CSE306 Software Quality in Practice

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Tests are written without regard to **HOW** code is written

Diagram showing flow from input to output.
blackbox testing

Tests are meant to capture the intended behavior of the system (the requirements/specifications): **WHAT** the code should do.

input → [black box] → output
whitebox testing

Tests are written taking into consideration **HOW** the code is written.

```java
if (x < y) {
    z = f(x, y);
} else {
    z = g(x, y, z);
}
```
Use a code coverage tool to ensure that tests exercise *ALL* possible computation paths.

```java
if (x < y) {
    z = f(x, y);
} else {
    z = g(x, y, z);
}
```
whitebox testing

Use a code coverage tool to ensure that tests exercise **ALL** possible computation paths.

```java
if (x < y) {
    z = f(x, y);
} else {
    z = g(x, y, z);
}
```
Code coverage

We will use gcov as our coverage tool.

Compile with,

```
-fprofile-arcs
-ftest-coverage
-lgcov
```

as in:

```
gcc $(CFLAGS) -fprofile-arcs -ftest-coverage -L /util/CUnit/lib -I /util/CUnit/include/CUnit/ $(OBJECTS) tests.c -o tests -lcunit -lgcov
```
using gcov to verify test coverage

- compile test code with extra flags
  - this instruments code to gather coverage information
- run tests
  - this runs your tests and allows the instrumentation to collect coverage data that shows what parts of the implementation were exercised by the tests
- run gcov on the source file (e.g. source.c) whose coverage you’re interested in exploring
- use ‘man gcov’ to see gcov command line options. Try -b.
- Look at the file produced by gcov (e.g. source.c.gcov)
Exercise:
https://tools.ietf.org/html/rfc3986#section-3.1

Clone:
https://classroom.github.com/a/Iw9unBmL

Docs:
https://gcc.gnu.org/onlinedocs/gcc/Invoking-Gcov.html#Invoking-Gcov