Intro to Testing
Quiz

- What are three perspectives/roles from which you may consider software quality?
- What is one concern for each of these perspectives?
Why Do We Test?

- Recall: What role did testing play in the process we saw last time?
Why Do We Test?

- Recall: What role did testing play in the process we saw last time?
  - Measurement – Testing provides a metric of software quality
Why Do We Test?

• Recall: What role did testing play in the process we saw last time?
  – Measurement – Testing provides a metric of software quality

  e.g. for requirements / criteria $R_1$, $R_2$, $R_3$, $R_4$

Each test $T$ can check a requirement
Why Do We Test?

• Recall: What role did testing play in the process we saw last time?
  – Measurement – Testing provides a metric of software quality

  e.g. for requirements / criteria $R_1, R_2, R_3, R_4$

  Each test $T$ can check a requirement

  $$T_1 \rightarrow R_1, R_2$$
Why Do We Test?

• Recall: What role did testing play in the process we saw last time?
  – Measurement – Testing provides a metric of software quality

e.g. for requirements / criteria $R_1$, $R_2$, $R_3$, $R_4$
Each test $T$ can check a requirement

$$T_1 \rightarrow R_1, R_2$$
$$T_2 \rightarrow R_3$$
Why Do We Test?

- Recall: What role did testing play in the process we saw last time?
  - Measurement – Testing provides a metric of software quality

E.g. for requirements / criteria $R_1, R_2, R_3, R_4$

Each test $T$ can check a requirement

- $T_1 \rightarrow R_1, R_2$ ✓
- $T_2 \rightarrow R_3$ ✓
- $T_3 \rightarrow R_4$ ×
But What is Testing?

*Reasoning* about behavior is hard/subtle.
But What is Testing?

*Reasoning* about behavior is hard/subtle.

*Running* a program is easy (easier)....
But What is Testing?

*Reasoning* about behavior is hard/subtle.

*Running* a program is easy (easier)....

*Testing* (informally):

Running the program to see if it behaves as expected
But What is Testing?

Reasoning about behavior is hard/subtle.
Running a program is easy (easier)....

**Testing** (informally):
Running the program to see if it behaves as expected

Simple idea, but...

- More than half of development cost
- Still cheaper than not testing
- Testing well is hard
Idea?

Run a program on all inputs:

```python
for test in allPossibleInputs:
    run_program(test)
```
Ideas?

Run a program on all inputs:

```python
for test in allPossibleInputs:
    run_program(test)
```

Why not?
Ideas?

Run a program on all inputs:

```python
for test in allPossibleInputs:
    run_program(test)
```

Why not?

Maybe select a few tests:

```python
import random.sample
for test in sample(allPossibleInputs, 100):
    run_program(test)
```
Ideas?

Run a program on all inputs:

```python
for test in allPossibleInputs:
    run_program(test)
```

Why not?

Maybe select a few tests:

```python
import random.sample
for test in sample(allPossibleInputs, 100):
    run_program(test)
```

Why not?
Ideas?

Run a program on all inputs:

```python
for test in allPossibleInputs:
    run_program(test)
```

Why not?

Maybe select a few tests:

```python
import random.sample
for test in sample(allPossibleInputs, 100):
    run_program(test)
```

Why not?

*A primitive example of fuzz testing.*
Need A Bit More Care

Testing:

• *Dynamically* examines (runs) a program
Need A Bit More Care

Testing:

- *Dynamically* examines (runs) a program
- Considers specific *software under test*
Testing:

- *Dynamically* examines (runs) a program
- Considers specific *software under test*
- Run *test cases* from a *test suite* that targets specific quality goals
Testing:

- *Dynamically* examines (runs) a program
- Considers specific *software under test*
- Run *test cases* from a *test suite* that targets specific quality goals
- Identifies differences between *observed behavior* and *expected behavior*
Need A Bit More Care

Testing:

• *Dynamically* examines (runs) a program
• Considers specific *software under test*
• Run *test cases* from a *test suite* that targets specific quality goals
• Identifies differences between *observed behavior* and *expected behavior*

We can use this framework to refine how we test
Targeting Quality Objectives

- **Functional**
  - Does the program provide expected output for a given input?
    e.g. ...
Targeting Quality Objectives

- **Functional**
  - Does the program provide expected output for a given input?
  - e.g. Correct Output. All features present. Interface design.
Targeting Quality Objectives

- **Functional**
  - Does the program provide expected output for a given input?
    e.g. Correct Output. All features present. Interface design.

- **Nonfunctional**
  - Are output independent goals met?
    e.g. ...
Targeting Quality Objectives

- **Functional**
  - Does the program provide expected output for a given input?
    e.g. Correct Output. All features present. Interface design.

- **Nonfunctional**
  - Are output independent goals met?
    e.g. Performance, Scalability, Security, *Documentation*
Targeting Quality Objectives

• **Functional**
  – Does the program provide expected output for a given input?
    e.g. Correct Output. All features present. Interface design.

• **Nonfunctional**
  – Are output independent goals met?
    e.g. Performance, Scalability, Security, *Documentation*

We'll start this semester by looking at functional goals.
Subtle Terminology

- **Fault / Defect**
  - Flaws in static software (e.g. incorrect code)
Subtle Terminology

- **Fault / Defect**
  - Flaws in static software (e.g. incorrect code)

- **Failure**
  - An observable, incorrect behavior as compared to expected results
Subtle Terminology

- **Fault / Defect**
  - Flaws in static software (e.g. incorrect code)
- **Failure**
  - An observable, incorrect behavior as compared to expected results
- **Error / Infection**
  - Incorrect internal state (not yet observed)
Subtle Terminology

- **Fault / Defect**
  - Flaws in static software (e.g. incorrect code)

- **Failure**
  - An observable, incorrect behavior as compared to expected results

- **Error / Infection**
  - Incorrect internal state (not yet observed)

- **Latent Defect**
  - Unobserved defects in delivered software that testing did not expose
Subtle Terminology

- **Fault / Defect**
  - Flaws in static software (e.g. incorrect code)

- **Failure**
  - An observable, incorrect behavior as compared to expected results

- **Error / Infection**
  - Incorrect internal state (not yet observed)

- **Latent Defect**
  - Unobserved defects in delivered software that testing did not expose

The later a defect is found, the more it costs to fix. *Why?*
void toUppercase(char *str) {
    for (int i = 0, e = strlen(str) - 1; i < e; ++i) {
        if (isletter(str[i]) && islower(str[i])) {
            str[i] = str[i] - 32;
        }
    }
    printf("\%s\n", str);
}
A Simple Example

```c
void toUppercase(char *str) {
    for (int i = 0, e = strlen(str) - 1; i < e; ++i) {
        if (isletter(str[i]) && islower(str[i])) {
            str[i] = str[i] - 32;
        }
    }
    printf("\%s\n", str);
}
```

- What is a fault in this program?
A Simple Example

What is a fault in this program?

What is a test case that has a failure?

```c
void toUppercase(char *str) {
    for (int i = 0, e = strlen(str) - 1; i < e; ++i) {
        if (isletter(str[i]) && islower(str[i])) {
            str[i] = str[i] - 32;
        }
    }
    printf(“%s\n”, str);
}
```
A Simple Example

void toUppercase(char *str) {
    for (int i = 0, e = strlen(str) - 1; i < e; ++i) {
        if (isletter(str[i]) && islower(str[i])) {
            str[i] = str[i] - 32;
        }
    }
    printf(“%s
”, str);
}

• What is a fault in this program?
• What is a test case that has a failure?
• What is a test case that does not have a failure?
A Simple Example

void toUppercase(char *str) {
    for (int i = 0, e = strlen(str) - 1; i < e; ++i) {
        if (isletter(str[i]) && islower(str[i])) {
            str[i] = str[i] - 32;
        }
    }
    printf(“%s\n”, str);
}

- What is a fault in this program?
- What is a test case that has a failure?
- What is a test case that does not have a failure?

What exactly do we mean by test case?
Test Cases

Test cases need

- Input to provide the program
- Expected output or behavior to check for correctness
Test Cases

Test cases need

- Input to provide the program
- Expected output or behavior to check for correctness

But where does the expected behavior come from?

- An oracle
Test Oracles

- In general, a means of deciding whether a test *passes* or *fails* (was the behavior expected or not)
Test Oracles

- In general, a means of deciding whether a test *passes* or *fails* (was the behavior expected or not)
- Sometimes very simple
  - How are unit tests evaluated?
Test Oracles

• In general, a means of deciding whether a test passes or fails (was the behavior expected or not)

• Sometimes very simple
  – How are unit tests evaluated? (Test Drivers!)
Test Oracles

- In general, a means of deciding whether a test passes or fails (was the behavior expected or not)
- Sometimes very simple
  - How are unit tests evaluated? (Test Drivers!)
- Sometimes tricky
  - Is result strictly specified? (content, order, timing, ...)
  - Is the program deterministic?
Test Oracles

- In general, a means of deciding whether a test passes or fails (was the behavior expected or not)

- Sometimes very simple
  - How are unit tests evaluated? (Test Drivers!)

- Sometimes tricky
  - Is result strictly specified? (content, order, timing, ...)
  - Is the program deterministic?

- Sometimes requires a person
  - Expensive and undesirable
  - "Does this software meet my needs?"
Coverage / Adequacy

Recall: can't look at all possible inputs.
Coverage / Adequacy

Recall: can't look at all possible inputs.

Need to determine if a test suite *covers / is adequate* for our quality objectives.
Coverage / Adequacy

Recall: can't look at all possible inputs.
Need to determine if a test suite *covers / is adequate* for our quality objectives.

- Sufficiently addresses criteria
- Lack of failures provides enough confidence that the software is acceptable
Coverage / Adequacy

Recall: can't look at all possible inputs.
Need to determine if a test suite covers / is adequate for our quality objectives.

- Sufficiently addresses criteria
- Lack of failures provides enough confidence that the software is acceptable

Key Idea:
- Find a smaller test suite that is representative of our goals
Approaches

- Test until you run out of time
- Test until you run out of money
Approaches

- Test until you run out of time
- Test until you run out of money
- Identify redundant inputs based on *the specification*
Approaches

- Test until you run out of time
- Test until you run out of money
- Identify redundant inputs based on the specification
- Identify redundant inputs based on program structure
Approaches

- Test until you run out of time
- Test until you run out of money
- Identify redundant inputs based on the specification
- Identify redundant inputs based on program structure
- Identify poorly tested areas by measuring how well your tests identify potential bugs
Approaches

- Test until you run out of time
- Test until you run out of money
- Identify redundant inputs based on the specification
- Identify redundant inputs based on program structure
- Identify poorly tested areas by measuring how well your tests identify potential bugs

No approach covers everything you want! Need to combine them for a balanced approach toward the desired goals.
Next Up...

Revisit the basics of unit testing.