing, abbreviated ”I&T”) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

- **System testing** of software or hardware is testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.
Unit Test of `rm`

GNU Coreutils, tests/rm/f-1

```
#!/bin/sh

# Test "rm -f" with a nonexistent file.

mkdir -p d || framework_failure

rm -f d/no-such-file || fail=1

Exit $fail
```

Spec

`rm -f` returns 0 (success) even for a non-existent file.
Unit Test of the \texttt{inet_ntop} Function

GNU C Library, resolv/tst-inet_ntop.c

```c
int main ()
{
    struct in_addr addr4; char buf[64];
    int failed = 0;

    addr4.s_addr = 0xe0e0e0e0e0;
    memset (buf, 'x', sizeof buf);

    if (inet_ntop (AF_INET, &addr4, buf, 16) != buf)
        failed++;
    if (memcmp (buf, "224.224.224.224" "xxxxxxxx", 24) != 0)
        failed++;

    return failed;
}
```

Spec

Convert the IPv4 address to text.

Return ENOSPC if the buffer is inadequate.
Unit Test of the `inet_ntop` Function
GNU C Library, `resolv/tst-inet_ntop.c`

```c
int main () {
    struct in_addr addr4; char buf[64];
    int failed = 0;

    addr4.s_addr = 0xe0e0e0e0;
    memset (buf, 'x', sizeof buf);

    if (inet_ntop (AF_INET, &addr4, buf, 15) != NULL)  
        failed++;
    else if (errno != ENOSPC)  
        failed++;

    return failed;
}
```

Spec:
Return ENOSPC if the buffer is inadequate.
Unit/Integration(?) Test of GCC

gcc/testsuite/c-c++-common/Wunused-var-1.c

void
f1 (void)
{
    int a; /* { dg-warning "set but not used" } */
    int b;
    int c;
    c = 1;
    a = b = c;
}
OK, what if my program deals with the hardware?
OK, what if my program deals with the hardware? Screwed?
Unit Test of a Disk Partitioning Tool

GNU Parted, libparted/tests/disk.c

disk = _create_disk_label (dev, ped_disk_type_get("ms"));
part = ped_partition_new (disk, PED_PARTITION_EXTENDED, NULL, 32, 29311);
ped_disk.add_partition (disk, part, constraint);

part = ped_partition_new (disk, PED_PARTITION_LOGICAL, ped_file_system_type_get("ext2"), 19584, 29311);
ped_disk.add_partition (disk, part, constraint);

ped_disk.commit (disk);
disk.dup = ped_disk_duplicate (disk);

/* Check if partitions match. */
for (int *i = part.num; *i != 0; i++) {
    part = ped_disk.get_partition (disk, *i);
    part.dup = ped_disk.get_partition (disk.dup, *i);
    fail_if (part->geom.start != part.dup->geom.start ||
             part->geom.end != part.dup->geom.end,
             "Duplicated partition doesn’t match.");
}
Key Ingredients to Test Anything

- abstract interfaces—e.g., hard disk in Parted
Key Ingredients to Test Anything

- abstract interfaces—e.g., hard disk in Parted
- “mocks”—dummy implementations to simulate real objects
Using Mocks to Test OO Programs

```java
public interface Turtle {
    void penDown();
    void penUp();
    void gotoLeft(int distance);
    void gotoRight(int distance);
    void gotoUp(int distance);
    void gotoDown(int distance);
}
```

```java
import junit.framework.TestCase;
import org.junit.Test;
import org.easymock.EasyMock;
public class TurtleTest extends TestCase {
    @Test public void testPenTakenAndReleased() throws Exception {
        ...
        Turtle mock = EasyMock.createMock(Turtle.class);
        EasyMock.checkOrder(mock, true);
        mock.penDown();
        EasyMock.checkOrder(mock, false);
        mock.gotoLeft(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoRight(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoDown(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoUp(EasyMock.anyInt());
        EasyMock.checkOrder(mock, true);
        mock.penUp();
        EasyMock.replay(mock);
        drawSomething(mock);
        EasyMock.verify(mock);
    }
}
```
UsingMocks to Test OO Programs

```java
import junit.framework.TestCase;
import org.junit.Test;
import org.easymock.EasyMock;

public class TurtleTest extends TestCase {
    @Test public void testPenTakenAndReleased() throws Exception {
        ...  
```
UsingMocks to Test OO Programs

Turtle mock = EasyMock.createMock(Turtle.class);
EasyMock.checkOrder(mock, true);
mock.penDown();

Spec

drawSomething() calls penDown() first, then draws, and finally calls penUp()

mock.penUp();

EasyMock.replay(mock);
drawSomething(mock);

EasyMock.verify(mock);

expected calls

method under test

expectations met?
Using Mocks to Test OO Programs

```java
import junit.framework.TestCase;
import org.junit.Test;
import org.easymock.EasyMock;

public class TurtleTest extends TestCase {
    @Test
    public void testPenTakenAndReleased() throws Exception {
        Turtle mock = EasyMock.createMock(Turtle.class);
        EasyMock.checkOrder(mock, true);
        mock.penDown();
        EasyMock.checkOrder(mock, false);
        mock.gotoLeft(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoRight(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoDown(EasyMock.anyInt());
        EasyMock.expectLastCall().anyTimes();
        mock.gotoUp(EasyMock.anyInt());
        EasyMock.checkOrder(mock, true);
        mock.penUp();
        EasyMock.replay(mock);
        drawSomething(mock);
        EasyMock.verify(mock);
    }
}
```
Mocks in Various Languages

- EasyMock (Java) → http://easymock.org/
- Google Mocks (C++) → https://code.google.com/p/gogolemock/
- OCMock (Objective-C) → http://www.mulle-kybernetik.com/software/OCMock/
- CMock (C) → http://cmock.sf.net/
- ...
But my program requires root privileges, is distributed, depends on a kernel module, blah blah blah...
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix

{ pkgs, ... }:
with pkgs;
{
    nodes = {
        master =
            args: {
                environment . systemPackages = [ gcc mpich2 ];
            },
        slave =
            args: {
                environment . systemPackages = [ gcc mpich2 ];
            },
    };
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix
{ pkgs, ... }:
with pkgs;

{

  nodes = {
    master =
      args: {
        environment .systemPackages = [ gcc mpich2 ];
        boot.kernelPackages = pkgs.linuxPackages_2_6_32;
      };
    }

  slave =
    args: {
      environment .systemPackages = [ gcc33 mpich2 ];
      boot.kernelPackages = pkgs.linuxPackages_2_6_29;
    };
  }

...
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix

... testScript = ‘’ startAll;

$master->mustSucceed("mpd --daemon --ifhn=master \ 
   --listenport=4444");
$slave->mustSucceed("mpd --daemon --host=master \ 
   --port=4444");

';
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix

... testScript = 'startAll;

$master->mustSucceed("mpd --daemon --ifhn=master \
    --listenport=4444");
$slave->mustSucceed("mpd --daemon --host=master \
    --port=4444");

$master->mustSucceed("mpicc -o example -Wall \
    ${./mpich-example.c}");
$slave->mustSucceed("mpicc -o example -Wall \
    ${./mpich-example.c}");

'}
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix

```nix
... testScript = 
  
  "$master->mustSucceed("mpd --daemon --ifhn=master \ 
    --listenport=4444");
  $slave->mustSucceed("mpd --daemon --host=master \ 
    --port=4444");

  $master->mustSucceed("mpicc -o example -Wall \ 
    ${./mpich-example.c}");
  $slave->mustSucceed("mpicc -o example -Wall \ 
    ${./mpich-example.c}");

  $master->mustSucceed("mpiexec -n 2 ./example");
};
```
Distributed Tests Using NixOS Virtual Machines

http://nixos.org/nixos/ — tests/mpich.nix

...  

testScript =  
'' startAll;

$master->mustSucceed("mpd --daemon --ifhn=master \  
--listenport=4444");

automated, “distributed”, reproducible integration test

$master->mustSucceed("mpicc -o example -Wall \  
${./mpich-example.c}");

$slave->mustSucceed("mpicc -o example -Wall \  
${./mpich-example.c}");

$master->mustSucceed("mpiexec -n 2 ./example");

''
What’s left to test?
Test Coverage
Using GCC & LCOV — http://ltp.sf.net/coverage/

$ make clean
$ make CFLAGS=--coverage LDFLAGS=--coverage
...
$ make check CFLAGS=--coverage LDFLAGS=--coverage
...
Test Coverage
Using GCC & LCOV — http://ltp.sf.net/coverage/

$ make clean
$ make CFLAGS=--coverage LDFLAGS=--coverage
...
$ make check CFLAGS=--coverage LDFLAGS=--coverage
...
$ lcov --capture --directory . > my-lib.info
...
Overall coverage rate:
  lines.......: 67.2% (5956 of 8861 lines)
  functions..: 78.3% (455 of 581 functions)
  branches....: 43.2% (2289 of 5304 branches)
Test Coverage
Using GCC & LCOV — http://ltp.sf.net/coverage/

$ make clean
$ make CFLAGS=--coverage LDFLAGS=--coverage
...$
$ make check CFLAGS=--coverage LDFLAGS=--coverage
...
$ lcov --capture --directory . > my-lib.info
...
Overall coverage rate:
  lines.......: 67.2% (5956 of 8861 lines)
  functions..: 78.3% (455 of 581 functions)
  branches....: 43.2% (2289 of 5304 branches)
$ genhtml -o out my-lib.info
Test Coverage
Using GCC & LCOV — http://ltp.sf.net/coverage/

$ make clean
$ make CFLAGS=--coverage LDFLAGS=--coverage
...
$ make check CFLAGS=--coverage LDFLAGS=--coverage
...
$ lcov --capture --directory . > my-lib.info
...
Overall coverage rate:
lines......: 67.2% (5956 of 8861 lines)
functions..: 78.3% (455 of 581 functions)
branches...: 43.2% (2289 of 5304 branches)

$ genhtml -o out my-lib.info

LCOV - code coverage report

Current view: top level - libguile
Test: unnamed
Date: 2011-03-30

<table>
<thead>
<tr>
<th>Filename</th>
<th>Line Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>alist.c</td>
<td>100.0%</td>
<td>100.0%</td>
<td>85.0%</td>
</tr>
<tr>
<td>alist.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>arbitors.c</td>
<td>100.0%</td>
<td>100.0%</td>
<td>66.7%</td>
</tr>
<tr>
<td>arbitors.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>array-handle.c</td>
<td>92.9%</td>
<td>100.0%</td>
<td>75.0%</td>
</tr>
<tr>
<td>array-map.c</td>
<td>36.0%</td>
<td>44.4%</td>
<td>28.6%</td>
</tr>
<tr>
<td>array-map.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>arrays.c</td>
<td>64.0%</td>
<td>76.0%</td>
<td>48.1%</td>
</tr>
<tr>
<td>arrays.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>async.c</td>
<td>52.7%</td>
<td>44.0%</td>
<td>33.3%</td>
</tr>
<tr>
<td>async.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>backtrace.c</td>
<td>64.2%</td>
<td>55.0%</td>
<td>50.0%</td>
</tr>
<tr>
<td>backtrace.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>bitvectors.c</td>
<td>43.5%</td>
<td>72.7%</td>
<td>25.3%</td>
</tr>
<tr>
<td>bitvectors.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>boolean.c</td>
<td>85.7%</td>
<td>80.0%</td>
<td>87.5%</td>
</tr>
<tr>
<td>boolean.x</td>
<td>100.0%</td>
<td>-</td>
<td>0/0</td>
</tr>
<tr>
<td>bitvectors.c</td>
<td>84.7%</td>
<td>91.5%</td>
<td>57.5%</td>
</tr>
</tbody>
</table>

Overall coverage:
Lines: 24315/31939 (76.1%)
Functions: 2129/3012 (70.7%)
Branches: 13842/24858 (55.7%)
Test Coverage

Using GCC & LCOV — http://ltp.sf.net/coverage/

$ make clean
$ make CFLAGS=--coverage LDFLAGS=--coverage
...
$ make check CFLAGS=--coverage LDFLAGS=--coverage
...
$ lcov --capture --directory . > my-lib.info
...
Overall coverage rate:
- lines......: 67.2% (5956 of 8861 lines)
- functions..: 78.3% (455 of 581 functions)
- branches...: 43.2% (2289 of 5304 branches)

$ genhtml -o out my-lib.info
Test-Driven Development

Write the tests & doc before the code.
Test-Driven Development

(Re)Write a test

Test succeeds

Check if the test fails

Test fails

Write production code

Test(s) fail

Run all tests

All tests succeed

Clean up code

Repeat

From Wikipedia.
Regression Testing

Commit a test case along with each bug fix.
Regression Testing in Action

bug fix for `strtok` in the GNU C Library, 2001-02-21

--- a/sysdeps/i386/i686/strtok.S
+++ b/sysdeps/i386/i686/strtok.S
@@ -123,6 +123,8 @@ ENTRY (BP (FUNCTION))
    the last run. */
   cmpl $0, %edx
   cmove %eax, %edx
+  testl %edx, %edx
+  jz L(returnNULL)
#if __BOUNDED_POINTERS__
 # ifdef USE_AS_STRTOK_R
   movl SAVE(%esp), %ecx /* borrow %ecx for a moment */
Regression Testing in Action

bug fix for `strtok` in the GNU C Library, 2001-02-21

```c
--- /dev/null
+++ b/string/tst-strtok.c
@@ -0,0 +1,23 @@
+ #include <stdio.h>
+ #include <string.h>
+
+ int main (void) {
+ char buf[1] = { 0 };
+ int result = 0;
+
+ if (strtok (buf, " ") != NULL)
+    result = 1;
+ else if (strtok (NULL, " ") != NULL)
+    result = 1;
+
+ return result;
+}
```
Icing on the Cake

Use Continuous Integration!
Icing on the Cake

Use Continuous Integration!

- informed as soon as something breaks
- know which commit broke what
Icing on the Cake

Use Continuous Integration!

- informed as soon as something breaks
- know which commit broke what

→ Soon available at your local SED!
1 Introduction

2 Practicalities
   - Running Tests
   - What to Test?
   - When Write Tests?

3 Beyond
“Testing is about producing failures.”

“A testing strategy’s most important property is the number of faults it uncovers as a function of time.”

— B. Meyer, Principle #7
Are manual tests good enough?
What is their bug-finding performance?
Example: bugs in GCC and LLVM

- $\approx 75\text{–}80\%$ of function/line coverage
Example: bugs in GCC and LLVM

Random testing with Csmith — http://embed.cs.utah.edu/csmith/

- \( \approx 75\text{–}80\% \) of function/line coverage
- randomly generated programs uncover 100+ of bugs
Example: bugs in GCC and LLVM

Random testing with Csmith — http://embed.cs.utah.edu/csmith/

- ≈ 75–80% of function/line coverage
- randomly generated programs uncover 100+ of bugs
- ... without noticeably increasing function/line coverage
The Needle-in-the-Haystack Metaphor

code paths

programming error

Courtesy of Thomas G. Moertel.
The Needle-in-the-Haystack Metaphor

GNU C Library, string/tst-strtok.c

```c
#include <stdio.h>
#include <string.h>

int main (void) {
    char buf[1] = { 0 };
    int failed = 0;

    if (strtok (buf, " ") != NULL)
        failed = 1;
    else if (strtok (NULL, " ") != NULL)
        failed = 1;

    return failed;
}
```

look under this piece of hay
ScalaCheck

http://scalacheck.googlecode.com/

import org.scalacheck._
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {
  property("startsWith") =
    forAll(( a: String, b: String ) => (a+b).startsWith(a))

  property("concat") =
    forAll(( a: String, b: String ) =>
      (a+b).length > a.length &&
      (a+b).length > b.length)

  property("substring") =
    forAll(( a: String, b: String ) =>
      (a+b).substring(a.length) == b)
}

$ scalac -cp scalacheck-1.8.jar 
  StringSpecification.scala

$ scala -cp .:scalacheck-1.8.jar 
  StringSpecification

+ String.startsWith: OK, passed 100 tests.
! String.concat: Falsified after 0 passed tests.

> ARG 0: ""
> ARG 1: ""

+ String.substring: OK, passed 100 tests.
ScalaCheck
http://scalacheck.googlecode.com/

import org.scalacheck._
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {
  property("startsWith") =
    forAll((a: String, b: String) => (a+b).startsWith(a))

  property("concat") =
    forAll((a: String, b: String) =>
      (a+b).length > a.length && (a+b).length > b.length)

  property("substring") =
    forAll((a: String, b: String) =>
      (a+b).substring(a.length) == b)
}

generate 2 random strings
$ scalac -cp scalacheck-1.8.jar StringSpecification.scala
$ scala -cp .:scalacheck-1.8.jar StringSpecification
+ String.startsWith: OK, passed 100 tests.
! String.concat: Falsified after 0 passed tests.

> ARG
ARG
ARG
+ String.substring: OK, passed 100 tests.
ScalaCheck

http://scalacheck.googlecode.com/

```scala
import org.scalacheck._
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {
    property("startsWith") =
        forAll((a: String, b: String) => (a+b).startsWith(a))

    // Is this really always true?
    property("concat") =
        forAll((a: String, b: String) =>
            (a+b).length > a.length && (a+b).length > b.length)

    property("substring") =
        forAll((a: String, b: String) =>
            (a+b).substring(a.length) == b)
}
```

ScalaCheck
http://scalacheck.googlecode.com/

import org.scalacheck._
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {
  property("startsWith") = forAll((a: String, b: String) => (a+b).startsWith(a))
  // Is this really always true?
  property("concat") = forAll((a: String, b: String) => (a+b).length > a.length && (a+b).length > b.length)
  property("substring") = forAll((a: String, b: String) => (a+b).substring(a.length) == b)
}

Compile...
$ scalac -cp scalacheck-1.8.jar \
  StringSpecification.scala
$ scala -cp .:scalacheck-1.8.jar \
  StringSpecification
+ String.startsWith: OK, passed 100 tests.
! String.concat: Falsified after 0 passed tests.
> ARG
0: ""
1: ""
+ String.substring: OK, passed 100 tests.
ScalaCheck

http://scalacheck.googlecode.com/

import org.scalacheck._
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {

  property("startsWith") =
    forAll((a: String, b: String) => (a+b).startsWith(a))

  // Is this really always true?

  property("concat") =
    forAll((a: String, b: String) =>
      (a+b).length > a.length && (a+b).length > b.length)

  property("substring") =
    forAll((a: String, b: String) =>
      (a+b).substring(a.length) == b)

}

Run!

$ scalac -cp scalacheck-1.8.jar \\n  StringSpecification.scala

$ scala -cp .:scalacheck-1.8.jar \\n  StringSpecification

+ String.startsWith: OK, passed 100 tests.

! String.concat: Falsified after 0 passed tests.
  > ARG_0: ""
  > ARG_1: ""

+ String.substring: OK, passed 100 tests.
Automated Specification-Based Testing Tools

- Haskell’s QuickCheck
- JVM (Scala, Java, etc.)
  - ScalaCheck → http://scalacheck.googlecode.com/
  - QuickCheck → http://quickcheck.dev.java.net/
  - Functional Java’s fj.test → http://functionaljava.org/
  - Randoop → https://code.google.com/p/randoop/
- Racket’s FastTest (Scheme)
- QuickCheck++ (C++) → http://software.legiasoft.com/quickcheck/
- qc (Python) → https://github.com/dbravender/qc
- LectroTest (Perl) → http://community.moertel.com/ss/space/LectroTest
- ...
However...

“Program testing can be used to show the presence of bugs, but never to show their absence.”

— E. Dijkstra
Summary

- testing allows **bugs to be uncovered**
- ... helps **keep them away**
Summary

- testing allows **bugs to be uncovered**
- ... helps **keep them away**
- ... makes it easier to **refactor code**
Summary

- testing allows bugs to be uncovered
- ... helps keep them away
- ... makes it easier to refactor code
- (almost) anything can be tested, with any granularity
Summary

- testing allows **bugs** to be uncovered
- ... helps keep them away
- ... makes it easier to **refactor** code
- (almost) anything can be tested, with any granularity
- randomly generated tests are good at finding bugs
References

- *QuickCheck Testing for Fun and Profit*, J. Hughes, 2007