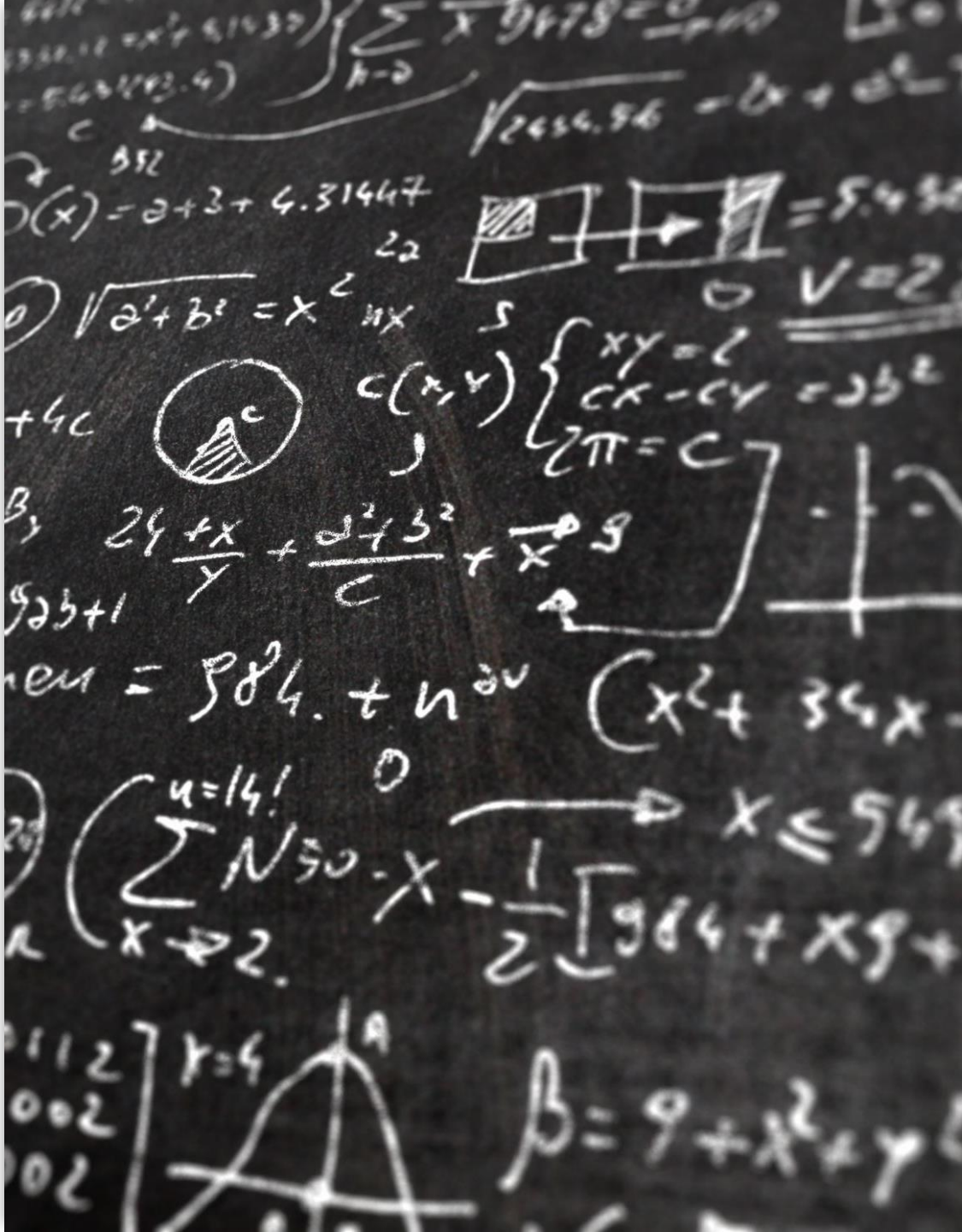
School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 7: Testing

Adrian Friday, Nigel Davies, Hansi Hettiarachchi, Saad Ezzini

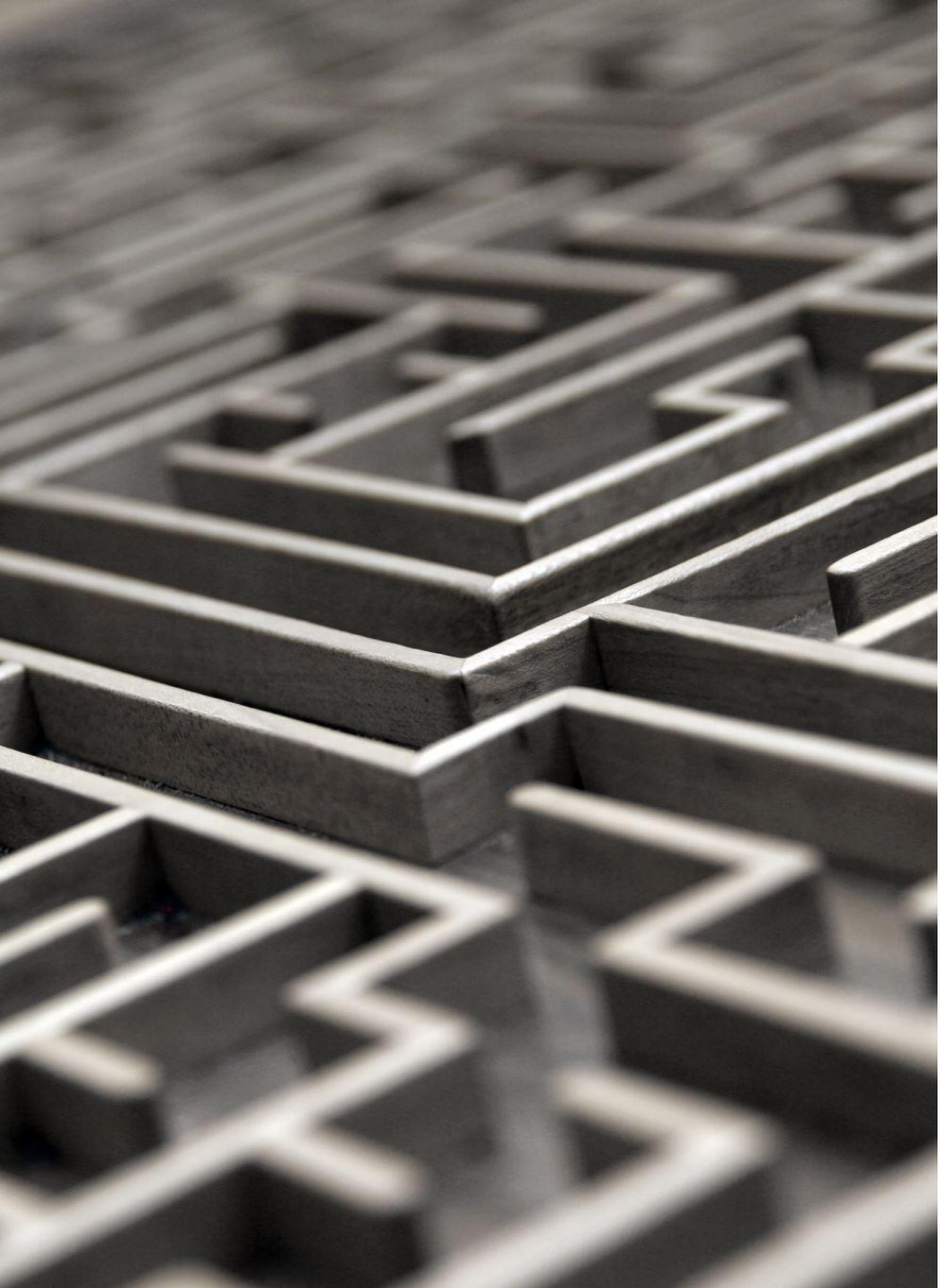
This lecture

- How to systematically test your code to find logical errors
- Effective test case choice
- Writing test cases



Programs that *aren't tested* are worse than useless

Most *developers* don't like testing – *it's perceived as simply confirming something they already know*

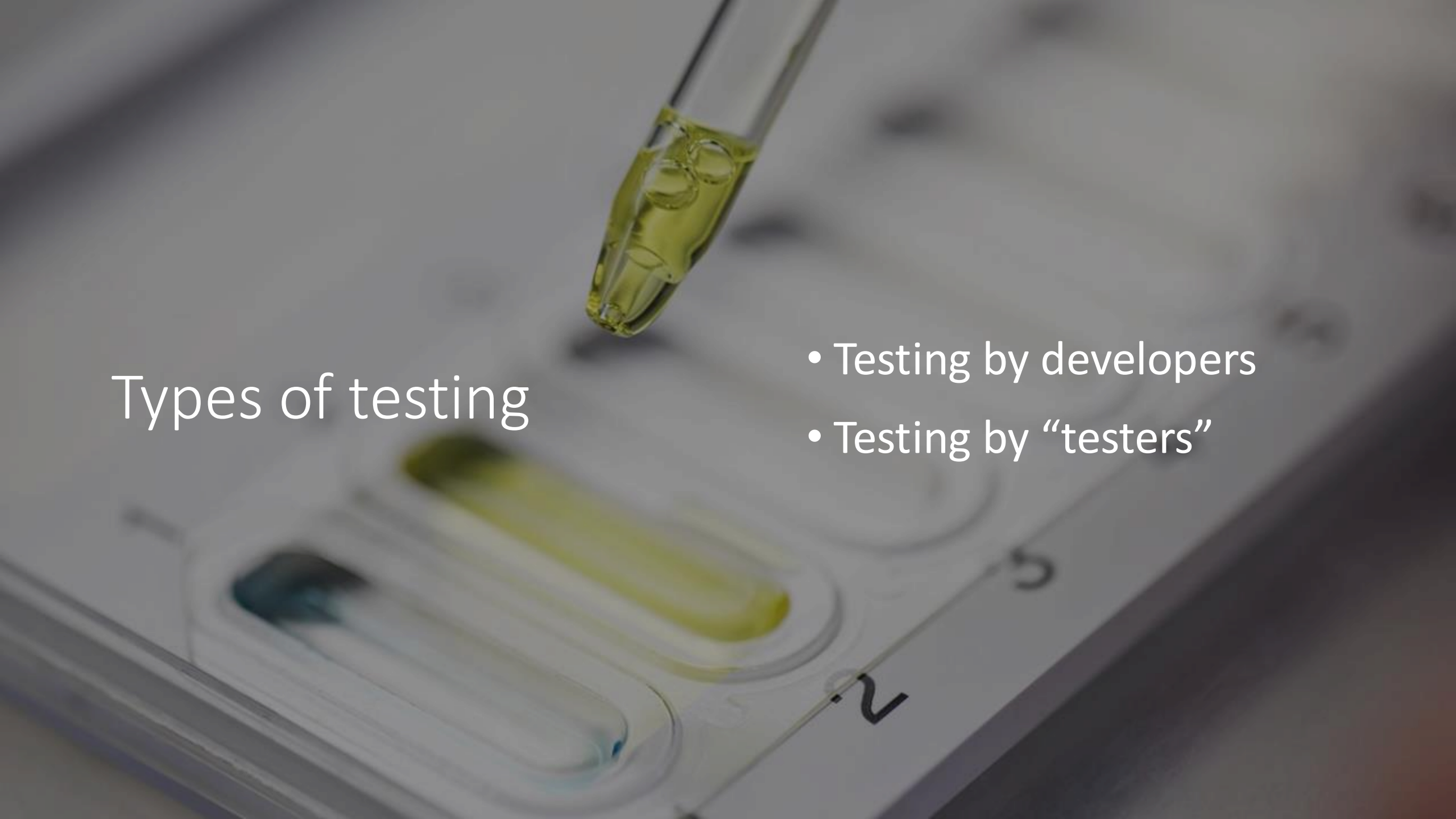


Many *testers* don’t like testing - *it’s seen as repetitive and a pretty thankless task*

Testing is absolutely vital!

It’s the only way to produce software of any value

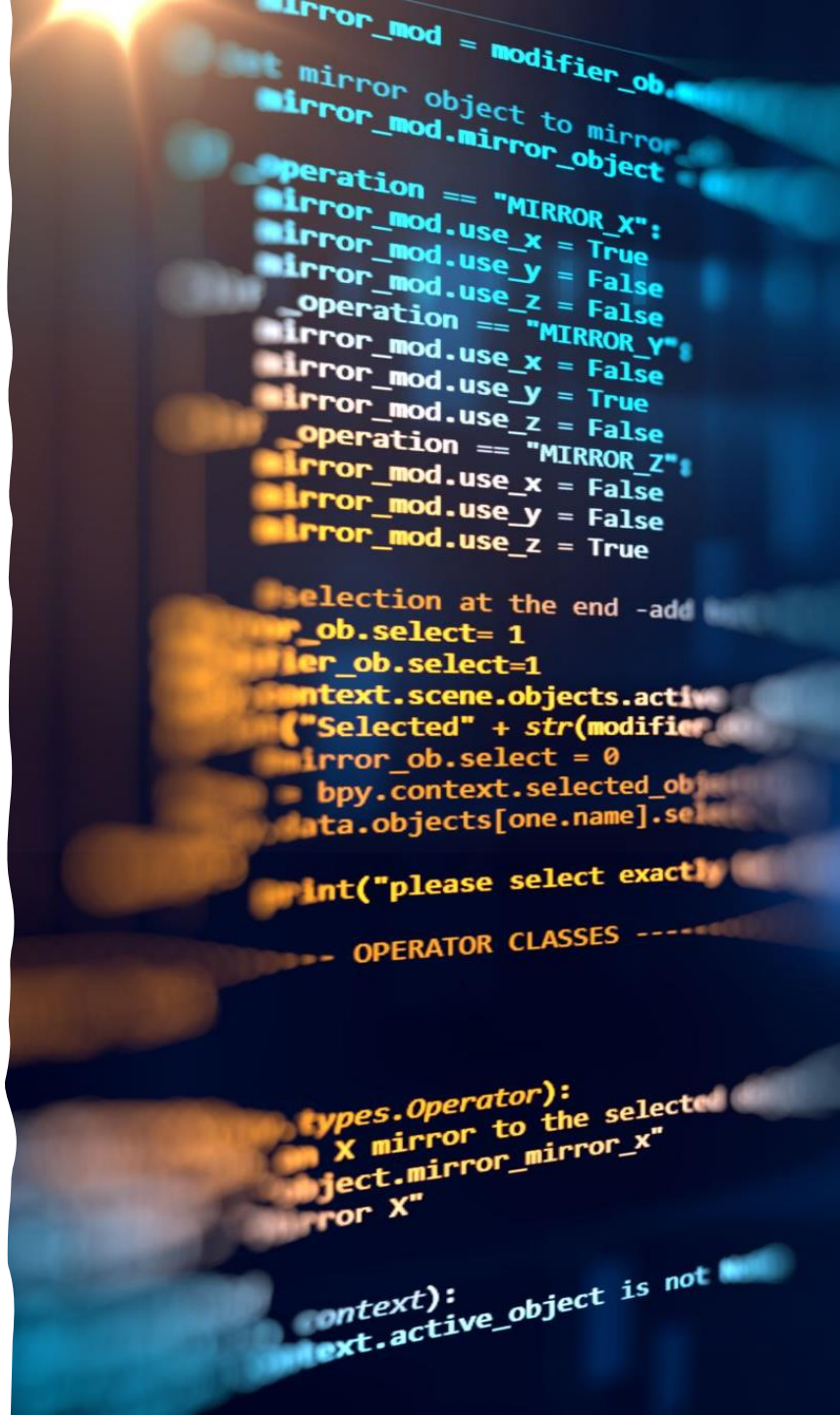
- Would you like to take an untested medicine or drive an untested car ?



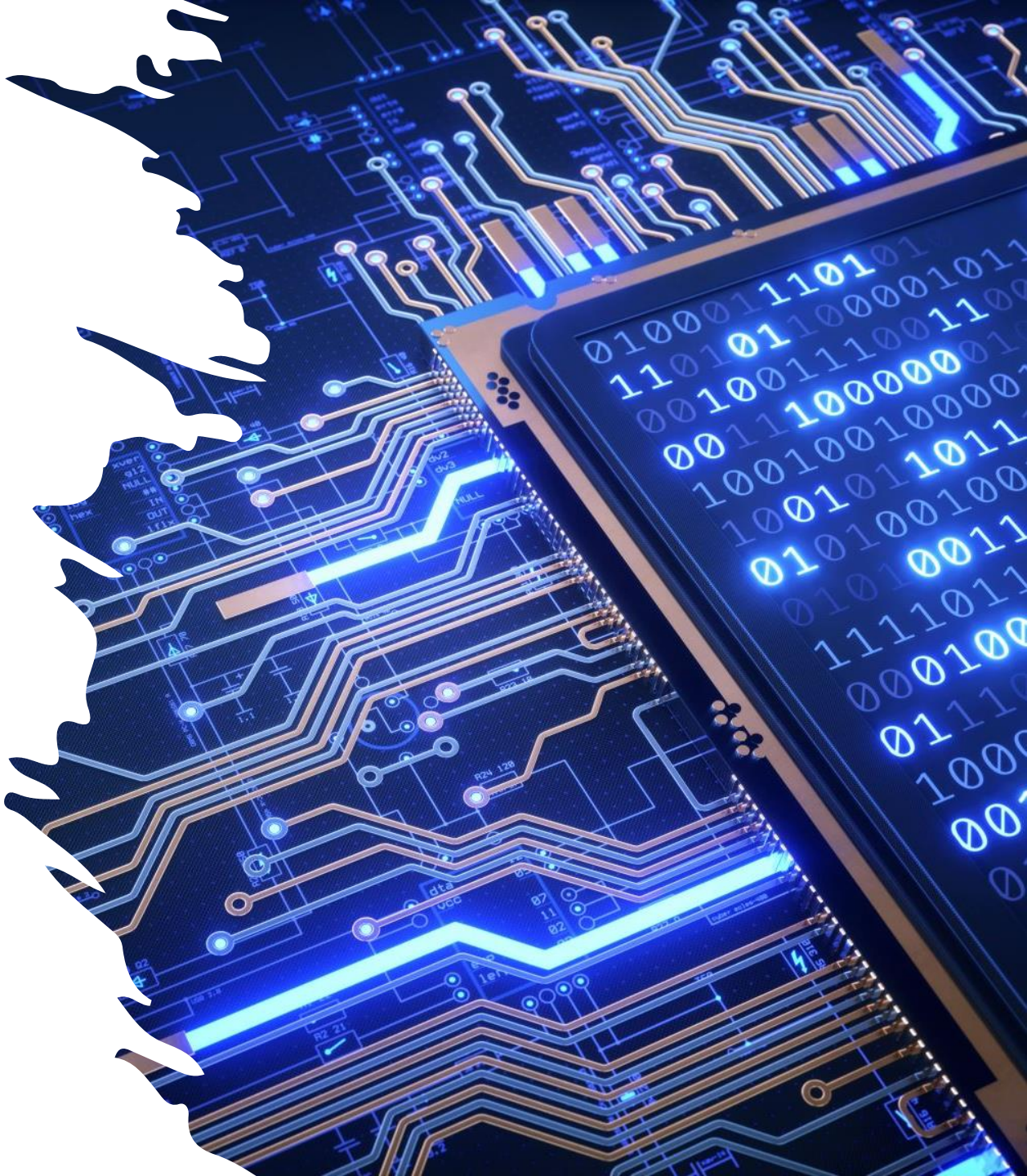
Types of testing

- Testing by developers
- Testing by “testers”

Developers try and make sure the software works *according to their understanding* of the spec



Testers try and make sure the software works according to *everybody else's* understanding of the spec :)



Behavioural vs. transparent box testing

- When constructing your test cases do you use knowledge of the internals of the program ?
- Mostly developers do transparent box testing and testers do behavioural testing

Running an effective test program is a **really** interesting and challenging problem

Requires a particular mindset similar to a spy, police officer, special forces unit, code breaker, hacker etc.

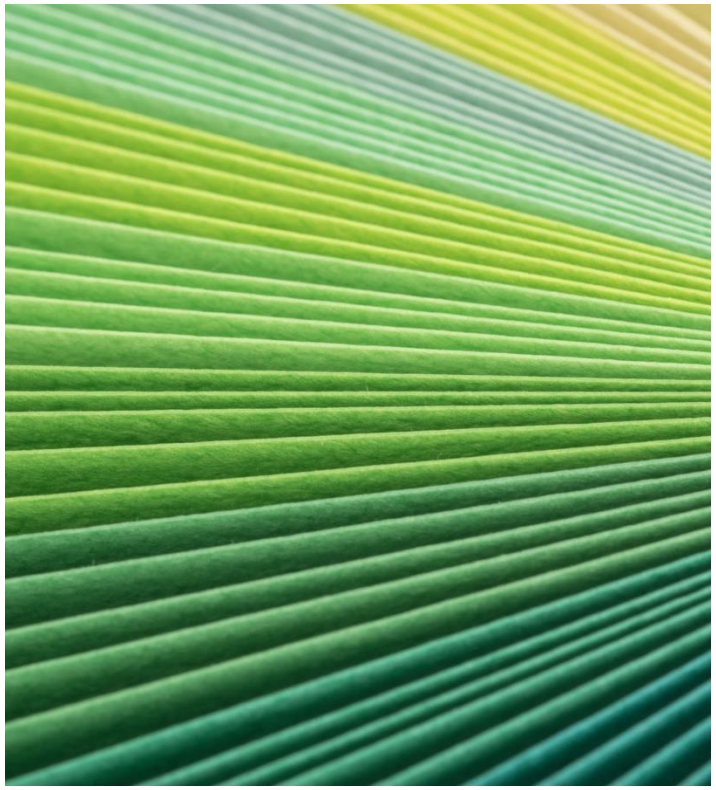
Simple tests

Does it work for a simple representative set of inputs ?

Does it work for all possible inputs ?

this is probably too hard in practice, so we look for boundary cases

Does it work means does the *output* match the *expected output* ?



An example

- Your program is supposed to let the user enter a score between 1 and 10 and print fail, pass, distinction:
  - 1-4 fail
  - 5-7 pass
  - 8-10 distinction

What could possibly *go wrong*?

```
#include <stdio.h>

/* Function to turn a numeric grade from 1 to 10 into an outcome */

void print_outcome_from_grade(int grade)
{
  // 1-4 fail

  if (grade >= 1 && grade <= 4)
    printf("fail\n");
  else
    // 5-7 pass

    if (grade >= 5 && grade <= 7)
      printf("pass\n");
    else
      // 8-10 distinction

      if (grade >= 8 && grade <= 10)
        printf("distinction\n");
}
```

Some sample test cases

- Normal operation
  - 2, 6, 9
- Boundary cases
  - 1, 4, 5, 7, 8, 10
- Invalid cases
  - 0, 11, -1, ‘a’
- Uncertain
  - 1.5, 0.5, 10.4

- Your program is supposed to let the user enter a score between 1 and 10 and print fail, pass, distinction:
  - 1-4 fail
  - 5-7 pass
  - 8-10 distinction

Designing “good” tests

Example +ve characteristics

- The tests are designed to help find errors !
  - not to demonstrate that the program works !
- They ideally isolate and test certain expected behaviours
- They are *repeatable*
- They are *necessary*

Summary

- You should know about the importance of software testing
- The difference between behavioural and transparent box tests
- How to choose good/effective test cases
- Be expecting to test your code to make sure it works/ meets the specification