Class 13: (Avoiding) Debugging

6.102 — Software Construction
Spring 2024

Warmup

Nanoquiz: yellkey.com/**method**

Exercise: yellkey.com/**strategy**

Go to `warmup.ts`. Run it if you want (`npm run warmup`) — what does it do?

Minimize every variable's scope
by changing it to one of the following:

- **local variable**
  - declared in as narrow scope as possible
- **parameter**
  - you can change any method's parameters or return value
- **instance variable** of a `Sudoku` object
  - `private` if possible

Hoare’s Revelation

Sir Tony Hoare (inventor of Quicksort), in his 1980 Turing Award lecture

On 11 October 1963, my suggestion … was to relax the ALGOL 60 rule of **compulsory declaration of variable names** and adopt some reasonable default convention such as that of FORTRAN.

I was astonished by the polite but firm rejection…. It was pointed out that the redundancy of ALGOL 60 was the **best protection against programming and coding errors** which could be extremely expensive to detect in a running program and even more expensive not to.

I was eventually persuaded of the need to design programming notations so as to **maximize the number of errors that cannot be made**, or if made, can be reliably detected at compile time.

Perhaps this would make the text of programs longer. Never mind! The way to shorten programs is to use functions, not to omit vital information.

Scope Minimization

yellkey.com/**strategy**

In `warmup.ts`, change every variable to one of:

- **local variable**
  - declared in as narrow scope as possible
- **parameter**
  - you can change any method's parameters or return value
- **instance variable** of a `Sudoku` object
  - `private` if possible

Where to declare `firstRow`?

- A, B, or C

How?

- `public`
- `private`
- `let`
- `const`
- `readonly`

```
class Sudoku {
  //A: firstRow: number;
  ...

  public isSolved():boolean {
    //B: firstRow: number;
    ...

    for each block {
      //C: firstRow: number;
      firstRow = block / blockSize;

      for each row {
        for each column {
          ... puzzle[firstRow+row] ...


// no other uses of firstRow
```

Where to declare `numbers`?

- A, B, or C

How?

- `public`
- `private`
- `let`
- `const`
- `readonly`

```
class Sudoku {
  //A: numbers: Set<number>;
  public isSolved():boolean {
    //B: numbers: Set<number>;
    for each row {
      //C: numbers: Set<number>;
      numbers = new Set();
      ... numbers.contains(), numbers.add() ...
    }
    for each column {
      //C: numbers: Set<number>;
      numbers = new Set();
      ... numbers.contains(), numbers.add() ...
    }
    for each block {
      ... just like row and column loops ...
    }

// no other uses of numbers
```

Where to declare `puzzleSize`?

- A, B, C, D
  (pick all that apply)

How?

- `public`
- `private`
- `let`
- `const`
- `readonly`
  (pick keyword if any of your declarations use it)

```
function main():void {
  //A: puzzleSize: number;
  puzzleSize = 4;
  sudoku = new Sudoku();
  ...
}

class Sudoku {
  //B: puzzleSize: number;
  ...
  public Sudoku(/* C: puzzleSize: number */) {
    //D: puzzleSize: number;
    for row from 0 to puzzleSize-1 {
        for column from 0 to puzzleSize-1 {
            ...
        }
    }
// no other uses of puzzleSize
```

yellkey.com/**strategy**

Let’s look at a user bug report

- in `solverTest.ts`, the first test (bug #1079) is currently marked “skip”
- change `it.skip` to `it.only` and run the tests

Write your answers to these questions in `bugs.txt`

1. Study the data

- What’s different about this test vs. the other tests?
- How many blanks does it start with?
- What is wrong with its result?

Let’s look at a user bug report

yellkey.com/**strategy**

- in `solverTest.ts`, the first test (bug #1079) is currently marked “skip”
- change `it.skip` to `it.only` and run the tests

Write your answers to these questions in `bugs.txt`

1. Study the data

- What’s different about this test vs. the other tests?
- How many blanks does it start with?
- What is wrong with its result?

2. Hypothesize

Let’s assess hypothesis priorities

- What modules (ADTs or functions) does `solver.solve()` depend on?
- How well-tested are they? Critique the `Sudoku` tests in particular

3. Experiment

General hypothesis first: the bug is in `solve()`

Use the debugger to see what's happening

- Set a breakpoint on the line in the test calling `solve()`
- *Run → Start Debugging*
- Step into `solve()`, and then step through and examine variables
    - Compare your mental prediction with what you actually see
    - Make notes about unexpected behavior in `bugs.txt`

3. Experiment

General hypothesis first: the bug is in `solve()`

Use the debugger to see what's happening

- Set a breakpoint on the line in the test calling `solve()`
- *Run → Start Debugging*
- Step into `solve()`, and then step through and examine variables
    - Compare your mental prediction with what you actually see
    - Make notes about unexpected behavior in `bugs.txt`

4. Repeat

What's a more specific hypothesis about the cause of bug 1079?

Try to fix it!

Minimizing a buggy test case

Minimizing a buggy test case

- Remove the `.only` on the “bug #1079” test
- Go to the last test (“covers all blanks”) and change `it.skip` to `it.only`
- Minimize the test case
- Find the bug

Hoare’s Apology

Sir Tony Hoare (inventor of Quicksort),
at a conference in 2009

I call it my billion-dollar mistake. It was **the invention of the null reference** in 1965.

At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement.

This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.