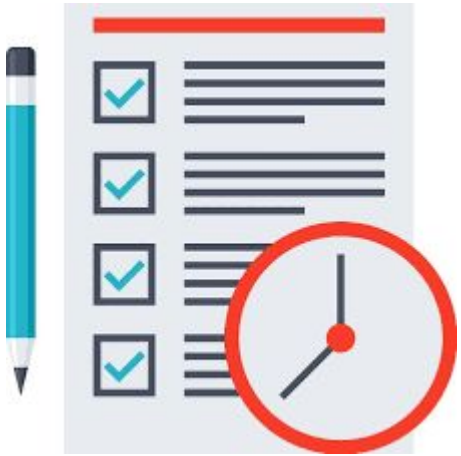
COMS W4156 Advanced Software Engineering (ASE)

October 20, 2022

Agenda

1. gtest/gmock demo
2. Finding Bad Code



gtest/gmock demo: Satyam Sharma

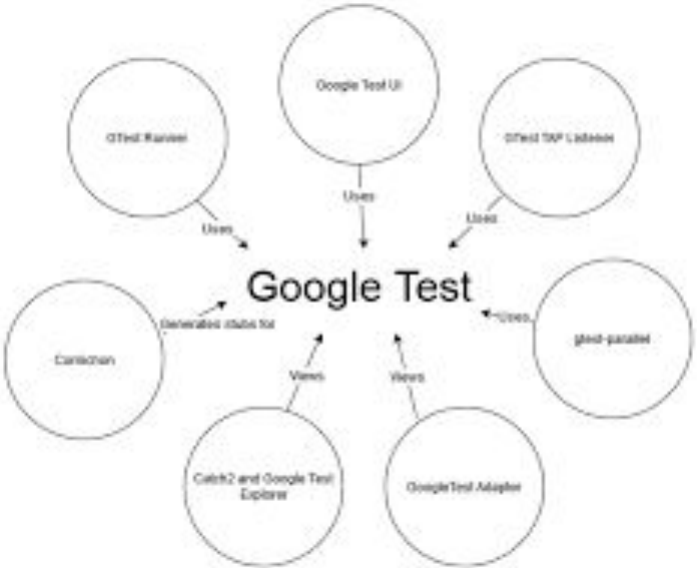
[demo repo]

Read the Docs

[GoogleTest - Google Testing and Mocking Framework]

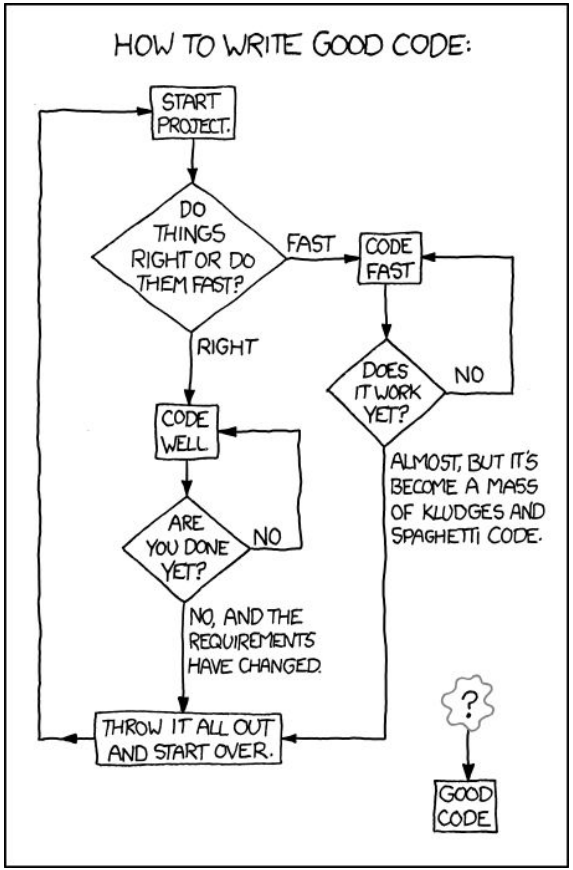
[GoogleTest User's Guide]

[Google Testing Blog]



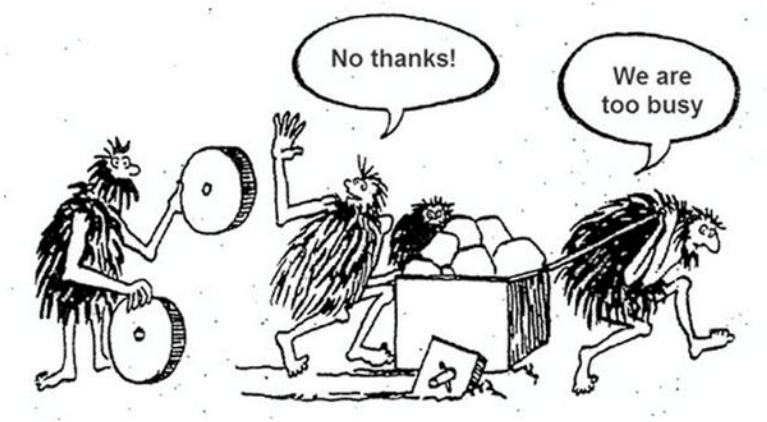
Agenda

1. gtest/gmock demo
2. Bad Code



“Technical Debt”

Cost of additional rework later on caused by choosing a fast and easy solution (bad code) now instead of a better solution (good code) that would take longer - not a big concern for a one-semester course project,
but indeed a big concern for industry software



Analogous to monetary debt - It’s ok to borrow against the future, as long as you understand that you need to pay it off

If technical debt is not repaid quickly, accumulates “interest” that adds up substantially over time - Justifies the time/effort spent on refactoring

Where does Technical Debt come from?

From developers …

A common example of incurring technical debt is when a code segment is copy/pasted from one location in the codebase to another

Copying existing code looks like a quick win—why write something new when it already exists?

"Code clones" = identical or nearly identical code that exists in more than one location in the codebase, typically arising via copy/paste

Prioritizing Technical Debt Repayment

Eventually, those two parts of the codebase might be maintained by different developers (or even different teams)

A developer finds a bug in one of those copies and another developer finds the same or different bug in another copy

Is it better for each developer to fix each copy independently?

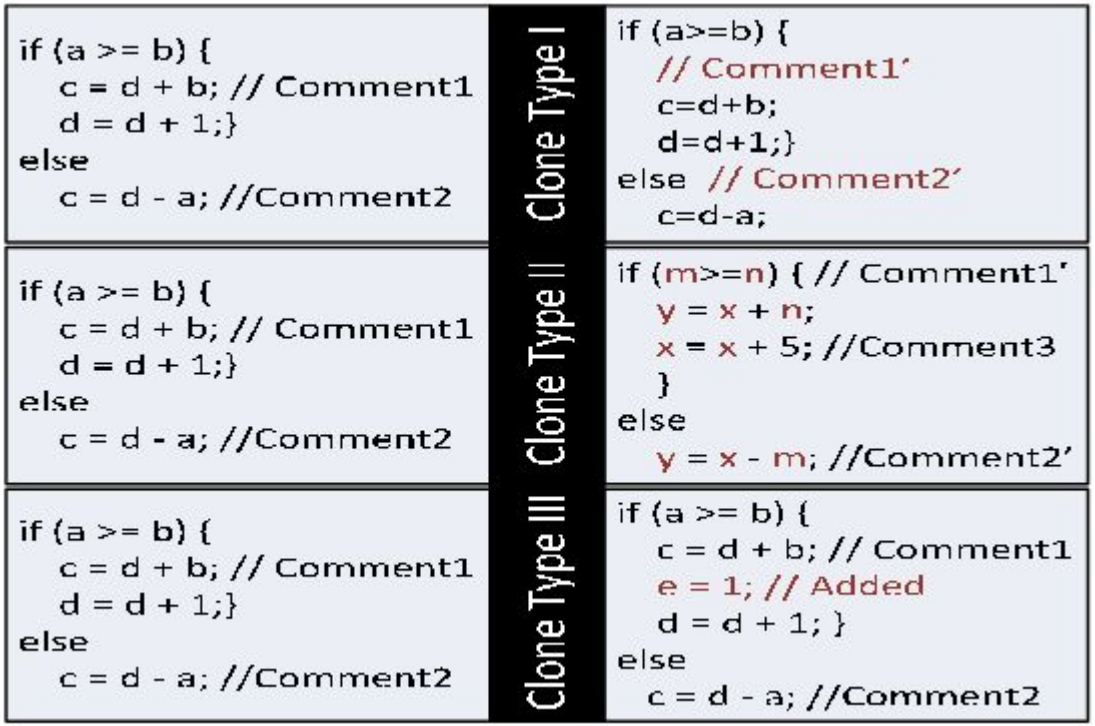
Or is it better to refactor so this piece of code appears in only one place and is called in the two original places, and then fix bugs?

Example

Code Clones Are Not Necessarily Identical

But how do the two developers know that there are other copies?

Even if it's the same developer working with both copies, the developer may not recognize that they are copies due to reformatting, identifier renaming, other divergences that accumulate over time



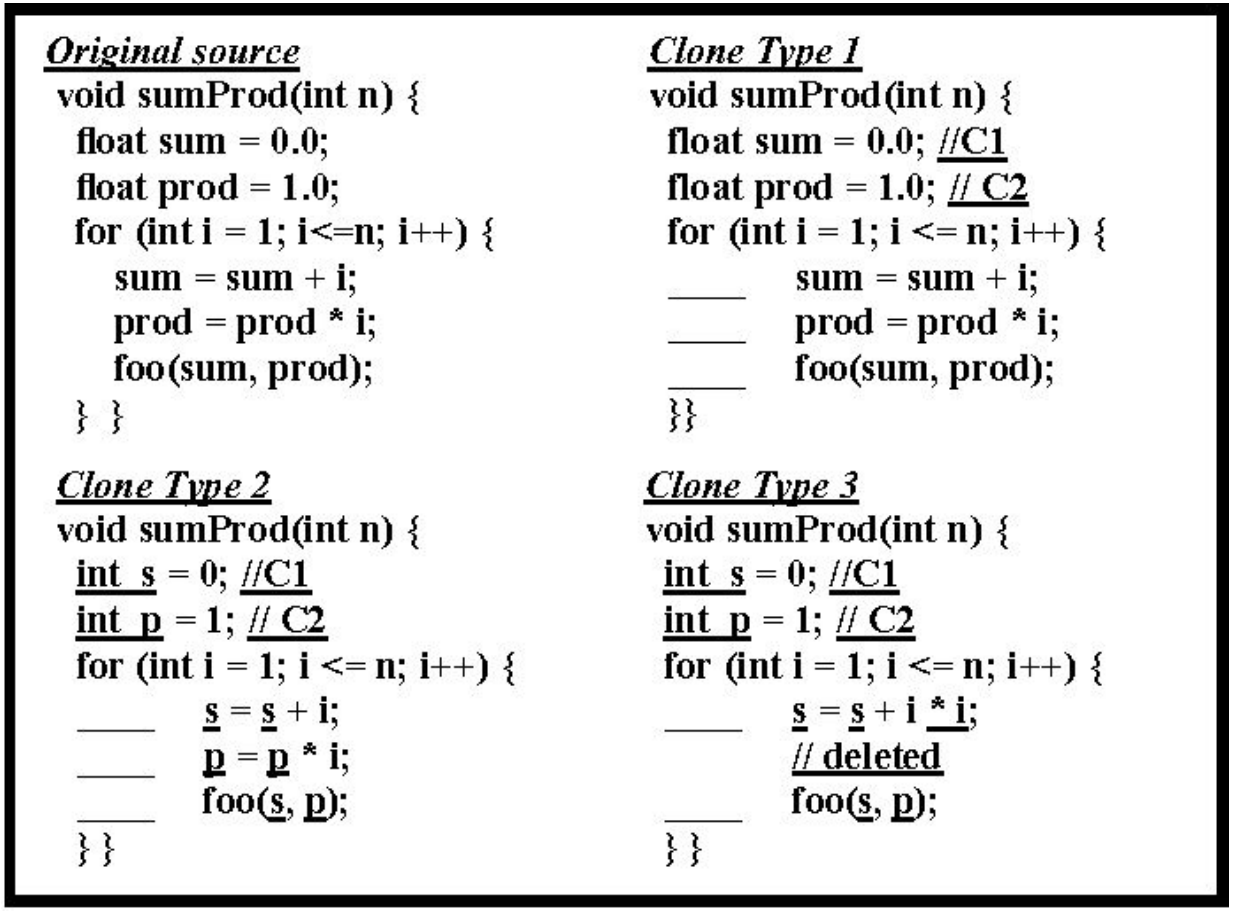


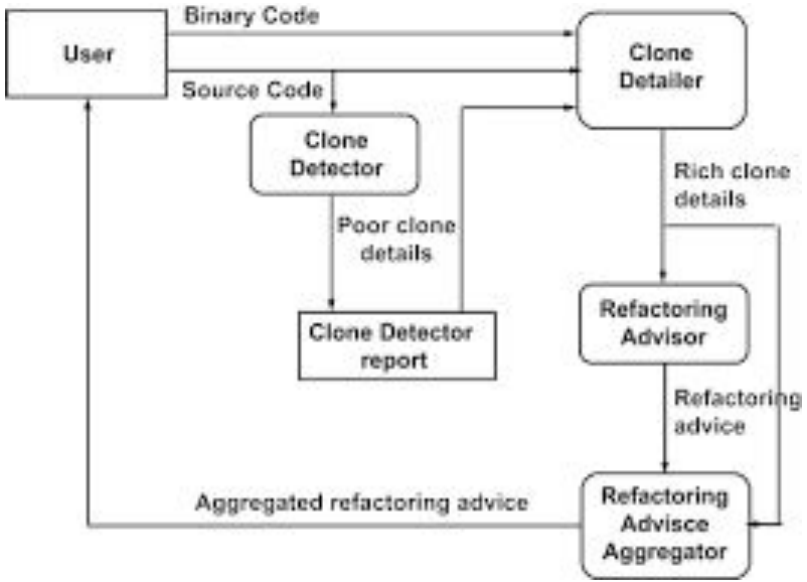
Fig. 1. Examples for three clone types.

Code Clone Detectors

Automated code clone detectors can help detect these locations

Some code clone detectors are language-specific, some work across languages, e.g., CPD

Some refactoring tools, e.g., JDeodorant, not only automatically detect a group of code clones but also help developers replace each instance of the clone with a call to a single code unit



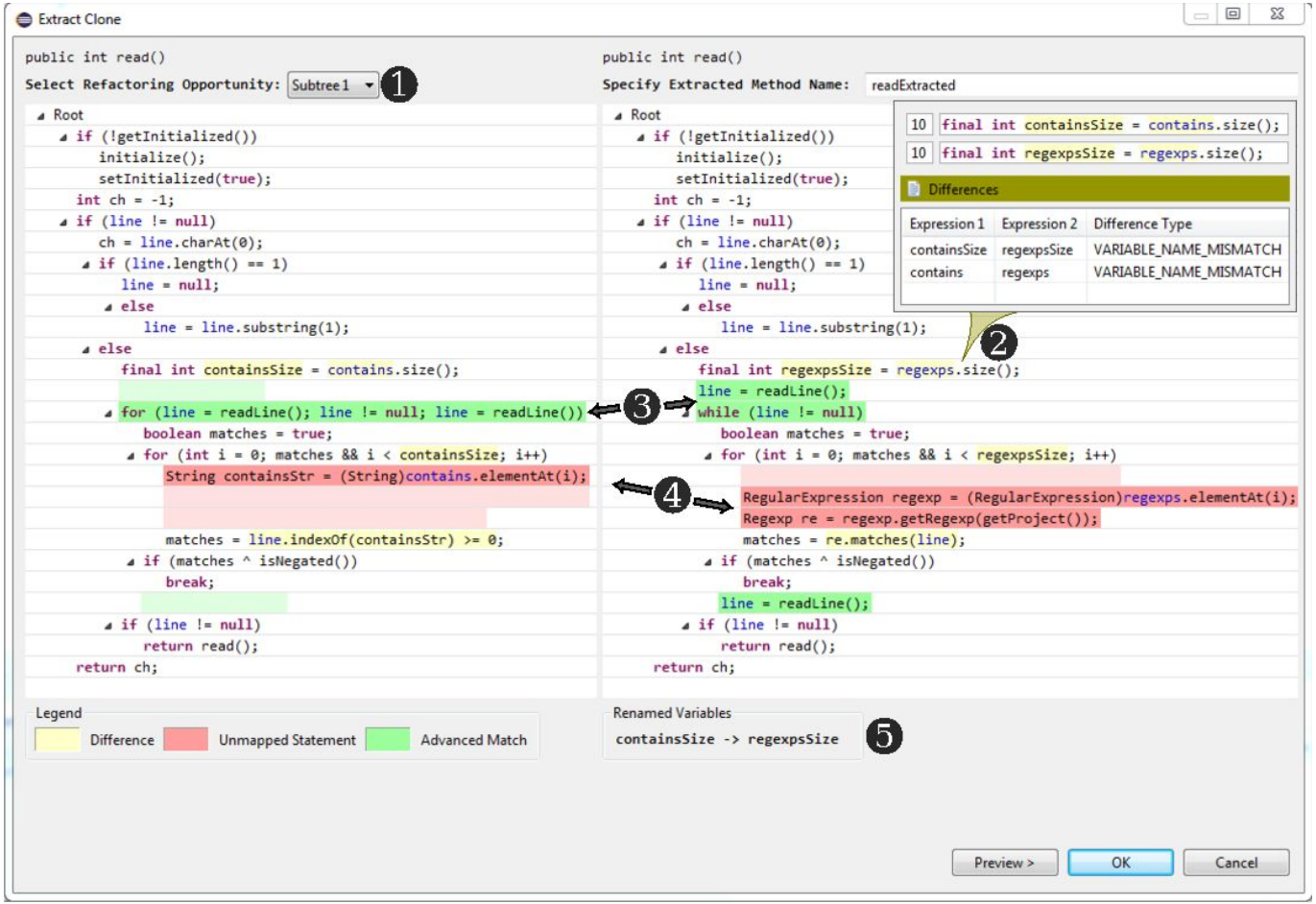


Figure 2: Clone pair visualization and refactorability analysis.

Code Clone Type 4

Type 4 clones are another kind of duplicated code in the same codebase - usually written independently and do not look similar, but have the same or similar *functionality*

Sometimes called semantic or behavioral clones to distinguish from the syntactic or structural clones (types 1-3), aka “simions”

```
int x, y, z;
z = 0;
while (x>0) {
  z += y;
  x -= 1;
}
while (x<0) {
  z = -= y;
  x += 1;
}
```

```
int x, y, z;
z = x*y;
```

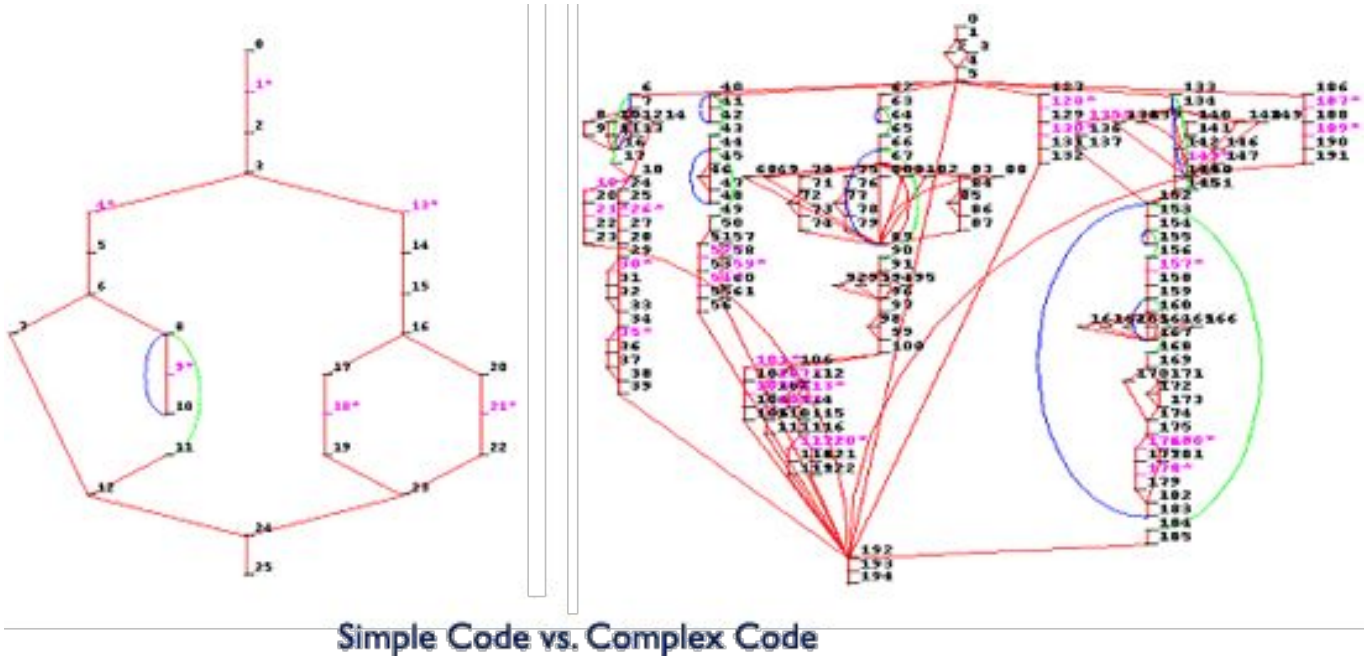
Detecting Semantic Clones

Semantic clones may be less likely to contain the same bugs as syntactic/structural clones, but the extra code still adds to maintenance costs

```
CHECK-HALT(program)
{
    if (program halts)
        infinite loop
    else
        halt
}
```

Impossible to detect automatically in the general case (aka “undecidable”), but can identify in many practical cases (similar I/O, execution trace, abstract memory state, metrics)

Code Complexity is another source of Technical Debt



Simple Code vs. Complex Code

Code Complexity

Cyclomatic complexity = number of conditions + 1

Minimum number of test cases required to achieve branch coverage

Higher cyclomatic complexity usually, but not always, makes code harder to understand and maintain

Could be considered a code smell, as a “bloater method” with too many paths

Not always?

This code has cyclomatic complexity 14, far beyond the typical threshold - but no developer would consider this code complex

```
String getMonthName (int month) {
    switch (month) {
        case 0: return "January";
        case 1: return "February";
        case 2: return "March";
        case 3: return "April";
        case 4: return "May";
        case 5: return "June";
        case 6: return "July";
        case 7: return "August";
        case 8: return "September";
        case 9: return "October";
        case 10: return "November";
        case 11: return "December";
        default: throw new
IllegalArgumentException();
    }
}
```

Both of these examples have cyclomatic complexity 5

Also need to consider *nesting depth* = maximal number of control structures (loops, if) that are nested inside each other, which probably correlates more closely with understanding and maintenance challenges

```
String getWeight(int i) {
        if (i <= 0) {
            return "no weight";
        }
        if (i < 10) {
            return "light";
        }
        if (i < 20) {
            return "medium";
        }
        if (i < 30) {
            return "heavy";
        }
        return "very heavy";
}
```

```
int sumOfNonPrimes (int limit) {
    int sum = 0;
    OUTER: for (int i = 0; i < limit; ++i) {
                if (i <= 2) {
                        continue;
                }
                for (int j = 2; j < i; ++j)  {
                        if (i % j == 0) {
                                continue OUTER;
                        }
                }
                sum += i;
        }
        return sum;
}
```

Other Technical Debt

Dispensable unused code - sometimes previously used (dead code), sometimes created “just in case” for future (speculative generality)

Don’t just comment out dead code, delete it - if it turns out you need it again, it’s still in the version control repository

Reducing code size (even in comments) means less code to understand and maintain

See Knightmare: A DevOps Cautionary Tale

Upcoming Assignments

First Iteration due October 24, next Monday!

First Iteration Demo due October 31

➔ You can keep coding and testing between first iteration submission and first iteration demo, but tag both separately! Also see github releases

First Individual Assessment
available 12:01am November 1, due 11:59pm November 4

Next Week

(In)Secure Coding

Design Principles

Bug Finder demo

Refactoring

Ask Me Anything