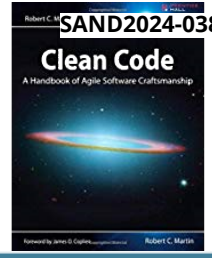
Sandia National Laboratories

SAND2024-03814C



Developing Coding Standards across the HPC Domain

Sustainable Scientific Software Conference (S^3C)

Manoj K Bhardwaj, Sandia National Labs

April 8-11, 2024

Seattle, WA

U.S. DEPARTMENT OF ENERGY | NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Acknowledgements

Scott Warnock, Stuart Baxley, Chris Sullivan, Gary Lawson, Akhil Potla, Richard Drake, Henry Gabaldon

Erik Strack, Terri Galpin, Tricia Gharagazloo, Dena Vigil, Richard Michael Jack Kramer, Salomé Thorson, Raisa Koshkin

Sierra Toolkit (STK) Team:

Alan Williams, Jesse Thomas, Todd Coffey, Nate Roehrig, C. Riley Wilson,

Tolu Okusanya, David Glaze, Johnathan Vo

These ideas lead to ~10% debt on STK team.

Developing coding standards, practices, guidelines, etc.

Using the word “standards” to mean more than that.

Want S^{3}C to be a place where developers can share/collaborate. DevOps is just Ops without Dev.

Why this is important.

Most are **not** taught how to write “sustainable” software

Many teams have technical **debt that exceeds 50%** of their budget

My hypothesis: if development teams start to get curious and apply these practices, they will see significant cost savings in two ways (and become sustainable):

- Debt can be significantly reduced
- Adding features will be much faster or kept at similar cost through time

>$100M impact annually?

Welcome to Fake Science and Engineering Company (FSEC)

CEO – me

New employees – you

Expectations on working here…

Orientation Agenda for FSEC

1. Your #1 job as software development professional
2. Test-Driven Development (TDD)
3. Scrum
4. Pair programming
5. Six line functions
6. Minimize # of function parameters
7. No comments in the code
8. Legacy code – add unit tests as we touch code
9. Measuring effectiveness
10. Scientific software thoughts

#1 job of software development professional

Martin Fowler

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

More important?

Making software readable is more important than making it work

Reading to Writing ratio: 10-1

All new code is developed using Test-Driven Development (TDD)

Test-Driven Development

When using TDD – Red, Green, Refactor

Refactor! Let classes/design come out of the refactor. This includes the test code!

Anticipating versus emerging designs

Four Rules from Martin Fowler for Simple Design (in priority order)

- Passes the tests
- Reveals intention
- No duplication
- Fewest elements

Training on S.O.L.I.D. principles to guide your refactoring

Benefits: Fast running verification unit tests, line coverage, fast feedback!

Scrum

Scrum

You will be on a Scrum team

Jeff Sutherland: Scrum by itself isn't the goal

Make time for daily refactorization

T (Total capacity) = T1 (time on new dev) + T2 (**time refactoring**) + T3 (time maintaining/fixing code)

Debt: if T2 is low, T3 will increase, and T1 will go to zero

One goal per Scrum team! Will NOT be multi-tasking

100% on team (no part-timing)

Pair programming

Pair programming

If it needs to be maintained, it will be pair programmed

No, it's not paying 2 people to do the same job

- 15% increase in cost initially, less cost long term!

A pattern that works:

- Navigator doesn't touch the keyboard (knows what needs to be written next)
- Driver executes the wishes of the navigator

IDE – using the same one

Six line functions

Six line functions

Don’t count curly braces!

Yes, maximum of 6

You can exceed 6 on a case-by-case basis with approval of me, your CEO

Minimize # of function parameters

Minimize # of function parameters

calculateArea(square, len, side, isRhombus, density)

calculateArea(square)

area = square.getArea()

0 parameters great

1 parameter good

2 parameter ok

3 parameters – meh

more than 3, need some refactoring

No comments in the code

// this slide introduces the “no comments” portion of the presentation

No comments in the code

Work towards making code more readable so comments aren't necessary

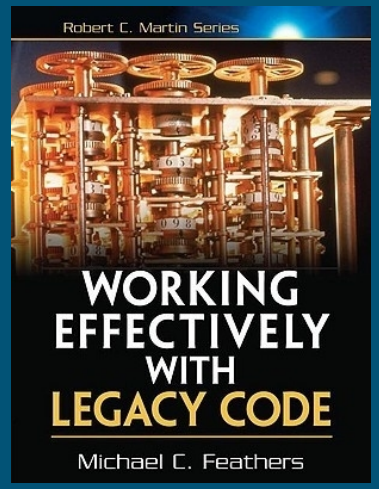
```
double a = 12.0 // a is Area
```

```
// the following function finds the local minima
int tryThis(const std::vector<int>& input) {   }
```

Legacy code – add unit tests as we touch code

Improving Legacy Code

Training will include lessons from this book →



Legacy code is code without automated verification unit tests

Licensed scientific software from national laboratory

Lots of debt

Will refactor as much as we can before we consider rewriting (using our practices)

Will use Kent Beck's Four Rules for Simple Design

Development and Operations (DevOps)

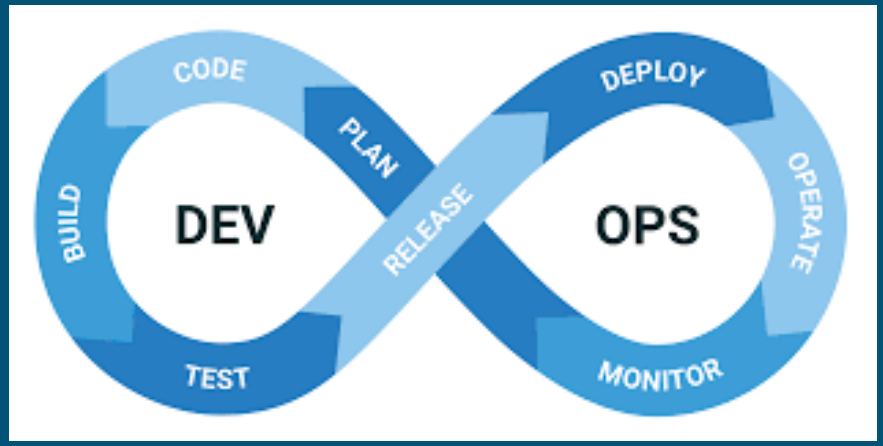
DevOps

You are the developer in DevOps

We are partnering with DevOps engineers

- Rapid feedback cycles
- Legacy code: Continuous integration times

Our resources are NOT unlimited



Measuring effectiveness

Measuring effectiveness

T (Total capacity) = T1 (time on new dev) + T2 (**time refactoring**) + T3 (time maintaining/fixing code)

Debt measure: T1/T

Continuous integration times < 1 minute (or so)

Line coverage > 99%

Always experimenting

Measuring effectiveness

T (Total capacity) = T1 (time on new dev) + T2 (**time refactoring**) + T3 (time maintaining/fixing code)

Debt measure: T1/T

Continuous integration times < 1 minute (or so)

Line coverage > 99%

Always experimenting

Scientific Software Thoughts

Scientific Software Thoughts

We need the expertise you are bringing to the team (various engineering fields)

We separate the idea of finding a solution to a problem **versus** how to implement the solution in code

Fungible in domain expertise versus Fungibility in code implementation

We choose team implementations over individual implementations

Closing of FSEC Orientation

1. Your #1 job as software development professional
2. Test-Driven Development (TDD)
3. Scrum
4. Pair programming
5. Six line functions
6. Minimize # of function parameters
7. No comments in the code
8. Legacy code – add unit tests as we touch code
9. Measuring effectiveness
10. Scientific software thoughts

Path forward ideas to consider

Path forward ideas to consider

Organizations should define minimum quality criteria (Definition of Done)

Would love to work with teams who want to try and adopt some of these ideas.

Maybe start “studying” these ideas and presenting results at future S^{3}Cs

Start developing tools for developers that help enforce good habits

Research ideas

- Use AI/ML to turn system tests to smaller tests? Invert the testing pyramid.
- Use ChatGPT to pair program
- Use AI/ML in CI pipeline to find “unclean” code and “fail”

Here to help! Creating change on existing teams is hard.