Testing and Debugging

Tips & Tricks

Ibrahim Albluwi

COS 126 Unofficial Coding "Strategy"

Repeat Until Deadline :

Hack!

Click Check All Submitted Files

If all correctness tests pass :

Celebrate

Break

Fake some test cases

If in the mood :

Choke off CheckStyle

COS 126 Unofficial Coding "Strategy"

Repeat 10 times :

~~Repeat Until Deadline~~ :

Hack!

Click Check All Submitted Files

If all correctness tests pass :

Celebrate

Break

Fake some test cases

If in the mood :

Choke off CheckStyle

Not Realistic!

Doesn't work in 226!

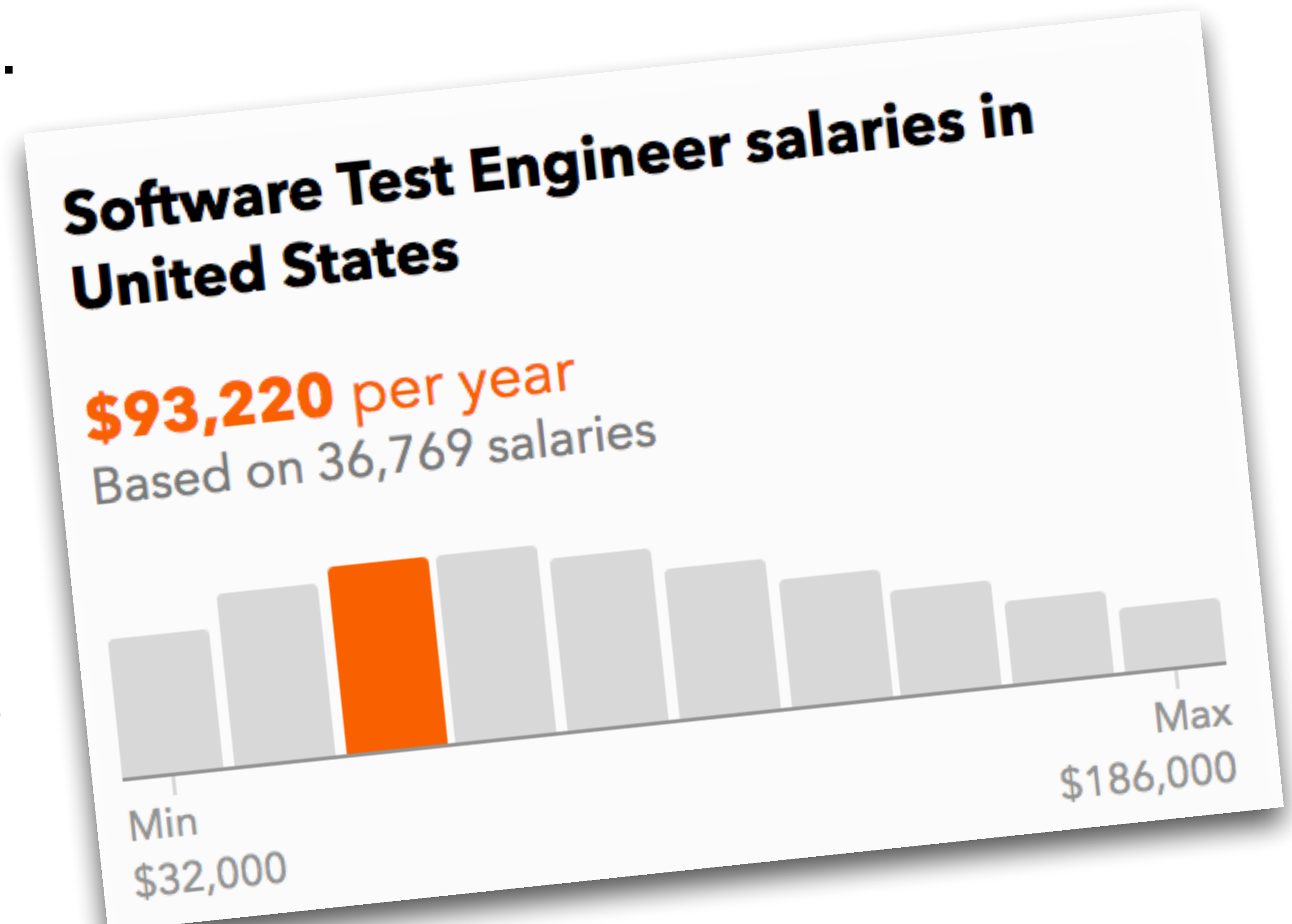
Intended Coding Strategy

```
Repeat :
    Repeat :
        Hack thoughtfully and with style!
        Test
        If all tests pass :
            Break
    Click Check All Submitted Files
    If all tests pass :
        Break
Celebrate!
```

Today's Class Meeting

Goals:

- Testing and debugging tips and tricks.
- Help you succeed in 226.
- Develop healthy programming habits.



Not Goals:

- Rigorous introduction to testing.
- Prepare you for a *SW Testing Engineer* job.

Warmup Quiz http://etc.ch/i7VR

Which of the following best describes you as you work on programming assignments?

(A) **Idealist**: Codes very carefully. Usually gets it right from the first shot. Doesn't need to test much.

(B) **Pragmatist**: Let's get something up and running quickly. Careful testing will let us know if there is an issue.

(C) **Submissionist**: Why code too carefully? Why test carefully? **KEEP CALM AND CHECK ALL SUBMITTED FILES**.

Quiz # 1 `http://etc.ch/i7VR`

Which of the following tests are ***necessary*** and ***sufficient*** for testing a method that ***returns the maximum of three integers***.

A. (1, 2, 3)

B. (1, 2, 3) (3, 2, 1) (1, 3, 2)

C. (1, 2, 3) (1, 3, 2) (2, 1, 3) (2, 3, 1) (3, 1, 2) (3, 2, 1)

D. None of the above.

Quiz # 1

http://etc.ch/i7VR

Which of the following tests are ***necessary*** and ***sufficient*** for testing a method that ***returns the maximum of three integers***.

integers can be negative!

A. (1, 2, 3)

B. (1, 2, 3) (3, 2, 1) (1, 3, 2)

C. (1, 2, 3) (1, 3, 2) (2, 1, 3) (2, 3, 1) (3, 1, 2) (3, 2, 1)

D. None of the above.

Tip # 1 Tests can be written ***before*** the program is implemented.
Blackbox Testing: Test based on problem description.

Tip # 2 Think carefully about the ***domain*** of the inputs.

Example

The following code passes all test cases with positive integers but fails all test cases with ***negative*** integers!

```
int max(int a, int b, int c) {
    int max = 0;

    if (a > max)
        max = a;
    if (b > max)
        max = b;
    if (c > max)
        max = c;

    return max;
}
```

Quiz # 2 http://etc.ch/i7VR

Which of the following tests are ***necessary*** and ***sufficient*** for testing a method that ***returns the maximum of three integers***.

A. (1, 2, 3) (3, 2, 1) (1, 3, 2) (-1, -2, -3) (-3, -2, -1) (-1, -3, -2)

B. All 3-permutations of -3, -2, -1, 1, 2, 3.

C. Thousands of randomly generated positive and negative integers.

D. None of the above.

Quiz # 2

http://etc.ch/i7VR

Which of the following tests are ***necessary*** and ***sufficient*** for testing a method that ***returns the maximum of three integers***.

- **A.** (1, 2, 3) (3, 2, 1) (1, 3, 2) (-1, -2, -3) (-3, -2, -1) (-1, -3, -2)
- **B.** All 3-permutations of -3, -2, -1, 1, 2, 3.
- **C.** Thousands of randomly generated positive and negative integers.
- **D.** None of the above.

Tip # 3 Think of input **equivalence classes.**

(1, 2, 3) is equivalent to (2, 3, 4) = (min, mid, max)
(3, 2, 1) is equivalent to (30, 5, 4) = (max, mid, min)

Example

The following code passes all test cases with 3-permutations of -3, -2. -1, 1, 2, 3. However, it could fail if the input has ***duplicates***!

```
int max(int a, int b, int c) {
    int max = 0;

    if (a > b && a > c)
        max = a;
    if (b > a && b > c)
        max = b;
    if (c > a && c > b)
        max = c;

    return max;
}
```

Now What?

Using **{-3, -2. -1, 1, 2, 3}**, is it enough to test
all possible 3-tuples (permutations with repetition)?

Not necessarily!

Example

The following code passes all test cases with 3-tuples from {-3, -2. -1, 1, 2, 3}. However, it could fail when the used numbers are too small or too large, like:

a = 2147483647   b = 2147483647   c = -2147483647

```
int max(int a, int b, int c) {
    int max = 0;

    if (a - b >= 0 && a - c >= 0)
        max = a;
    if (b - a >= 0 && b - c >= 0)
        max = b;
    if (c - a >= 0 && c - b >= 0)
        max = c;

    return max;
}
```

Overflow!

```
   2147483647
+  2147483647
  ------------
=  -2
```

Tip # 4 Always test boundary inputs and corner cases.

Tip # 5

Blackbox testing may not be enough.
Whitebox Testing: Generate tests based on code.

Examine code and make sure there are test cases that cover all possible program flow paths.

Example

```
if (a == true)
    doSomething();
else
    doSomethingElse();

for (int i = n; i > 0; i--)
    doAnotherThing();

doAFinalThing();
```

Test both branches

Test entering the loop and not entering the loop

Quiz # 3

http://etc.ch/i7VR

```
push(x):
    if (size == cap)
        Error
    last++
    data[last] = x
    size++
```

```
pop():
    if (size == 0)
        Error
    x = data[last]
    last--
    return x
```

```
toString():
    s = "";
    for (i=0 --> size-1)
        s += data[i]
    return s
```

Which of the following tests could reveal the bug in the following ***stack*** code?

Hint: Is size always correctly updated?

- **A.** Calling **`push`** then **`toString`** then **`pop`**.
- **B.** Calling **`push`** then **`pop`** then **`toString`**.
- **C.** Calling **`push`** (many times) then **`pop`** (many times) then **`push`**.
- **D.** All of the above.

Quiz # 3

http://etc.ch/i7VR

```
push(x):
    if (size == cap)
        Error
    last++
    data[last] = x
    size++
```

```
pop():
    if (size == 0)
        Error
    x = data[last]
    last--
    return x
```

```
toString():
    s = "";
    for (i=0 --> size-1)
        s += data[i]
    return s
```

Which of the following tests *could* reveal the bug in the following ***stack*** code?

Hint: Is size always correctly updated?

- **A.** Calling **`push`** then **`toString`** then **`pop`**.
- **B.** Calling **`push`** then **`pop`** then **`toString`**.
- **C.** Calling **`push`** (many times) then **`pop`** (many times) then **`push`**.
- **D.** All of the above.

Tip # 6 Test different ***orderings*** of method calls.

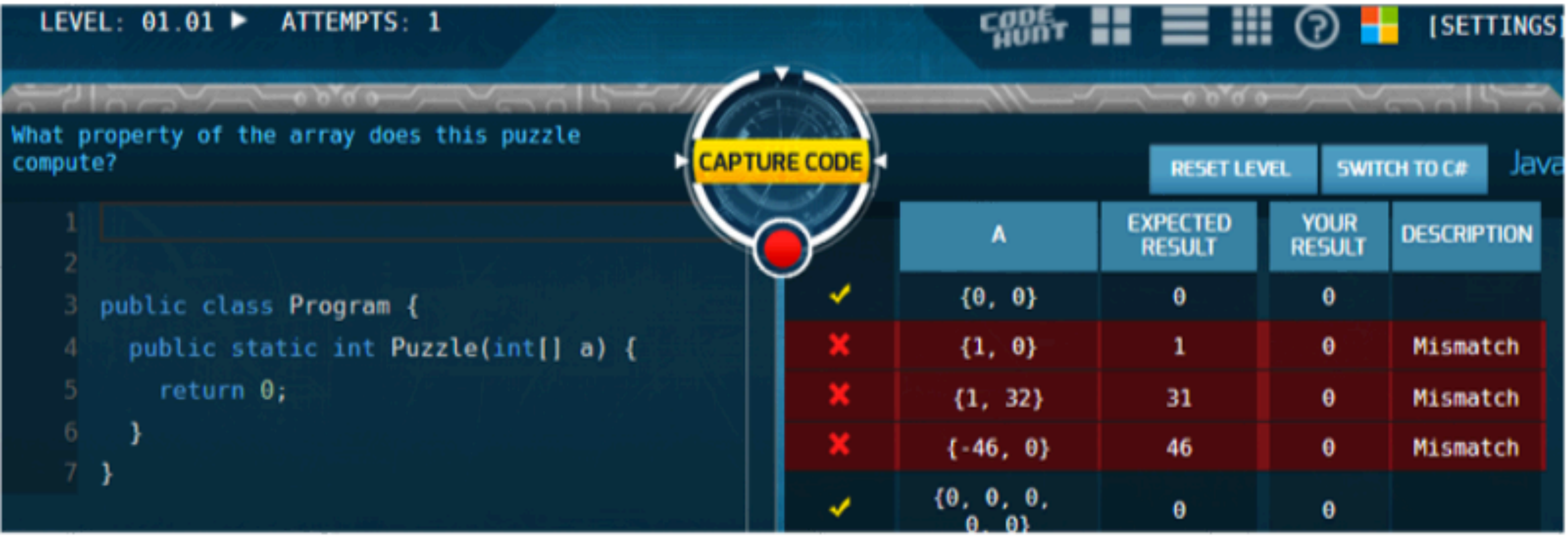
Tip # 7 Test methods on different ***states*** of the data structure

Summary

- Implement a simple test client ***before*** starting to code the ADT.
- Write tests that cover all ***input equivalence classes***.
- Always test ***boundary inputs*** and corner cases.
- Write tests that ***cover all possible flow paths*** in the code.
- ***Intermix method calls*** to see if one breaks another.
- Test method calls with all possible ***states*** of the object.

Lesson Testing can show the presence of errors but not their absence!

This slide is brought to you by Microsoft



Failed test cases?

Time to Debug!

Easiest Bugs: Compile Time Errors

Tip # 1 Understand what the error means.

Confused?

- Copy and paste error to Google.

```
Students.java:90: error: illegal start of expression
}
^
```

- Use Java compiler messages cheatsheets.

Examples:

https://introcs.cs.princeton.edu/java/11cheatsheet/errors.pdf
https://dzone.com/articles/50-common-java-errors-and-how-to-avoid-them-part-1
http://mindprod.com/jgloss/compileerrormessages.html#TYPESAFETYERASED

Easiest Bugs: Compile Time Errors

Tip # 2 Focus on the first error first.

- An error can produce a cascade of other errors. Fixing the first error, automatically fixes all subsequent errors caused by it.

```
ErdosRenyi.java:29: error: '.class' expected
        For (int i = 0; i < n; i++)
                 ^
ErdosRenyi.java:29: error: > expected
        For (int i = 0; i < n; i++)
                             ^
ErdosRenyi.java:29: error: not a statement
        For (int i = 0; i < n; i++)
                          ^
ErdosRenyi.java:29: error: ';' expected
        For (int i = 0; i < n; i++)
                                  ^
4 errors
```

Runtime Exceptions

Know the anatomy of a runtime exception.

Exception name **Message**

```
Exception in thread "main"
java.lang.IndexOutOfBoundsException: -1
  at test.convertIndex(test.java:6)
  at test.findMax(test.java:15)
  at test.max(test.java:31)
  at test.main(test.java:41)
```

Stack Trace

Confused? Copy and paste exception to Google
(without message and trace).

Debugging Non-trivial Errors

Tip # 3 Know exactly ***when*** the error happens.

Can you **reproduce** the error?

Example 1: Assume max(-1, 1, 1) fails:

- Does max fail for all inputs?
- Does it fail only with negative numbers?
- Does it fail only when there are duplicates?
- Does it fail only with (min, max, max)?

Example 2: Autograder says: Intermixing calls to **`push`**, **`pop`** and **`top`** throws an exception.

- Can you come up a sequence of **`push`**, **`pop`** and **`top`** calls that would produce the same exception?

Debugging Non-trivial Errors

Tip # 4 Use ***print*** statements to know ***where*** and ***why*** the error happens.

Example:

```
int myFunction(int x) {
    StdOut.println(x);

    if (isInState1())
        x = doSomething();
    StdOut.println(x);

    if (isInState2())
        x = doSomethingElse();
    StdOut.println(x);

    return x;
}
```

Helps understand how the value of x changes.

Debugging Non-trivial Errors

Tip # 4 Use ***print*** statements to know ***where*** and ***why*** the error happens.

Example:

```
int myFunction(int x) {

    if (isInState1()) {
        x = doSomething();
        StdOut.println("State1");
    }

    if (isInState2()) {
        x = doSomethingElse();
        StdOut.println("State2");
    }

    return x;
}
```

Helps understand program flow.

Using Print Statements

Trick # 1 Use ***java.util.Arrays.***

Print 1D array: `StdOut.print(Arrays.toString(a))`

Print 2D array: `StdOut.print(Arrays.deepToString(a))`

Fill array: `StdOut.print(Arrays.fill(a, value))`

Compare arrays: `Arrays.equals(a1, a2)`

`Arrays.deepEquals(a1, a2)`

Warning # 1 Do not forget to remove debugging print statements before submitting!

Debugging Non-trivial Errors

Tip # 5 Check common sources of errors.

- **Copied-and-pasted code:** It is very common to forget to make the needed changes after copying code.
- **Variable scope:** Are there different variables with the same name?
- **Others?**

```
-if (var = true)
-if (str1 == str2)
-Boolean [] isTrue = new Boolean[10];
```

Debugging Non-trivial Errors

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

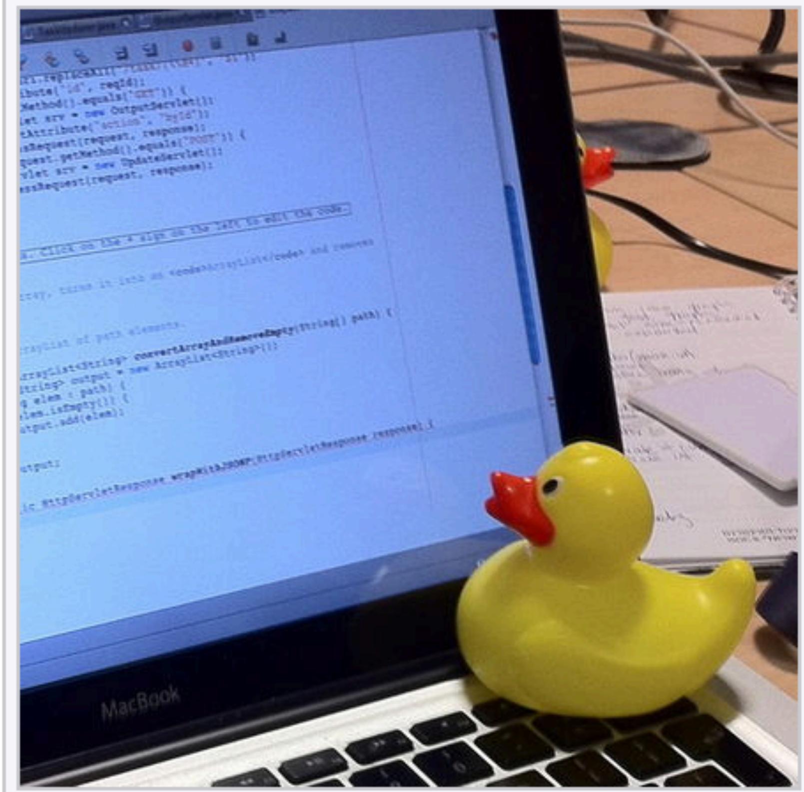
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

Rubber duck debugging

From Wikipedia, the free encyclopedia

In software engineering, **rubber duck debugging** or **rubber ducking** is a method of debugging code. The name is a reference to a story in the book *The Pragmatic Programmer* in which a programmer would carry around a rubber duck and debug their code by forcing themselves to explain it, line-by-line, to the duck.[1] Many other terms exist for this technique, often involving different inanimate objects.



A rubber duck in use by a developer to aid code review

Many programmers have had the experience of explaining a problem to someone else, possibly even to

Debugging Non-trivial Errors

Tip # 6 Document your assumptions

When you walk through the code (with the rubber duck):

- ***Insert comments*** explaining your logic.
- After certain blocks of code, insert comments explaining ***why you are sure the code must be correct*** at to that point.
- Use ***assertions***!

Your assumption.
An error is thrown if not true.

Message displayed
when error is thrown

```
assert booleanExpression : Value
```

Assertions Examples

```
void foo() {
    for (...) {
        if (...)
            return;
    }
    assert false;
}
```

Assumption: Flow should never reach here!

```
if (i % 3 == 0) {
    ...
} else if (i % 3 == 1) {
    ...
} else {
    assert i % 3 == 2 : i;
    ...
}
```

Assumption: i % 3 = 2. Fails if i is negative.

Precondition

Postconditions

```
void insert(int val) {
    assert isBST() : "BST properties violated"

    // tree insertion code

    assert isBST() : "BST properties violated"
    assert isBalanced() : "Insertion misbalances BST";
}
```

Debugging Non-trivial Errors

Trick # 2 Use the Debugger.

Demo!

Image on slide 1 retrieved on February 11 from:
http://weclipart.com/gimg/3386E8144A791493/bad-bug.png

Image on slide 4 retrieved on February 11 from:
https://www.indeed.com/salaries/Software-Test-Engineer-Salaries

Image on slide 21 retrieved on February 11 from:
http://www.skaip.org/bug-emoticon

Two assertion examples on slide 32 are from:
https://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html