University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Error Taxonomy

Specification Left Some Issues Unaddressed

Does your think-pair-share solution work if the human types no values (starts with -1)?

No (fewer than 1% of students have identified the issue while solving this problem).

Why not?

Our specification did not cover this possibility!

Specification Ambiguities Can be Hard to Find

This type of error is
- called a **specification ambiguity**
- (not in P&P's taxonomy—we will add it).

Why did it happen?

Why didn't you define the interface fully?

People can't think of everything.

Specification ambiguities are **difficult to uncover** (no one thought of them).

Ask Someone to Help you Spot Bugs in Your Code

What else do our think-pair-shares teach us?

Code reading!

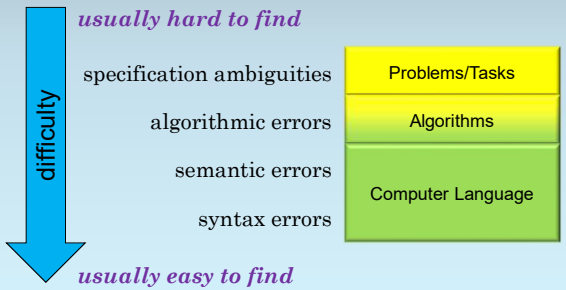
Caveat:
- writing on paper/chalkboard
- under time pressure
- is not the best way to produce error-free code.

However, even when
- coding leisurely
- with computer tools to help,
- **someone else is more likely to spot mistakes**.

How to Make Code Reading Successful

1. **Get a friend** (not in the same class, unless >1 person is allowed on assignment!).
2. **Friend SITS AT KEYBOARD and reads** your code.
3. **You** sit nearby and **answer questions** or provide minor guidance.
4. **Code must be neat and well-commented** (unless you're trying to lose the friend).
5. **Did you read point #2?**

Error Taxonomy Relates to Abstraction Layers



Examples of Specification Ambiguities

Let's start from the top.

Specification Ambiguities

Examples include
- unspecified elements (as in t-p-s),
- assumptions buried in code (Ariane 5),
- assumptions forgotten/misused
- choices made differently but never communicated

Avoid Assumptions, and Document Any That You Make

Avoid assumptions when possible:
- if an assumption saves 10 lines of code,
- just write the extra code (don't make the assumption).

Document any assumptions
- that you must make, and
- communicate them to others.

Assert Requirements and Avoid Overloading Meaning

Assert all requirements
- (usually at module boundaries),
- meaning that **if a requirement is not met,**
- **crash the program**
  (deliberately and immediately).

Avoid overloading meaning.
- What does `print_square (-10)` mean?
- Draw a triangle of size 10?
- (No!)

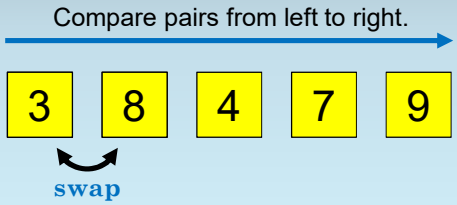
Algorithmic Errors are Algorithms that Fail

What is an **algorithmic error**?

- Use of **an algorithm that does not solve the problem**.

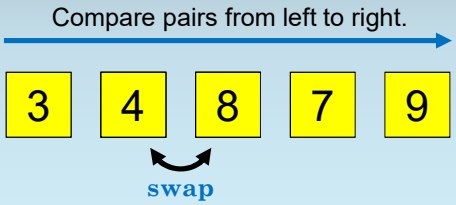
- There are two types:
  - logical and
  - numerical.

A Great New Idea: Loop Swap Sort!



Swap iff right value smaller than left value.

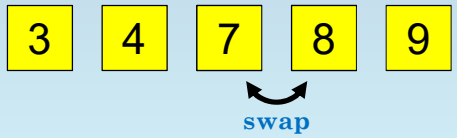
Loop Swap Sort! (Step 2)



Swap iff right value smaller than left value.

Loop Swap Sort! (Step 3)

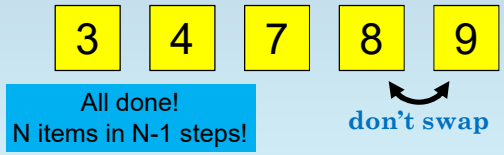
Compare pairs from left to right.



Swap iff right value smaller than left value.

Loop Swap Sort! (Step 4)

Compare pairs from left to right.



Swap iff right value smaller than left value.

Counterexample for Loop Swap Sort

Try "loop swap sort" on these...

… oops.

Loop Swap Sort is an Example of an Algorithmic Error

"Loop swap sort"
- is an example of a **logical algorithmic error**.

The algorithm
- works on some inputs
- but not in general.

Algorithmic Errors are Hard to Find

Algorithmic errors are also hard to find.

Why?

Had programmer
- thought of an input
- for which the algorithm fails,
- the programmer would have chosen a different algorithm!

What Day is It?

I need your help again…

220 midterms are on Thursdays.

This year,
Valentine's Day
fell on Wednesday.

What about next year?

Time for Some Computation

How many days in a year? 365

How many days in a week? 7

What's 365 % 7? 1

So …

A Different Day … Every Seven Years?

14 Feb 2017	Tuesday	**Correct!**
14 Feb 2018	Wednesday	
14 Feb 2019	**Thursday**	**Right?**
14 Feb 2020	Friday	
14 Feb 2021	Saturday	
…		
14 Feb 2026	**Thursday**	**Still right?**

Example of Numerical Algorithmic Error

What do you mean, "leap year?"

Oh … so … in some years,
we change by two days of the week?

That's an **example of** a fairly simple **numerical algorithmic error**.

Example of Numerical Algorithmic Error

Here's a story about a more complex one…

In the mid-1990s,
- US National Labs were moving from
- Cray supercomputers to MPPs.*

They wanted some assurance that their programs produced the same results.

*Massively parallel processors, supercomputers built from collections of smaller nodes.

Example of Numerical Algorithmic Error

At the Supercomputing conference in 1995 (or maybe it was 1996),
- David Abramson presented "differential debugging:"
- run a program on both machines and compare the results visually.

Then he said (paraphrased from memory),
- "I didn't write this example code.
- I just want to show you how differential debugging works."

Example of Numerical Algorithmic Error

As I recall, he showed results from a code that
- predicted how smog (air pollution)
- changed over time
- in the LA basin.

The two machines
- produced results of predicted smog (parts per million of particulate matter)
- that differed by 30%.

Example of Numerical Algorithmic Error

Using differential debugging,
- they were able
- to find the cause of the problem.

Any guesses?

Example of Numerical Algorithmic Error

On one machine, IEEE floating-point operations rounded up.

On the other machine, they rounded down.

One part in millions (or maybe even less) became 30% error.

The code was unstable.

That's **an example of a numerical algorithmic error.**

Test with Corner Cases and Use a Debugger

Test with corner cases:
- execute loops 0 or 1 times,
- examine equality case for loop tests,
- check both directions on conditionals, and
- think about possible overflows.

Walk through all paths for all code
- in a debugger,
- setting state as necessary
- (required by some companies).

Numerical Errors are Hard: Take a Class

Numerical errors are much harder!

Test corners cases for small code sections.

Use analysis for whole programs.*

*Take a numerical analysis class, such as ECE491.

Examples of Semantic Errors

Semantic errors are
- **mistakes in implementation** and are
- often typographic / small.

Code to print numbers from 1 to 10…

```
int32_t i;
for (i = 0; 10 >= i; i++); {
    printf ("%s\n", &i);
}
```


Avoiding and Fixing Semantic Errors

Walk through code in a debugger.

Use regression testing
- Bugs tend to resurface (more than one person makes the same mistake).
- **Every** time someone finds a **bug**, **add a test** that exposes the bug.
- **Pass all tests before committing changes.**

Syntax Errors and How to Avoid Them

Syntax errors are
- **Errors caught by a compiler** a
- (as errors or warnings).

Easy to avoid and fix:
- **Turn on all warnings** (`-Wall`).
- **Fix all warnings and errors**…
- …**but NOT by guessing**!

Summary of Techniques

1. Use code reading (and/or pair programming).
2. Avoid making assumptions.
3. Document assumptions.
4. Assert requirements.
5. Avoid overloading meaning.
6. Test corner cases.
7. Walk through all paths in debugger.
8. Use regression tests.

Good Testing and Debugging Takes Time

Note that many of these techniques are
- **standard in industry**, but
- **fairly time-consuming**, so
- you may put off trying them while you're a student.

