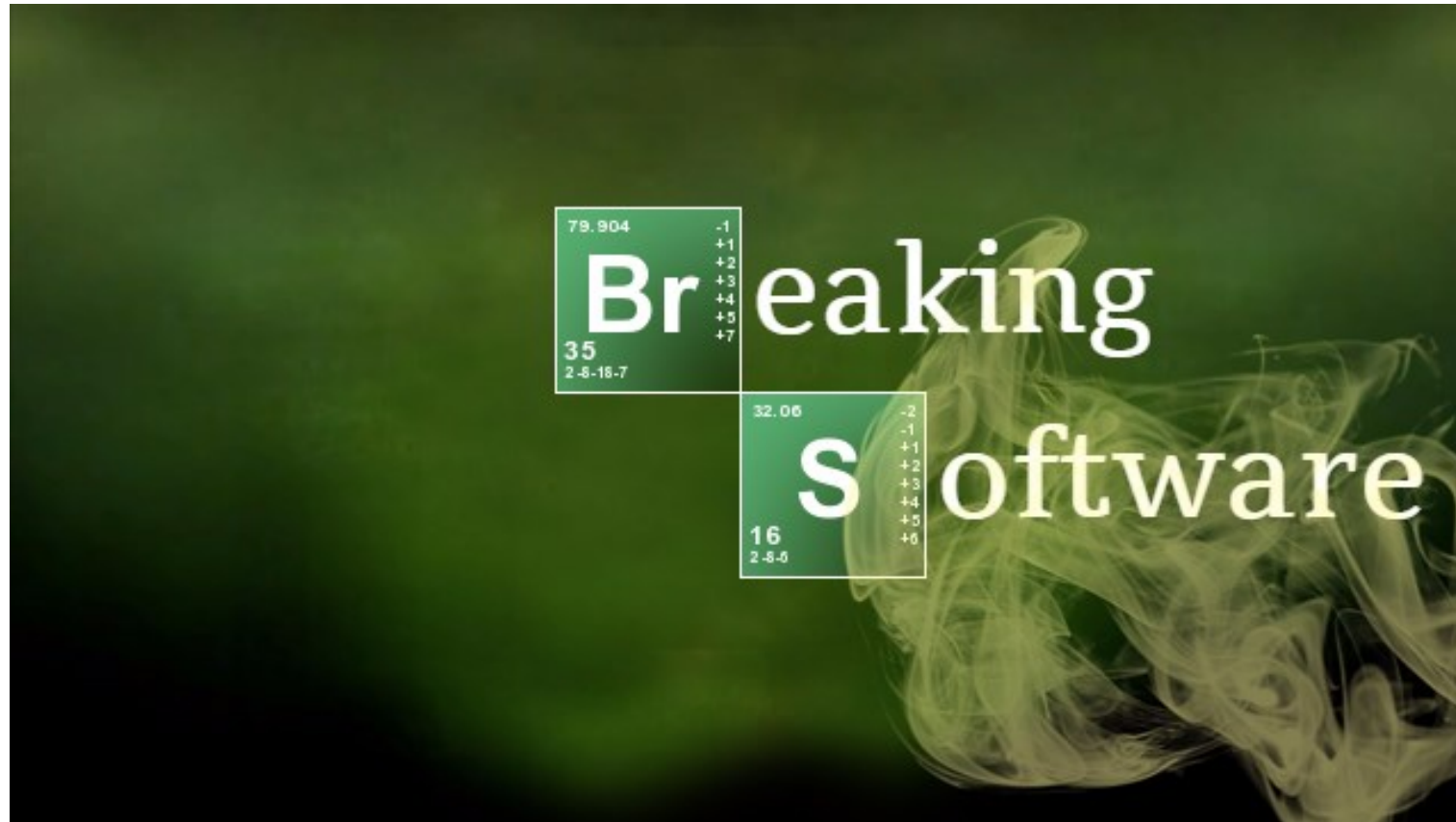# CS1632

Wonsun ahn

Software Usually Works Fine on the "Happy Path"

- *Happy Path*: A case where user inputs typical values; expected usage

- Software breaks on the unexpected cases
  - Corner cases or edge cases
  - When subsystems go down
  - When software is starved of resources (memory / bandwidth / CPU cycles)
  - When malicious users intentionally try to break your software
  - When you're off in the wilderness

Logic Errors:
The logic of the program is incorrect

- Requirement: Driving while drunk shall get a student a DUI.

- Erroneous Code:

```
if (student.isDrunk() || student.isDriving()) {
    student.setDUI(true);
} else {
    student.setDUI(false);
}
```

☛ Test all equivalence classes with interior and boundary values

OFF-by-one Errors:
Value is specified incorrectly by one unit

- A subset of logic errors, since code is logically incorrect
- Requirement: The minimum drinking age for student shall be 21.
- Erroneous Code:

```
if (student.getAge() > 21) {
    student.setCanDrink(true);
} else {
    student.setCanDrink(false);
}
```

☛ Pay special attention to boundary values

Null pointer error:
The program dereferences a null pointer.

- Erroneous Code:

```
String cs1632 = null;
cs1632 = cs1632.toUpperCase();

System.out.printf("Welcome to " + cs1632);
```

☛ When whitebox-testing, test cases where references may become null

☛ When blackbox-testing, test scenarios that may lead to null references
- Unsuccessful database or network connection
- Leaving a textbox empty in a fill-out form

Rounding / Floating Point Errors:
Errors due to Floating Point Imprecision

```
double oneVal = 1.0 / 857.0;
double total = oneVal * 857.0;

System.out.println("Total = " + total);
```

```
$ cd sample_code/breaking_software
$ javac FloatingPointError.java
$ java FloatingPointError
Total = 0.9999999999999999
```

• Is this a serious problem? After all, it's a miniscule difference …

Rounding / Floating Point Errors:
Errors due to Floating Point Imprecision

- Patriot Missile Defense failure of 1991
  - Failed to detect an incoming Scud missile during the Gulf War
  - Caused 28 deaths and around 100 injuries

- Why? Due to accumulated floating point precision error
  - Missile trajectory calculation involves millions of floating point operations
  - Precision errors tend to accumulate, significantly impacting final result

- ☛ Test long calculations where errors have a chance to accumulate

Integration Errors:
Errors at boundaries between systems/subsystems

Subsystem 1:

```
public class Spacecraft
    public void setDistance(int distanceInMiles) {
        ...
    }
}
```

Subsystem 2:

```
int startDistanceInKilometers = 14;
spacecraft.setDistance(startDistanceInKilometers);
```

Integration Errors:
Errors at boundaries between systems/subsystems

Subsystem 1:

```
OutputFile.write(TAB_DELIMITED);
```

Subsystem 2:
```
InputFile.read(COMMA_DELIMITED);
```

☛ Always test system after integration, even if subsystems are tested

Missing Data Errors:
Error in handling missing data correctly

```
public static void main(String[] args) {
    System.out.println(args[3]);
}
```

☛ If system expects some data, try testing without that data!
   (What happens when you pass no command line arguments?)

Bad Data Errors:
Error in handling invalid data correctly

```
Enter two numbers to divide: 7 0
Exception in thread "main"
    java.lang.ArithmeticException: / by zero
```

☛ If system expects certain values or certain formatting of values, try testing unexpected values or unexpected formatting!

☛ Later, we will learn *fuzz testing*: randomly generating bad data

Display Errors:
Data is correct but not displayed properly

- Requirement: System shall display value of PI with two decimals.
- Erroneous Code:

```
double pi = Math.PI;
// Prints 3.1 instead of 3.14
System.out.printf("Pi is equal to %.1f", pi);
```

☛ Need to test the display of data, as well as the data value
  - For text UI, need to parse output stream, extract the displayed data, and test
  - For graphical UI, need to extract data from displayed window or web page

I/O Errors:
Disk, network, I/O malfunction handled incorrectly

- **Erroneous Code**

```
try {
  // read in file
} catch (FileNotFoundException e) {
  // AAARGH WHAT DO I DO
  System.out.println("Panic! Emergency protocol!");
}
```

☛ If you see the above, try running without the file ☺

Configuration error:
Bad configuration handled incorrectly

- Requirement: *my.pitt.edu* shall warn user if cookies not enabled

- Expected behavior: Page displays "Website requires cookies."

- Observed behavior:

If system expects certain software,
or software with a specific version,
or software configured in a specific way,

☛ Test incorrect configurations,
  as well as correct configurations

The list goes on...

- Data type errors
  - Error arising due to incorrect implicit / explicit data type conversion
- Permission errors
  - No permission to access a required resource (file, database table, etc.)
- Version mismatch errors
  - Library version not the same version software was intended to be used with
- Distributed system errors
  - Error while communicating between different parts of distributed system
  - E.g. Error in data marshalling / demarshalling between client / server
- Interface errors
  - Error arising from developer misunderstanding behavior of an API

Now Please Read Textbook Chapter 7