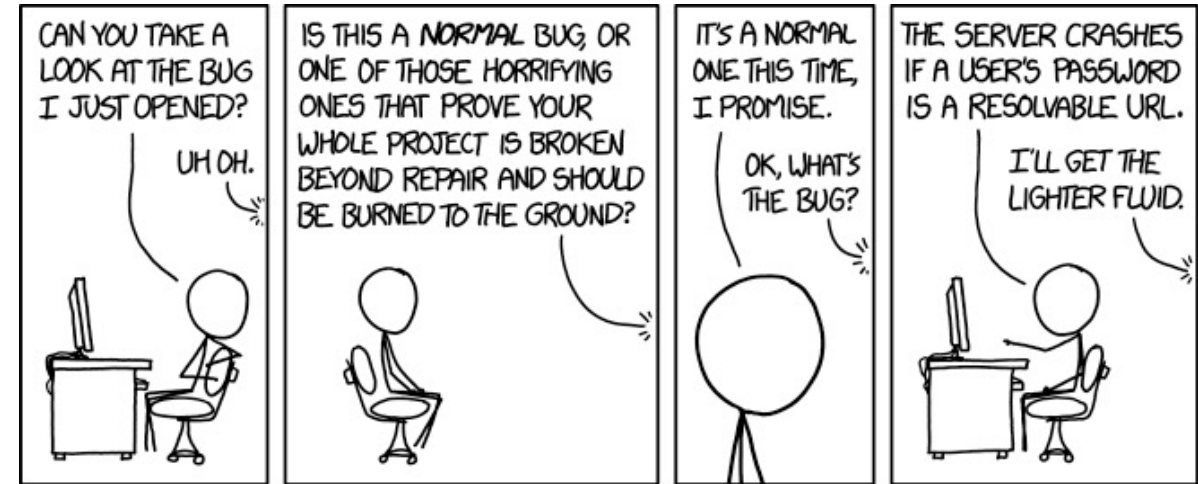


# ENGSCI 233 Lecture 7

## Software Testing and Error Handling



Bryan Ruddy and Andreas Kempa-Liehr




UNIVERSITY OF  
**AUCKLAND**  
Waipapa Taumata Rau  
NEW ZEALAND

**ENGINEERING**  
DEPARTMENT OF ENGINEERING SCIENCE

# The very first bug

9/9

0800 Antan started  
 1000 " stopped - antan ✓  
 1300 (032) MP - MC 1.58264000  
 (033) PRO 2 2.130476415  
 conv 2.130676415  
 Relays 6-2 in 033 failed special speed test  
 in relay 10.000 test.  
 Relays changed  
 1100 Started Cosine Tape (Sine check)  
 1525 Started Multi-Adder Test.  
 1545  Relay #70 Panel F  
 (moth) in relay.  
 First actual case of bug being found.  
 1630 Antan started.  
 1700 closed down.

Relay 3345  
 Relay 3370



Rear Admiral Grace Brewster Murray

In 1946, when Hopper was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the [Mark II](#) and [Mark III](#). Operators traced an error in the Mark II to a [moth](#) trapped in a relay, coining the term *bug*. This bug was carefully removed and taped to the log book. Stemming from the first bug, today we call errors or glitches in a program a *bug*.<sup>[8]</sup>

# Today's objectives:

- Understand the concepts of unit testing and edge cases
- Write unit tests for simple functions
- Use asserts and raise errors to manage bad input
- Catch and handle errors to prevent program crashes

# Good code does more than just work.

- Who is going to use it?
- Will they use it the way you expect?
- Will the computer work the way you expect?



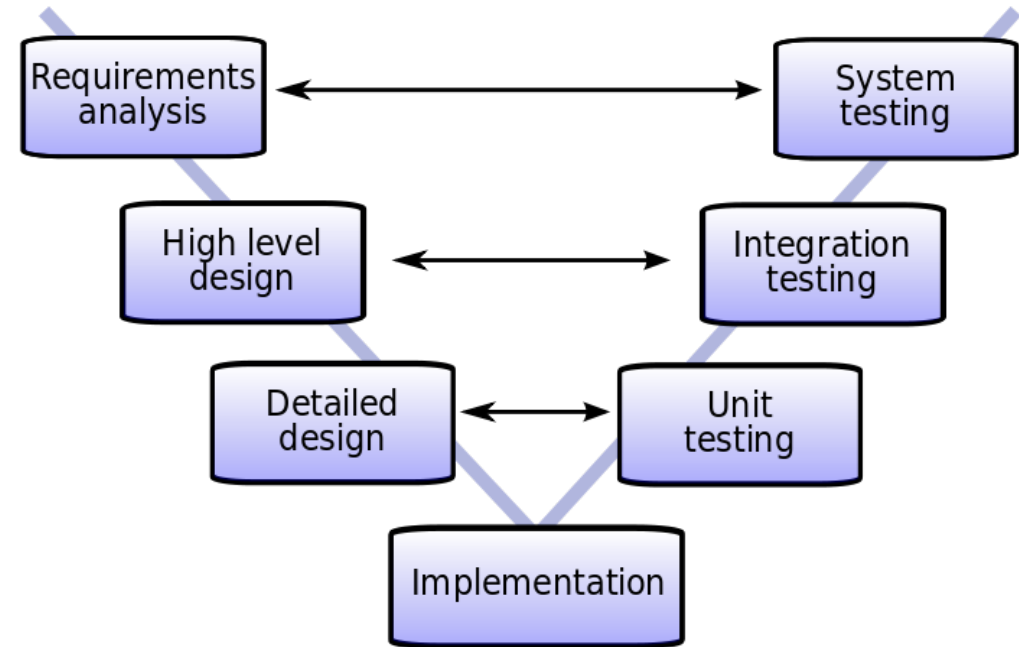
# How do we know we can trust your code?



- Black Box Testing
  - Ignore how it's written
  - Just make sure output is right
- Formal verification/proof
  - Ensure only the correct output is possible

# What forms can testing take?

- Unit testing
  - Tests a single function at a time
- Integration testing
  - Tests several functions together
- System testing
  - Tests whole software program
- Hardware-in-the-loop testing
  - Tests software and hardware together



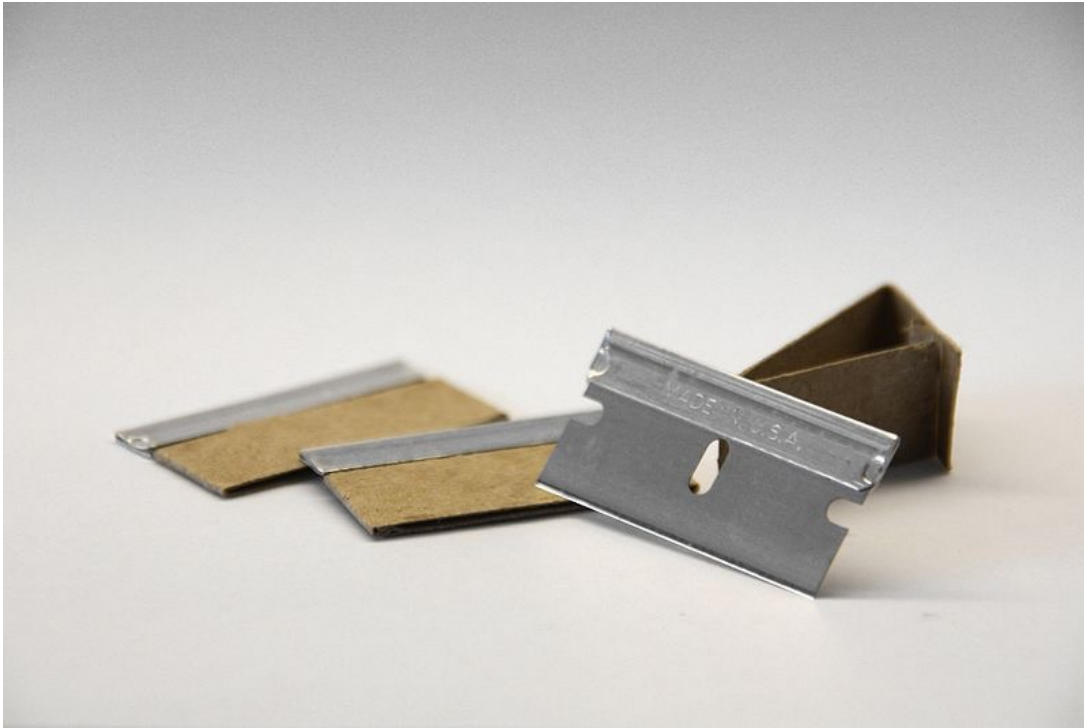
# Unit tests help you test as you go.

- Test just one function
- Test all conceivable inputs
- Make sure outputs are correct
- But, how on earth can we test everything?
  - Most inputs act the same
  - Check typical ones and edge cases
  - Try to use every side of every conditional





# Pay special attention to edge cases.



- Edge cases have tricky or counterintuitive behaviour
- Typically on the boundary between domains of consistent behaviour
- Often have inputs or outputs of zero or infinity
- Could also be values you know need special handling



# How much do you need to do?

- How important is your software?
  - Will people die if it fails?
  - Will it cost millions of dollars?
  - Will it ruin your reputation?
  - Do you control how it will be used?
- 100% code coverage may not be practical



# Automated testing helps.



- You might write lots of unit tests!
- There are systems to help run them all
- `pytest` is easy to use
  - It's in the lab...

# How should you handle bad input?

- Errors are an important part of healthy code
- Assert statements are good for testing
- Standard kinds of errors are more communicative
  - `ZeroDivisionError` > `AssertionError`

```
[In [6]: assert 2 * 2 == 4
[In [7]: assert 2 * 2 == 2
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-7-2d4f72fb84fe> in <module>
----> 1 assert 2 * 2 == 2

AssertionError:
```

```
[In [1]: def inverse(x):
...:     return 1 / x
...:

[In [2]: inverse(2)
Out[2]: 0.5

[In [3]: inverse(0)
-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-3-7538d73c586c> in <module>
----> 1 inverse(0)

<ipython-input-1-9fb9d397eff2> in inverse(x)
      1 def inverse(x):
----> 2     return 1 / x
      3

ZeroDivisionError: division by zero
```

# You can create your own error conditions.

```
In [12]: def inverse(x):
...:     if type(x) == str:
...:         raise ValueError("The inverse of a string is not defined.")
...:     return 1 / x
...:

[In [13]: inverse(4)
Out[13]: 0.25

[In [14]: inverse('4')
-----
ValueError                                Traceback (most recent call last)
<ipython-input-14-a5c3ea4e63e6> in <module>
----> 1 inverse('4')

<ipython-input-12-64c48259748d> in inverse(x)
      1 def inverse(x):
      2     if type(x) == str:
----> 3         raise ValueError("The inverse of a string is not defined.")
      4     return 1 / x
      5

ValueError: The inverse of a string is not defined.
```

- `raise` causes the named error
  - Any error you have seen can be manually raised
    - `IndexError`, `NameError`, `ValueError`, etc.
    - You can pass parameters to give more information
- You can create custom errors, but it takes work.

# What happens if the real world intrudes?

- Good code may still experience errors
  - Lost internet?
  - Someone removed the USB stick?
  - The ventilator display cable came unplugged?
- You need to handle them anyway.



# Catch the errors and handle them.

```
In [19]: def inverse(x):
...:     if type(x) == str:
...:         raise ValueError("The inverse of a string is not defined.")
...:     return 1 / x
...:

[In [20]: inverse(5)
Out[20]: 0.2

In [21]: try:
...:     inverse('5')
...: except ValueError:
...:     print("I new it. You cannot compute the inverse of strings.")
...:
I new it. You cannot compute the inverse of strings.
```

- `try ... except` is the error handling structure in Python
- You can catch specific error or all errors
- You can make errors invisible to the user
- You can propagate an error upstream if you can't fix it

# Only handle the errors you can handle.

- Catching errors you don't deal with is dangerous
- Don't indicate error conditions with special output values unless they're otherwise impossible
- Make sure you clean up the messes
  - Finish saving data, putting equipment in a safe state, etc.





# Image References

Slide 1: *New Bug*, by Randall Munroe, from <https://xkcd.com/1700/> (CC BY-NC 2.5)

Slide 2 left: U.S. Naval Historical Center Online Library Photograph NH 96566-KN

Slide 2 right: James S. Davis, United States Navy ID DN-SC-84-05971

Slides 4, 9, 13: by أمين, from [https://commons.wikimedia.org/wiki/File:Medical\\_ventilator\\_001.jpg](https://commons.wikimedia.org/wiki/File:Medical_ventilator_001.jpg) (CC BY-SA 4.0)

Slide 5: by unlimicon, from <https://thenounproject.com/term/box/566042/> (CC BY 3.0)

Slide 6: by Herman Bruyninckx, from <https://commons.wikimedia.org/wiki/File:V-model.svg> (CC BY-SA 3.0)

Slide 7: by IOHK Design Department, from <https://thenounproject.com/term/unit-testing/2199735/> (CC BY 3.0)

Slide 8: by Zephyris, from <https://commons.wikimedia.org/wiki/File:RazorBlades.jpg> (CC BY-SA 3.0)

Slide 15: by Matthew Inman, from [https://commons.wikimedia.org/wiki/File:Tumbeasts\\_servers.png](https://commons.wikimedia.org/wiki/File:Tumbeasts_servers.png) (CC BY 3.0)