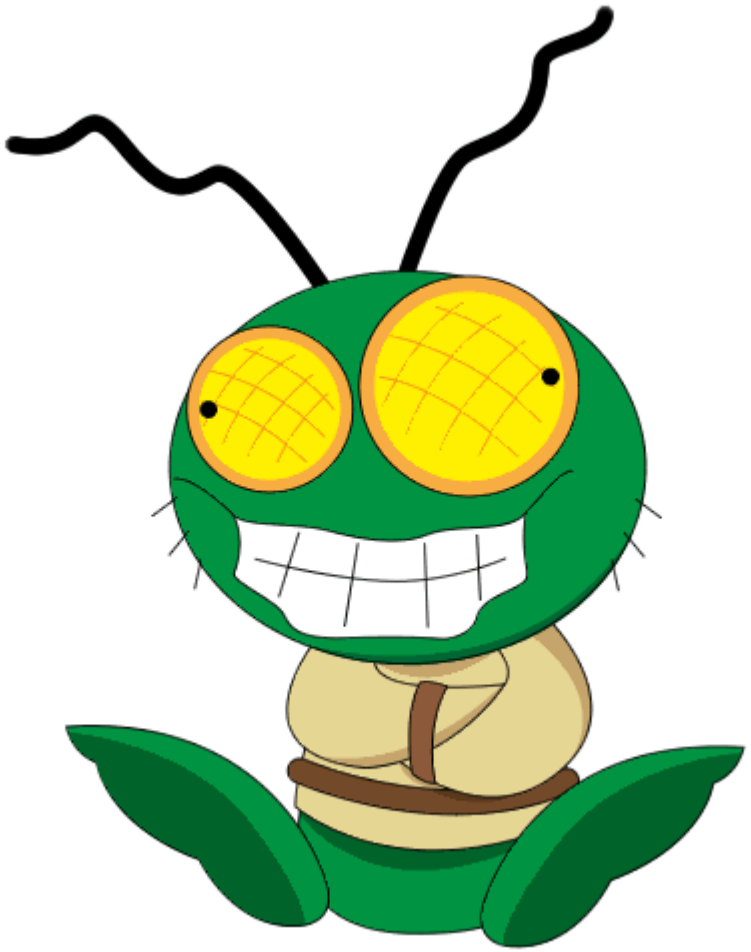
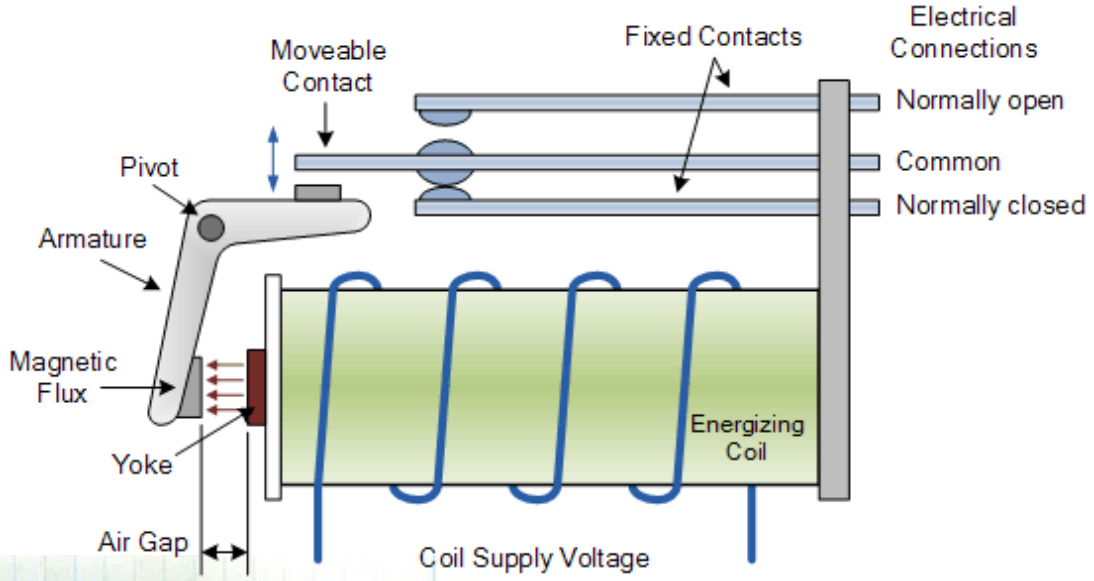
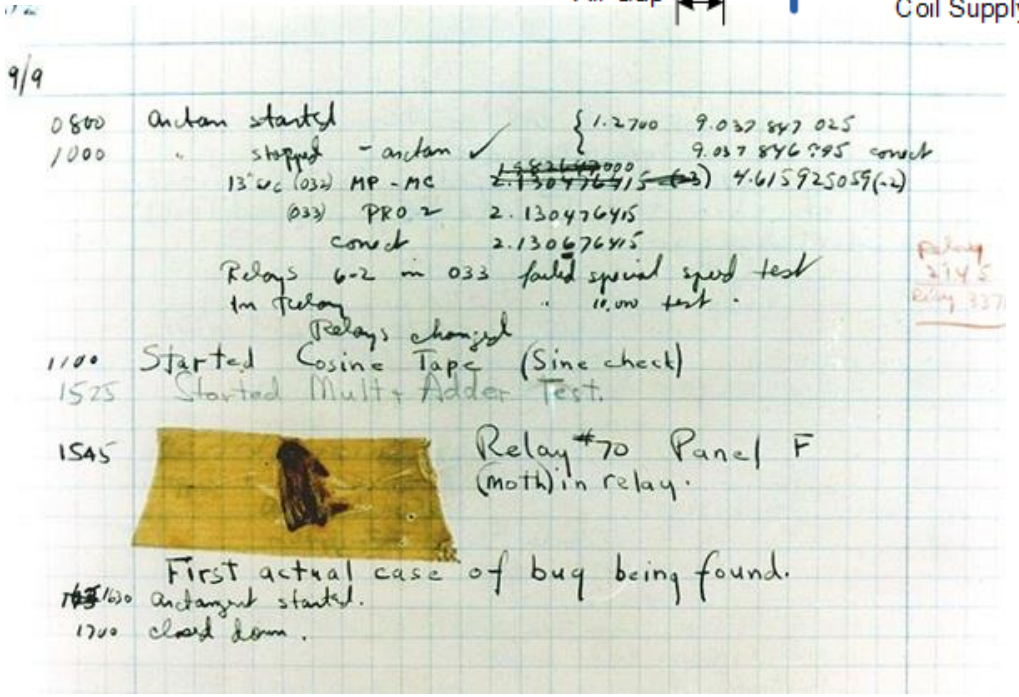
Debugging



First Computer Bug

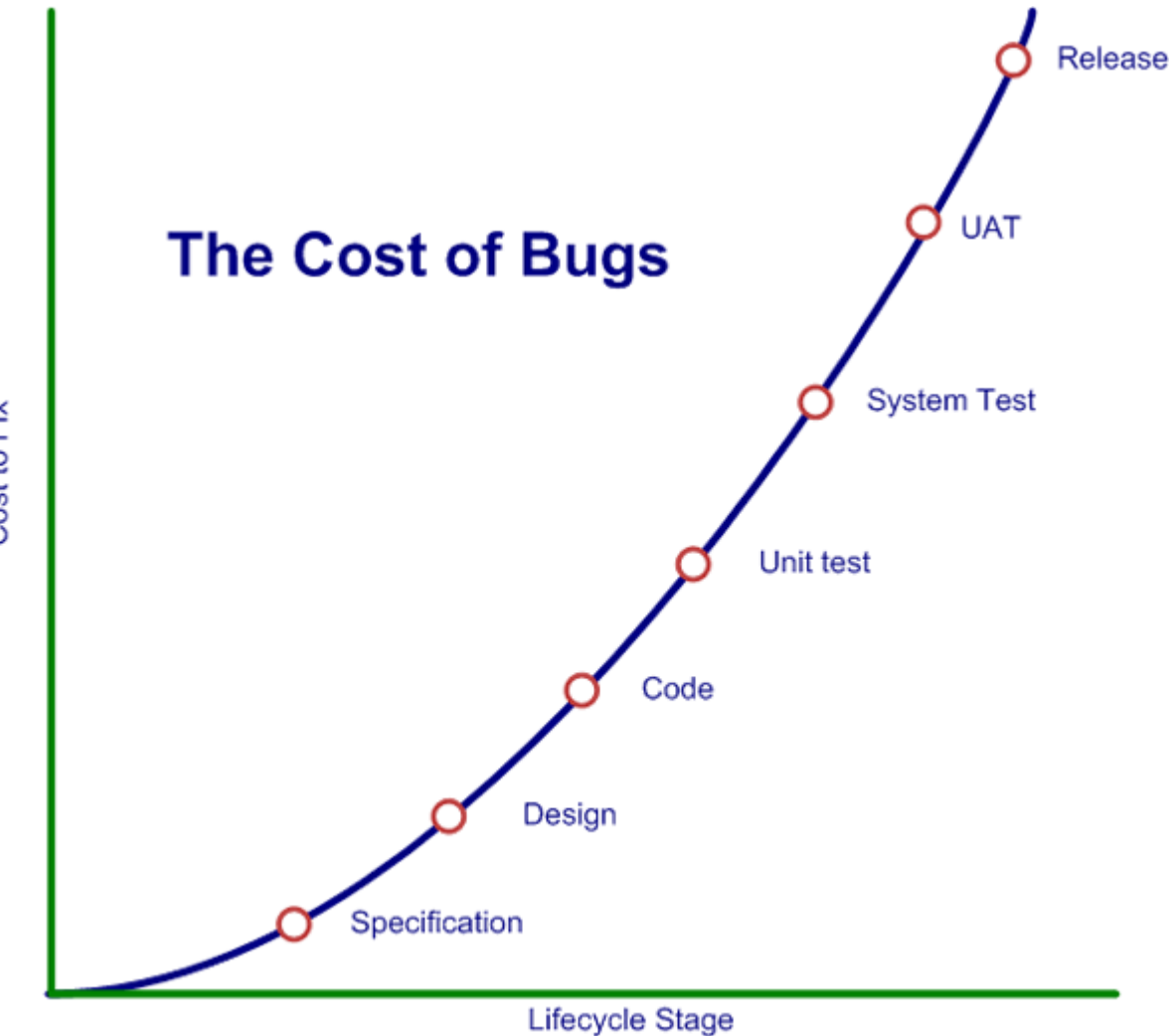


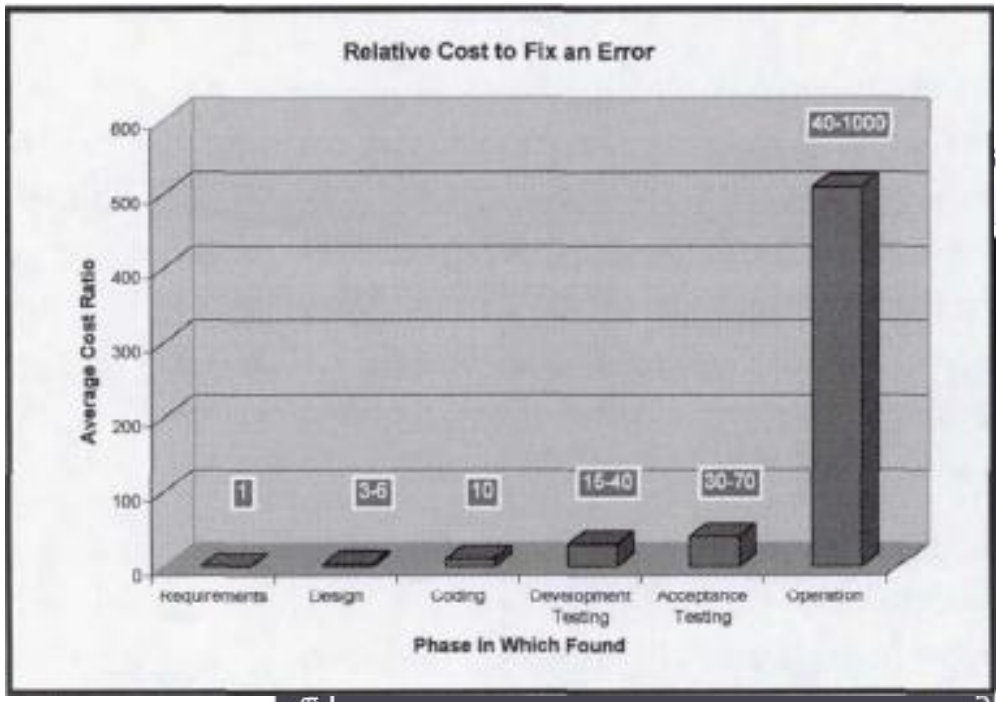


If you think you have no bugs…

"There's nothing wrong with the system; therefore, we'll need to perform exploratory brain surgery on you."

Find Bugs Early!





Preventing Bugs

1. Think through how things are going to work *before* writing code
2. Be meticulous (the computer is)
3. Use “assert” (next slide)
4. After writing code, think about what could go wrong
5. Fix compiler errors and warnings
6. Test your code for both expected *and* unexpected cases

Assert

- Method to check specific “assertions”
- An assertion is any logical expression, evaluated to true or false
- Expression passed as an argument to the “assert” function
  - If the expression is true, “assert” does nothing.
  - If the expression is false, “assert”…
    - writes an error message to stderr which contains
      - function name, assertion, and line number
    - aborts the C program (including a “core dump” if enabled)
- Assertion evaluation can be turned off once program is debugged
  - #define NDEBUG

Using Assert

```
#include <assert.h>
#include <stdio.h>

int main() {
        int x=atoi(getc());
        assert(x!=0);
        printf("Fract: %f\n", 10.0/x);
}
```

```
~>testAssert
3
Fract: 3.33333333
~>testAssert
0
assert: test.c:7: main: Assertion 'x!=0' failed.
Aborted
~>
```

Notes on Assert

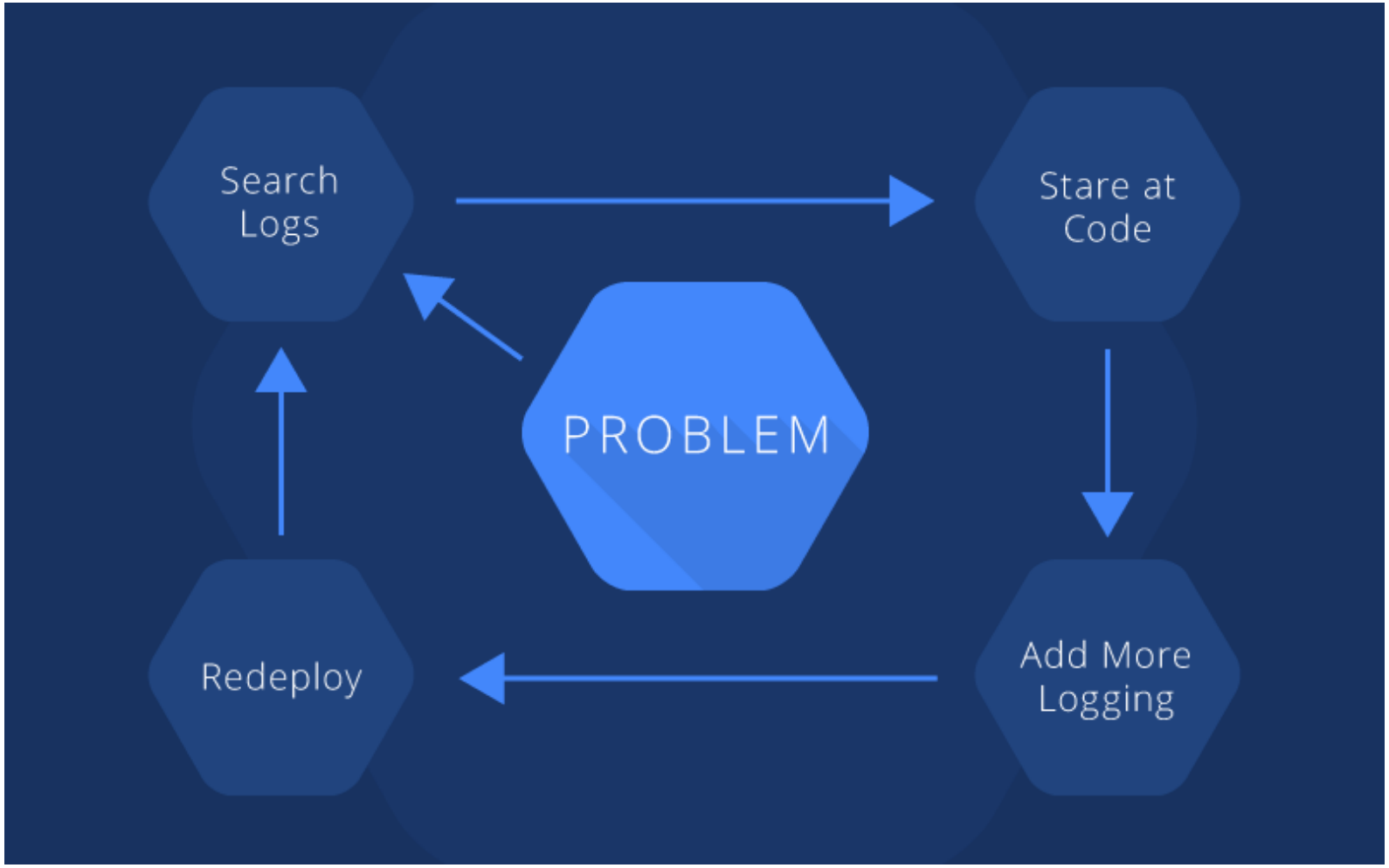
- Assert things you always expect to be true
  - When the assertion is false, it will cause other problems in your program
  - It's easier to understand what went wrong if you have an assertion
- Assert things you ASSUME to be true
  - It's very important to know when your assumption is wrong!
- Assertions are not very user friendly
  - If you are checking for a user error, write your own checker/message
  - Use assertions for UNEXPECTED problems
- Fix bugs which cause assertions to be false
  - Don't let another user fall into the same problem

Another assertion example

```
#inclue <assert.h>
#include <stdio.h>

int main() {
        int j;
        for(j=0;j!=100;j++) {
                assert(j<100);
                j=factorial(j);
        }
}
```

The “printf” debugger



printf debug pro's and con's

Advantages

- Don't need any special tools
- Works anywhere you can compile
- Use full power of C
  - if (xyz) printf("debug…");
  - …

Disadvantages

- Requires many trips around the edit/compile/test/evaluate loop
- Need to remove debug before delivering to customer

printf debug suggestions

- Start debug in column 1 so it looks different from real code
- Don't remove debug (you might need it again later)
  - Instead, comment using line (//) comment delimiter
- Use a debug marker in debug messages
  - I like the prefix "DBG:", so my debug messages read:
    DBG: x=17, y=19, about to call testfn(17,19)
    DBG: x=17, y=20, about to call testfn(17,20)
    …
- Give a hint about where the debug message comes from.

Alternative: GDB

Next time…

Resources

- Programming in C, Chapter 17
- Wikipedia: assert.h (https://en.wikipedia.org/wiki/Assert.h)