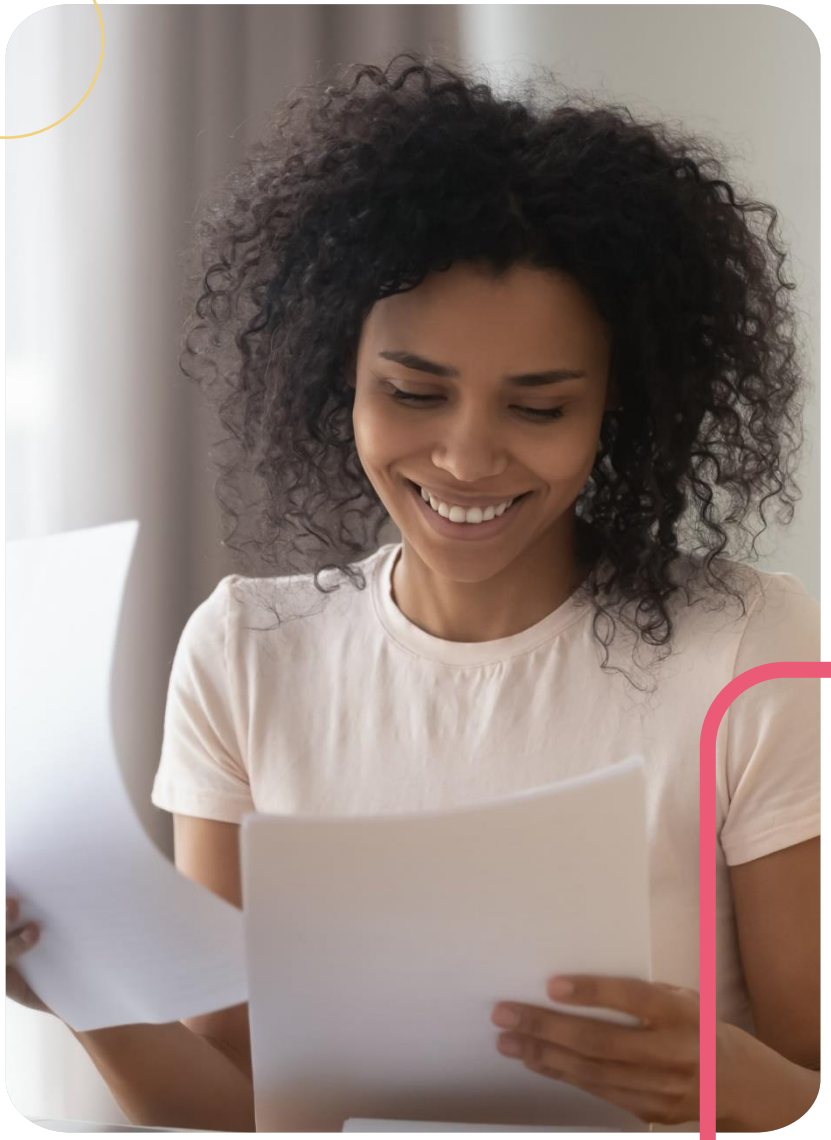
Diploma in

Computer Science

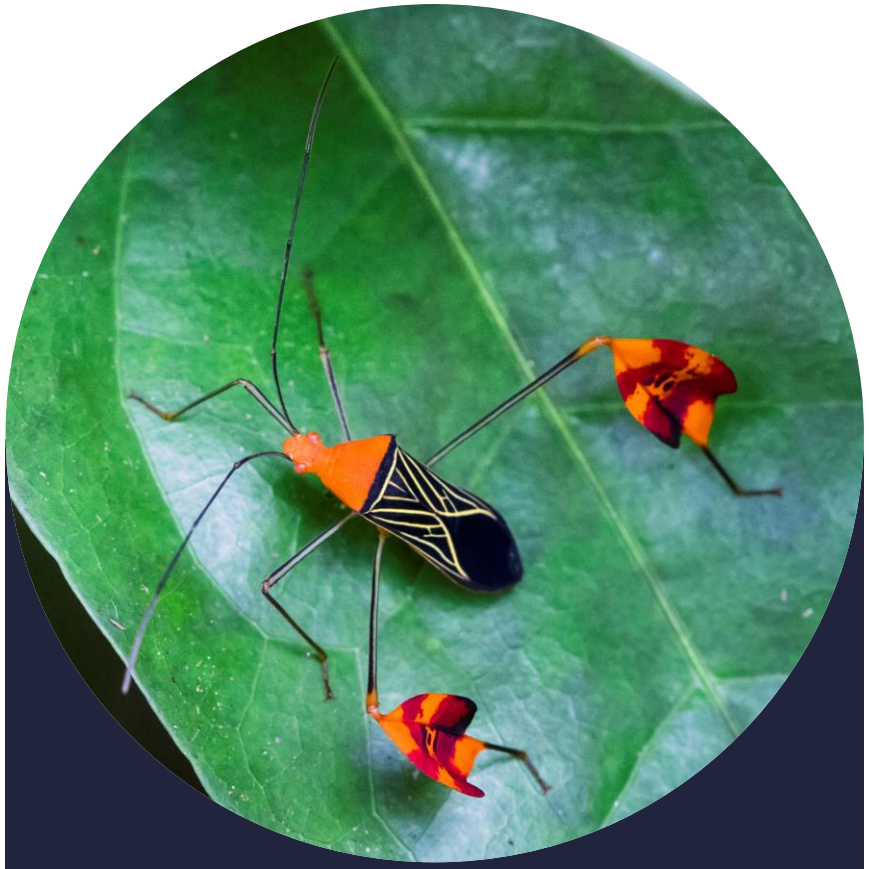
Debugging



Objectives

- Identify bugs and how they affect code
- Explain the most common debugging resources
- Become aware of strategies used for debugging

Bugs in the computer world



What is a computer bug?

- An error, flaw, or fault that causes the program to behave unexpectedly or produce unexpected results
- Applies to both computer hardware and software
- Result of errors made by developers during development process

The first computer bug

- Term originates from the 1800s
- First instance was in 1947
- First bug was actually a moth

Bugs vs viruses

A virus can cause your computer to behave in unexpected ways.

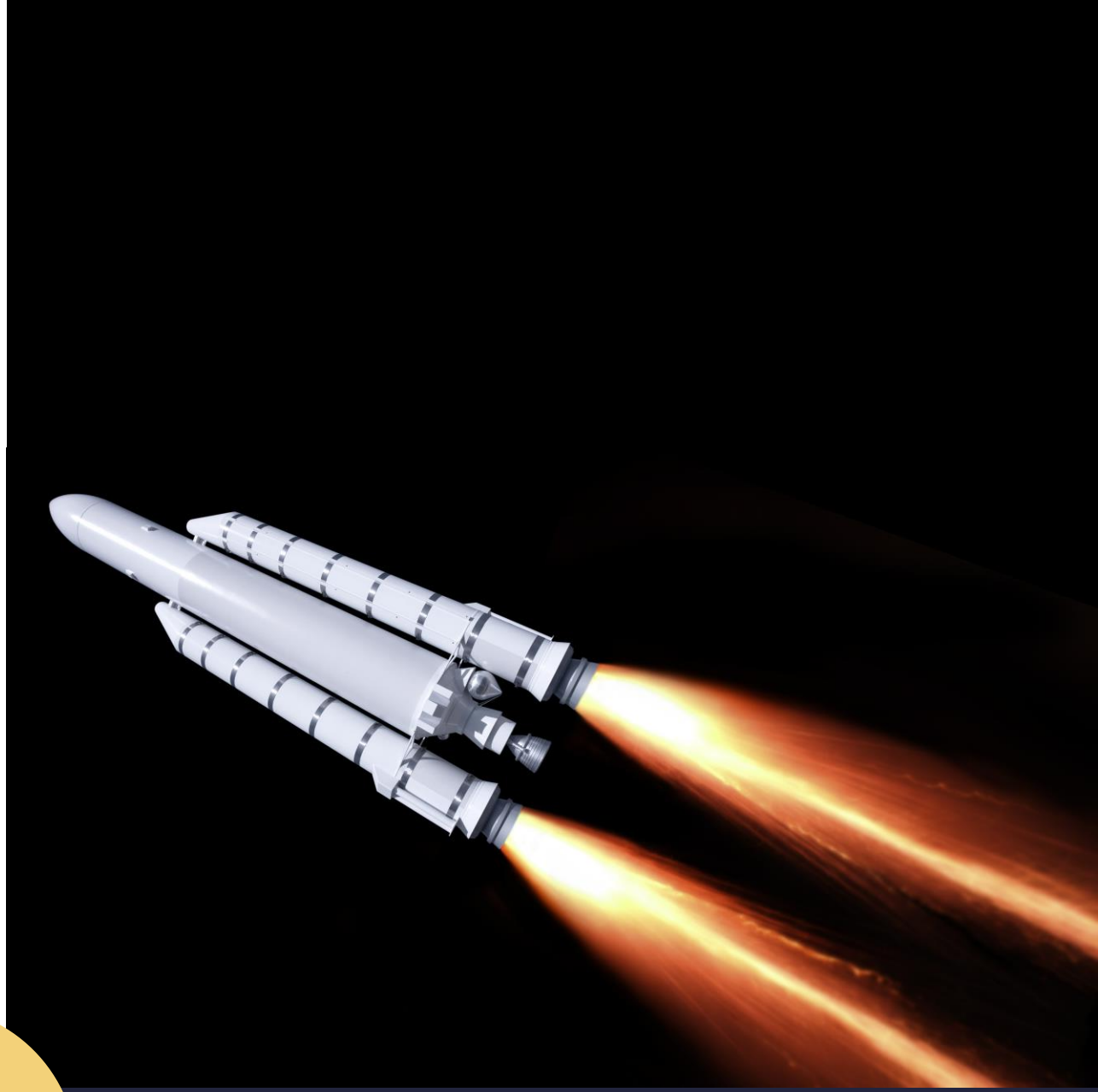
A bug is unintentional.

A virus is intentional.

A bug is a programmer’s error.

Bugs can cost a fortune – Ariane 5

- One of the most expensive software mistakes in history
- European rocket used to launch satellites
- Exploded 40 seconds after launch
- Crashed because of an integer overflow error



Bugs can cost a fortune – Y2K

- Standard practice to represent years with the last 2 digits
- On 1 January 2000, many computers read the year as 1900
- Affected many industries
- Cost billions of dollars worldwide to upgrade systems

Bugs can cost a fortune – NASA

- Mars Climate Orbiter cost approx $125 million
- Orbiter lost communication with Earth and entered mars atmosphere at wrong angle
- Software used imperial units instead of metric units

Did you know?

The song ‘Gangnam Style’ broke YouTube!

When it was released, it racked up 3 billion views which exceeded the maximum value on YouTube.

Types of bugs

Software defects classified by nature

- Functional
- Performance
- Usability
- Compatibility
- Security

Functional defects

- Software behaves in a way that is not compliant with functional requirements
- Discovered during functional testing

Example: when a system that should only accept integers accepts floating point numbers

Performance defects

- Software does not perform at acceptable speed and stability
- Software package should use minimal resources with maximum possible throughput
- Should be as stable as possible

Example: mouse click registering five seconds later

Usability defects

- Software package is difficult to use or navigate or is unnecessarily complex

Example: a sign-up process that has seven stages instead of two

Compatibility defects

- Software doesn’t show consistent performance and behaviour across various platforms

Examples: website that doesn't show all elements on a mobile device OR operating system that doesn’t work across all Android versions

Security defects

- A software package might work correctly but comes with security defects
- Attack surface must be kept at a minimum

Examples: weak authentication, buffer overflows, encryption errors

Software defects classified by severity

- Critical
- High
- Medium
- Low

Critical severity

- Highest level
- Should be addressed immediately
- Testing cannot proceed without fixing

Example: application that doesn’t store input

High severity

- Defect affects the key functionality of an application
- Software package may work in some way

Example: two-person login system that only allows one person access

Medium severity

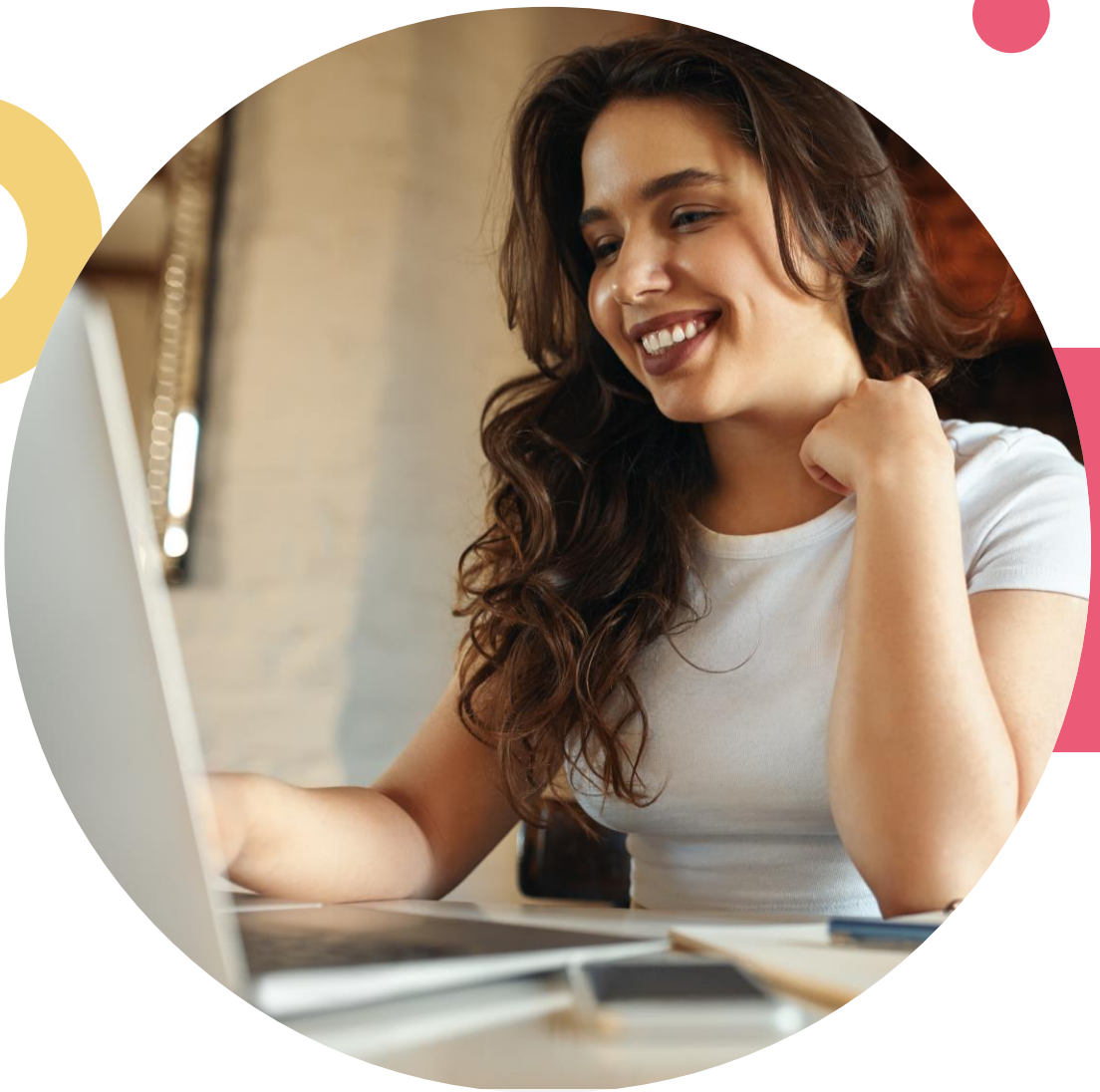
- Refers to minor defects
- A minor function exhibits behaviour that is contrary to what is specified in the requirements but does not affect the core functionality of the program

Example: ABOUT US page displaying on a website

Low severity

- Typically minor defects
- Production team can still release software

Example: text not aligned properly



Software defects by priority

- Urgent
- High
- Medium
- Low

Urgent priority

- Need to be fixed immediately
- Not limited to severity
- Project manager identifies the priority of the bug

High priority

- Scheduled to be fixed in upcoming release
- User can make do with the system as is
- Typically in software package's exit criteria

Medium priority

- Errors that can be fixed in any subsequent release
- Changes can be skipped in current release

Low priority

- Can be dealt with when there is time
- Don’t adversely affect the software package

To summarise...

- Allocating the correct severity and priority level speeds up defect fixes and increases capacity and efficiency
- Crucial for project milestones and deadlines





Avoiding bugs

Keep code simple

- Avoid code that has internal structure – each function should do one thing
- Functions are interconnected using shared variables and parameters
- Putting code in ‘containers’ makes it easier to isolate and test

Use well-tested libraries

- C and other programming languages allow you to create your own libraries
- Use a well-tested library instead of writing your own code

The process of debugging a software package can take more time than it initially took to write the program.

Debugging

Debugging

- Identify the error
- Locate the error
- Analyse the code
- Prove the analysis
- Build the fix
- Test and validate

Step 1: Identify the error

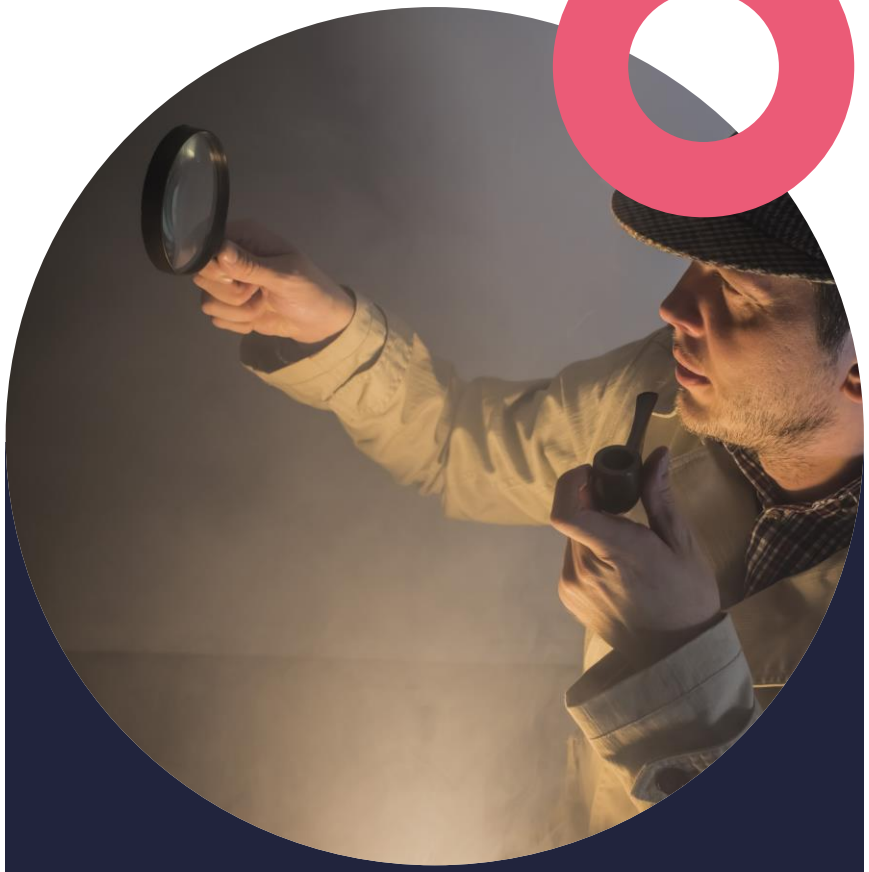
- A badly-identified error can lead to wasted development time
- Production errors are typically reported by users and are hard to interpret
- Key is to identify the actual error

Step 2: Locate the error

- Error needs to be located in the code
- Need to know where it is coming from

Step 3: Analyse the code

- Understand what the code is doing against what it is supposed to do
- Helps to understand the error
- Two main goals:
  - Check for ripple effects
  - Assess risks of collateral damage



Step 4: Prove the analysis

- Documents bug and issues
- Info used for writing automated tests in a test framework

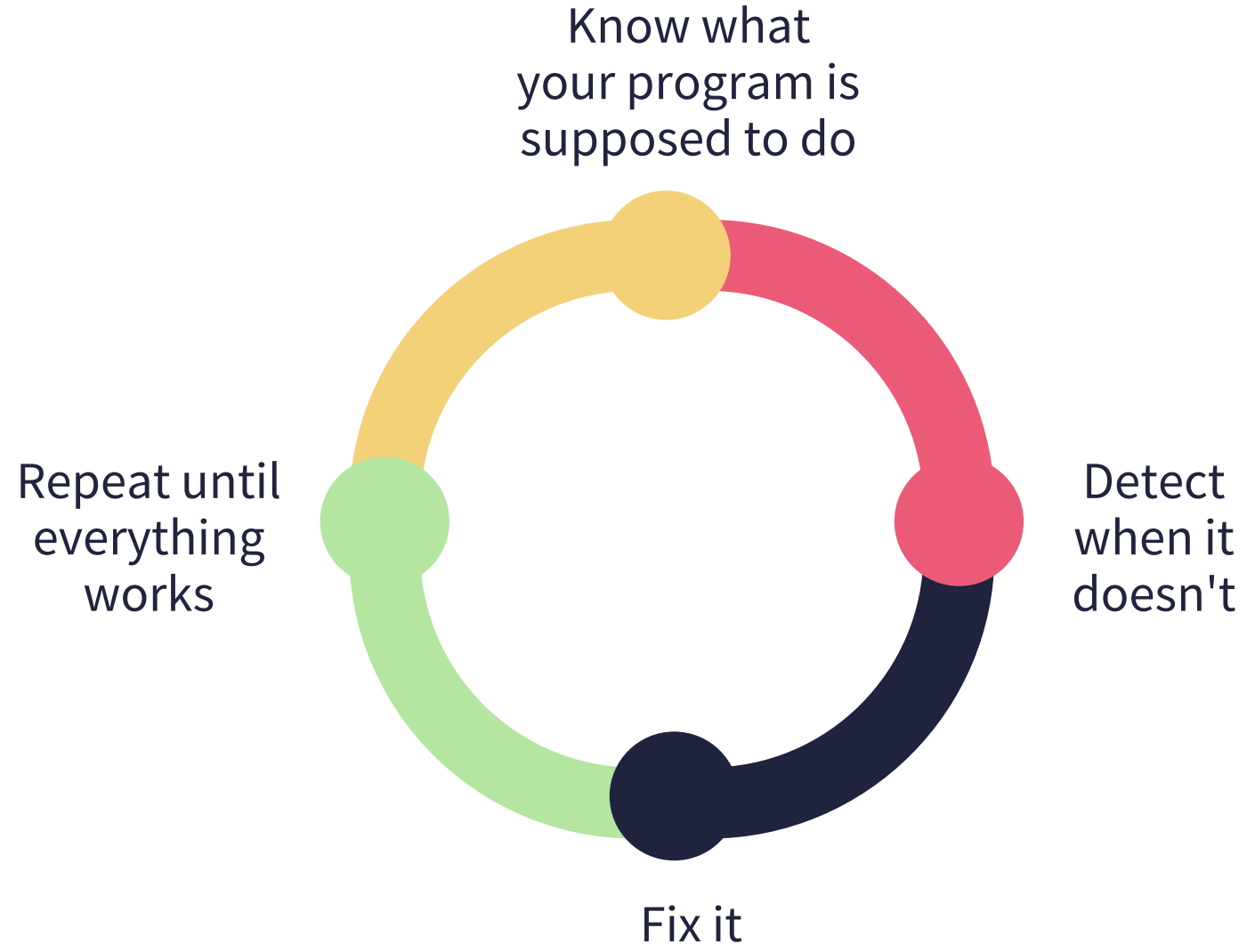
Step 5: Build the fix

- Actual code is written
- Added to the existing code

Step 6: Test and validate

- Program is tested again
- Validation – determines if added code causes any new problems

Basic steps for debugging



Don’t…

- Focus on making whatever code we have work in whatever
- Try to debug the code by just looking at the output

Do add asserts into your code

```c
#include <assert.h>
int main(int argc, char
**argv)
{
 assert(1+1 == 10);
 return 0;
}
```

Example

For Linux, use the gdb debugger

- Follows along as your program runs
- Can see what the program did every step
- Some versions of Windows compilers come with the tool too

Linux

Use valgrind program

- Good for pointers and dynamic storage allocation
- Produces report of allocations and deallocations
- Can also check for memory leaks

Breakpoints

- Places where debugging tools stop when they encounter problems
- Execution is halted intentionally
- Tools employ breakpoints by testing for certain conditions at certain points in the program
- Can be set in different ways to suit your needs

Debugging strategies

- Automated testing
- Change development strategy
- Logging
- Cluster errors
- Debugging output
- Change your attitude

Lesson 5
Challenge

Try to debug the code in the demo at the end of this lesson.

Automated testing

- Widely used in the software
- Carried out using a testing framework
- Makes routine testing take less time
- Key feature of agile development methods

Change development strategy

- Incremental and bottom-up program development allows programmers to isolate errors
- Reduces number of errors

Logging

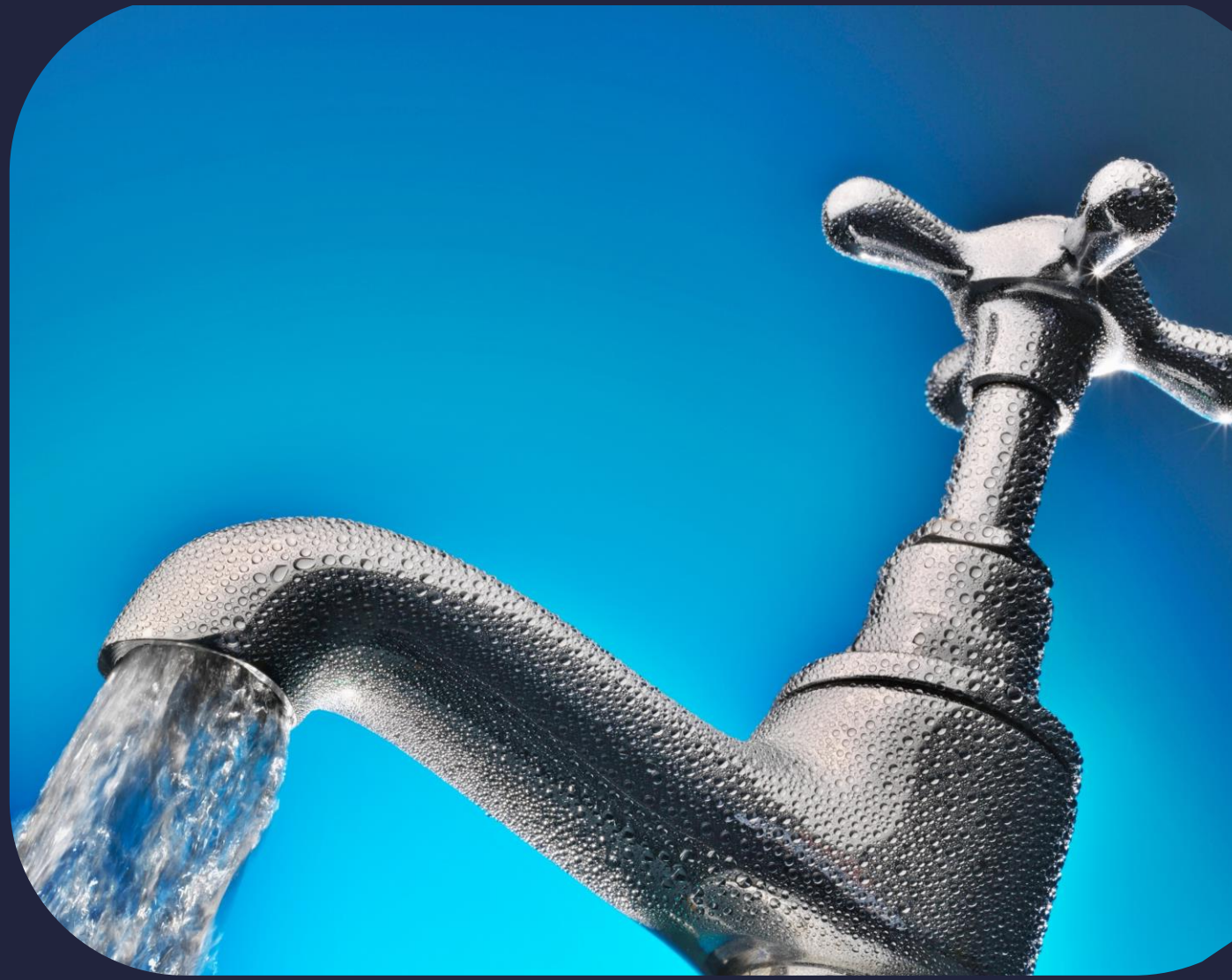
- Insert printf statements that write to a file called a ‘log’
- Each critical step in the program is acknowledged in the file
- Useful to pinpoint errors in deployment

Cluster errors

- Group similar bugs then examine only one bug from each class
- Addressing the root problem can fix the bug
- Helps localise problems

Debugging output

- Not recommended as it takes too much time to go back to source code
- Only really useful if you understand the source code
- If there are patterns, can pick up where error occurred and go back to source code



Change your attitude

- Keep an open mind
- Ask yourself where the bug is NOT
- Articulate the problem
- Inspect input data and test harness

Change your attitude

- Check the source code
- Ensure correct libraries are linked
- Take a break to clear your mind