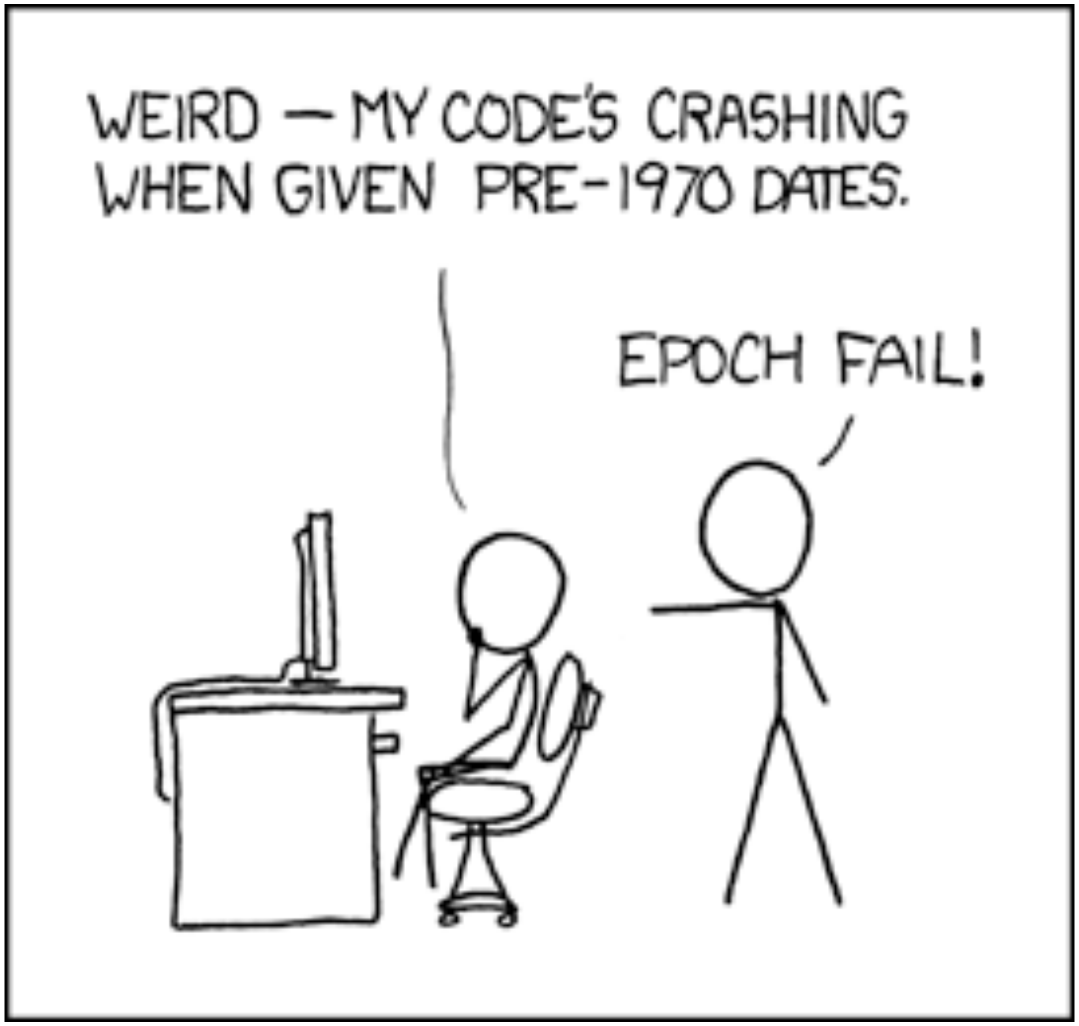
WIT COMP1000

Testing and Debugging

Testing Programs

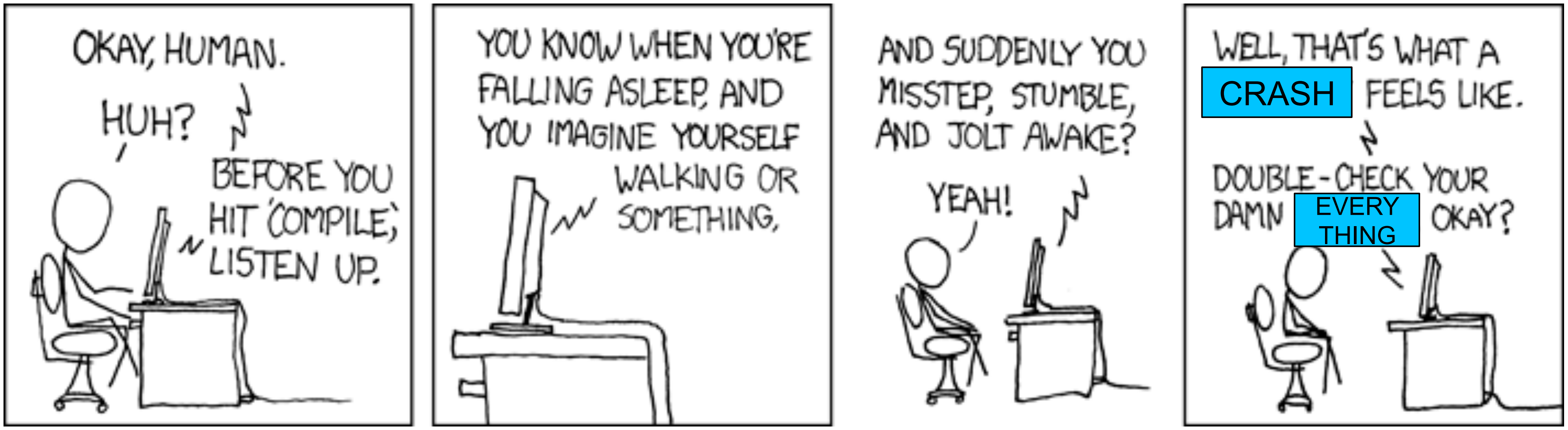
- When testing your code, always test a variety of input values
- Never test only one or two values because those samples may not catch some errors
- Always test "interesting" values
  - » Values that show up in the code
  - » For example, boundary values that change if/else behavior

"Interesting" Values



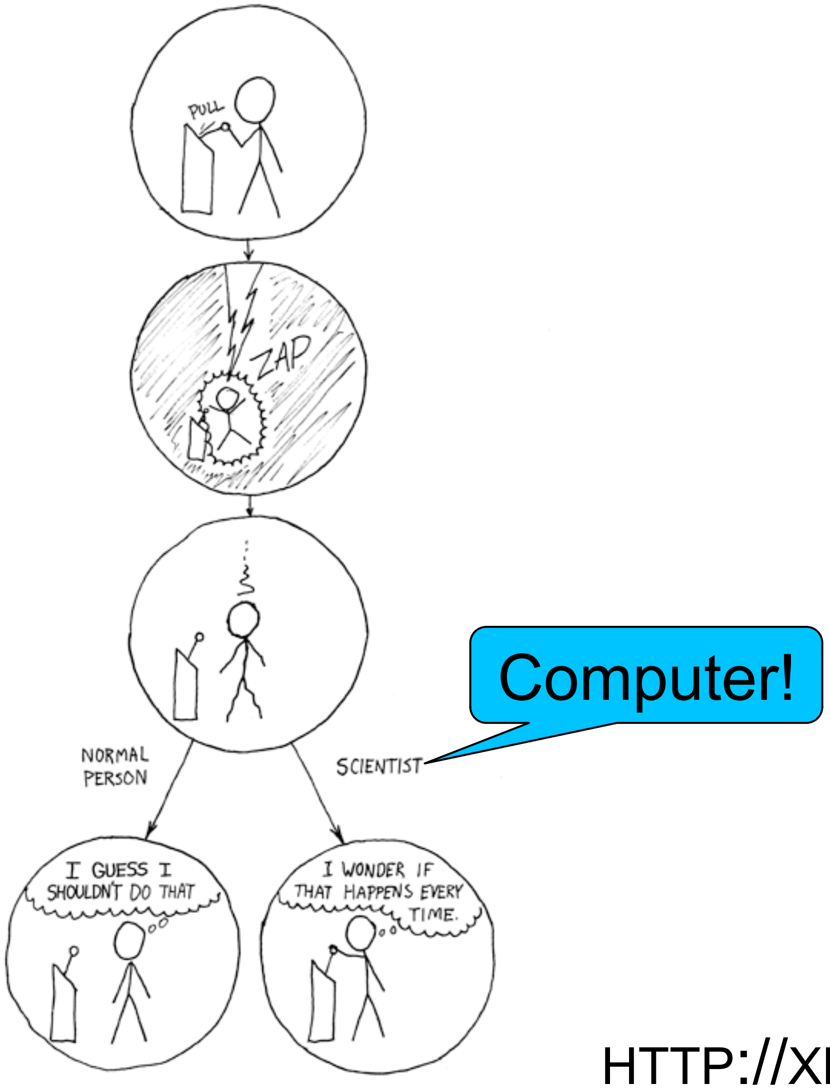
HTTP://XKCD.COM/376/

Also: Be Nice to Your Computer


OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE,' LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING,
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A CRASH FEELS LIKE.
DOUBLE-CHECK YOUR DAMN EVERY THING OKAY?

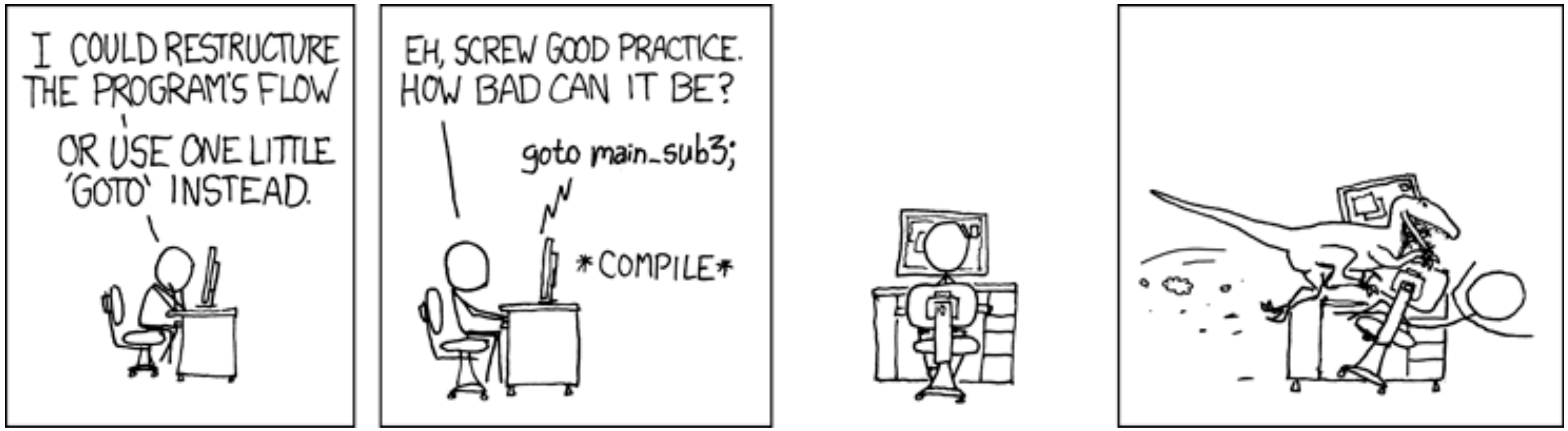
HTTP://XKCD.COM/371/

Also Also: Be Diligent!


PULL
ZAP
NORMAL PERSON
SCIENTIST
Computer!
I GUESS I SHOULDN'T DO THAT
I WONDER IF THAT HAPPENS EVERY TIME.

HTTP://XKCD.COM/242/

One Last Warning


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

HTTP://XKCD.COM/292/

Example Bug: Microsoft, 12/31/2008

*Early this morning we were alerted by our customers that there was a widespread issue affecting our 2006 model Zune 30GB devices (a large number of which are still actively being used). The technical team jumped on the problem immediately and isolated the issue: a bug in the internal clock driver related to the way the device handles a **leap year.** That being the case, the issue should be resolved over the next 24 hours as the time change moves to January 1, 2009. We expect the internal clock on the Zune 30GB devices will automatically reset tomorrow (noon, GMT). By tomorrow you should allow the battery to fully run out of power before the unit can restart successfully then simply ensure that your device is recharged, then turn it back on. If you're a Zune Pass subscriber, you may need to sync your device with your PC to refresh the rights to the subscription content you have downloaded to your device.*

Zune Bug Excerpt

```
// days is the number of days since 1980, e.g., 10000 or 365 or 10592
// year is the current year


    if (IsLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    }
```

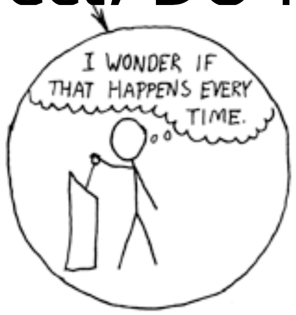
366 and 367 seem like good values to test

General Advice

- Always check for common errors
  - » Using = instead of ==
  - » Using > or < instead of <= or >=
  - » Other off-by-one errors
  - » Integer division
- Always test your code
  - » Use lots of values to test, including ones that change the behavior of the code

Localize Errors

- When you don't get the output you expect, DO NOT just change code randomly



- Narrow down where the problem is by checking values throughout the program
  - » Use output statements at key points to check current variable values
  - » Also use output statements to verify which branch of `if-else` statements are taken

Example (with Errors)

```java
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        double radius, height, volume;

        System.out.print("Enter the radius of the cone: ");
        radius = input.nextDouble();
        System.out.print("Enter the height of the cone: ");
        height = input.nextDouble();

        volume = 1 / 3 * Math.PI * radius * radius * height;

        System.out.println("The volume is: " + volume);
    }

}
```

Error Localized

```
Scanner input = new Scanner(System.in);

double radius, height, volume;

System.out.print("Enter the radius of the cone: ");
radius = input.nextDouble();
System.out.print("Enter the height of the cone: ");
height = input.nextDouble();

System.out.println("radius is " + radius);
System.out.println("height is " + height);

//volume = 1 / 3 * Math.PI * radius * radius * height;
//volume = 1 / 3 * Math.PI;
volume = 1 / 3;

System.out.println("The volume is: " + volume);
```

Integer division!

Simple Testing and Debugging

- When your code doesn't produce the correct output, don't just randomly change things
- Localize the error by using output statements to track variable values, branch conditions, and other interesting areas of your code
- Always check for common mistakes (look very hard, because they are easy to overlook)

Debugging Exercise

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    int answer = 3;
    int guess;

    System.out.print("Enter a guess between 1 and 10: ");
    guess = input.nextInt();

    if ((guess <= 1) || (guess >= 10)) {
        System.out.println("Invalid guess!");
        System.exit(0);
    }

    if (guess == answer) {
        System.out.println("Wrong! Try again.");
    }
    else {
        System.out.println("You got it!");
    }
}
```

Answer

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    int answer = 3;
    int guess;

    System.out.print("Enter a guess between 1 and 10: ");
    guess = input.nextInt();

    if ((guess < 1) || (guess > 10)) { // used <= and >= instead of < and >
        System.out.println("Invalid guess!");
        System.exit(0);
    }

    if (guess == answer) {
        System.out.println("You got it!"); // swapped two output messages
    }
    else {
        System.out.println("Wrong! Try again.");
    }
}
```

JUnit Testing

- Another widely used type of testing is known as *unit testing*
- The idea is to write additional code, called unit tests, that will automatically test your program with certain inputs to ensure that they produce the correct output
  - » Of course, you must verify that your unit tests are correct!
- JUnit is a very commonly used (and open source!) unit testing framework for Java
  - » The tests you've been in running in Eclipse all semester!
  - » Take some time to look at the test code to see that it's just more Java

Debugging Tools

- Debuggers help you to quickly find and identify errors in your code
- Allow you to:
  - » *Step* through your code one line at a time
  - » View current values of all variables as the program progresses
  - » Set *breakpoints* that will stop the code at certain places in your code
- Should NOT be used in place of proper testing and analysis, but rather as an assistive tool

Eclipse Debugger

- Eclipse has a built in debugger that has all the normal debugging functionality
- Set breakpoints by double clicking on the left of a line of code in the light blue vertical bar
- Start the program with debugging (**F11**, the bug button, or go to the **Run** menu and click **Debug**)
- The program will execute like normal until you come to a breakpoint
- When a breakpoint is encountered, the program will stop and Eclipse will ask you to switch to the Debug perspective (which you should do)

Eclipse Debug Perspective

- Once debugging, you can see the value of all variables in the upper right window (make sure you are looking at the **Variables** tab)
- Use the Play/Resume (**F8**) and Step Over (**F6**) buttons to navigate through the code
  - » Play/Resume starts the program running again, and it will continue until another breakpoint is encountered or the program ends
  - » Step Over executes *only* the next line of code and then stops
- There are two other options you'll use in the future
  - » Step Into (**F5**) executes the next line of code, and goes into a method if that line is a method call
  - » Step Return (**F7**) starts the program running again, and it continues until the current method call finishes, then stops again

Debugging Example

```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    int answer = 3;
    int guess;

    System.out.print("Enter a guess between 1 and 10: ");
    guess = input.nextInt();

    if ((guess <= 1) || (guess >= 10)) {
        System.out.println("Invalid guess!");
        System.exit(0);
    }

    if (guess == answer) {
        System.out.println("Wrong! Try again.");
    }
    else {
        System.out.println("You got it!");
    }
}
```

Steps

- Start by putting a breakpoint at the first `if` statement
  - Double click in the blue bar on the left of the line
  - It should add a blue dot to indicate that there is a breakpoint there now
  - You can double click on the dot to remove the breakpoint
- Hit **F11** to start debugging, and enter a value of 1 in the console
- Eclipse will then stop automatically at the `if` statement
  - Make sure to switch do the Debug perspective
- Examine the `guess` variable and see that it is 1
- Click the Step Over (**F6**) button to execute the `if` statement

Steps

- The program jumps into the `if` code block
  - This code should only be used when the guess was invalid, but 1 is a valid guess!
  - We now know to look very hard at the `if` condition because it must be wrong
  - In this case (as it often is), the problem is an off by one error cause by using <= instead of <
- Fix the `if` statement, then stop debugging and start the program again with debugging
- Next, set breakpoints at the two output statements that tell the user if their guess was correct or not
  - Examine the values of `guess` and `answer`
  - You'll see that the wrong output is printed when they are the same!

Debugging Exercise

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    double length, width, area;

    System.out.print("Enter positive rectangle length: ");
    length = input.nextDouble();
    System.out.print("Enter positive rectangle width: ");
    width = input.nextDouble();

    if ((length < 0) && (width < 0)) {
        System.out.println("Invalid measurements!");
        System.exit(0);
    }

    area = length * width;

    System.out.println("The area is: " + area);

}
```

Steps

- Run the program and find the following bugs:
  - » The program does not print an error message if either measurement is zero
  - » The program does not print an error message if only one measurement is negative
- There must be something wrong around the `if` statement
- Set a breakpoint on the `if` statement
- Use the Step Over button to execute one statement at a time

Steps

- You'll see that it skips the error message and `return` statement on several occasions when they should be executed
  - Verify this by looking at the values of `length` and `width` in the variables section of the debugger
- That means that the error must be in the `if` condition
- Try running with interesting boundary conditions such as -1, 0, and 1
  - As you test, pay attention to the conditions and change it when you note buggy behavior
- In particular, both conditions should be <= instead of < and it should be || instead of &&

Take Home Points

- Testing your code is vital, and it takes time to learn how to do it well

- Use simple testing and debugging techniques such as adding output statements throughout your code
  - Of course, be sure to remove them once you have fixed any problems!
- Use debuggers, like the Eclipse debugger, to trace through code in order to find errors
  - Set breakpoints near lines you want to check to skip ahead to those areas of the code