# How to Create a Mediocre Open-Source Repository

Jae-Won Chung

# About this talk

***Caution: Extremely opinionated talk***

- More like the reasons of whatever I'm doing with Zeus
- Slightly tailored to open-sourcing academic projects

***What does getting a PhD mean?***

- Getting the certificate
- Securing a nice big tech job
- Publishing papers
- Becoming a world-leading expert in a very specific topic
- Generating real world impact with cool & useful research

# A brief history of ZEUS

| | |
|---|---|
| *Aug 2022* | **First open-sourced** with logo and docs |
| *Nov 2022* | **CarbonHack** hackathon 2nd place (25k USD) |
| *Oct 2023* | **2023 PyTorch Conference** presentation |
| *Dec 2023* | **Mozilla Technology Fund** (50k USD) |
| *Jan 2024* | **Salesforce** gift (20k USD) |

# "It works. What more do you want?"

*What more can you possibly want?*

- "Working code" – "Good code" continuum
- Depends on where the code is for

*Example scenarios*

- Open sourcing research artifacts – More towards good code
  - At this point, its purpose is to be *read & modified*
- One-time experiment – More towards working code
  - But never hurts to have good code, because it reduces bugs

# “It works. What more do you want?”

***What is good code?***

- Code that is easy to change
- Industry perspective: *Change* is the only constant
- Academica perspective: Easy to build on top of your work

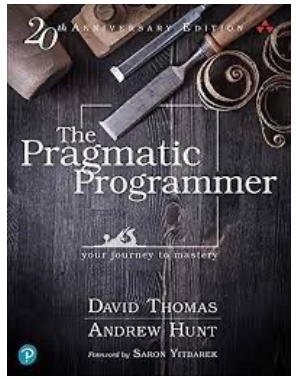
***What kind of code is difficult to change?***

- Coupled code
- Complex code

# "It works. What more do you want?"

***Learn the language thoroughly***

- Read the language documentation
- Solve Advent of Code and compare solutions
- Do medium-sized side projects with it




***Read books and watch YouTube***

- "The Pragmatic Programmer" – Classic but still relevant
- ThePrimeagen (Programming in general)
- JonGjengset (Rust)
- mCoding (Python)

# “It works. What more do you want?”

*You won’t have time to refactor later*

- The only way to end up with good code is to always write good code
- Hacking your way to the next experiment result produces hard-to-change and error-prone code

*Multiple objectives*

- Performance – For the numbers in my paper
- Readability & Flexibility – For the citations

# “People don’t read READMEs anyway”

*The Linux kernel README…*

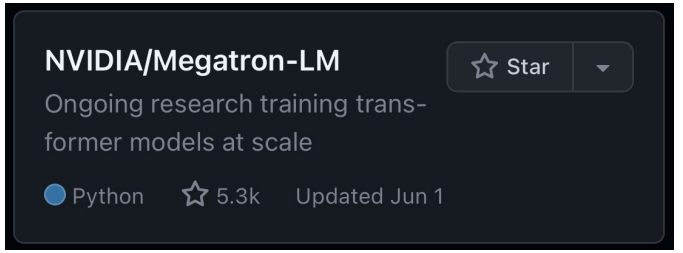
- Well, my project isn’t Linux

**README**

```
Linux kernel
============

There are several guides for kernel developers and users. These guides can
be rendered in a number of formats, like HTML and PDF. Please read
Documentation/admin-guide/README.rst first.

In order to build the documentation, use ``make htmldocs`` or
``make pdfdocs``.  The formatted documentation can also be read online at:

    https://www.kernel.org/doc/html/latest/

There are various text files in the Documentation/ subdirectory,
several of them using the Restructured Text markup notation.

Please read the Documentation/process/changes.rst file, as it contains the
requirements for building and running the kernel, and information about
the problems which may result by upgrading your kernel.
```

*People don’t care about my project*

- Very few people will read beyond the first sentence
- It’s up to me to spoon-feed people
- Carefully craft a 7-8-word description



# “People don’t read READMEs anyway”


Small efforts can make a difference

# “Don’t mess with my coding style”

***Linting***

- Static analysis for finding obvious bugs and code smells
- Potentially a great time saver!
- *Ruff for Python, clippy for Rust, clang-tidy for C++*

***Formatting***

- More about consistency than aesthetics
- Removes conflicts from style differences during collaboration
- *Black for Python, rustfmt for Rust, clang-format for C++*

# “Testing just shows lack of confidence”

*Obvious advantages*

- Catch bugs before your users do
- Industry people usually don’t trust anything without tests
- Tests are the first user of your APIs – Evaluate ergonomics

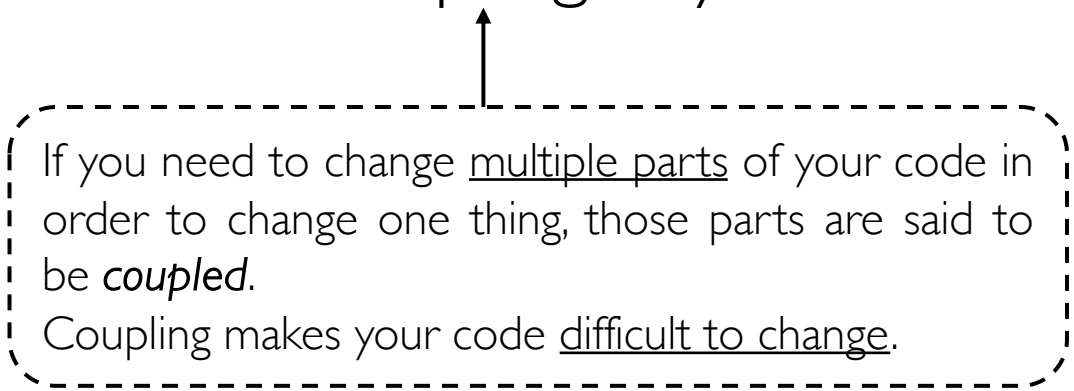
*Refactor with confidence*

- In software development, **the only constant is change**
- Without tests, you can’t be sure whether a refactor altered functionality/correctness

# “Testing just shows lack of confidence”

*Design for testability*

- Clear and isolated component boundaries
- Observable components
- Ultimately reduces coupling in your codebase



# “Good code documents itself”

*Auto-generate documentation from code*

- Making people read raw source code is torture
- Modern languages even define special doc comment syntax
- Mkdocs for Python, cargo-doc for Rust

```
1    //! This crate is a Rust port of Google's high-performance [SwissTable] hash
2    //! map, adapted to make it a drop-in replacement for Rust's standard `HashMap`
3    //! and `HashSet` types.
```

*Essential sections*

- Feature Overview – What does the user get by using this?
- Getting Started – How to get from zero to up & running

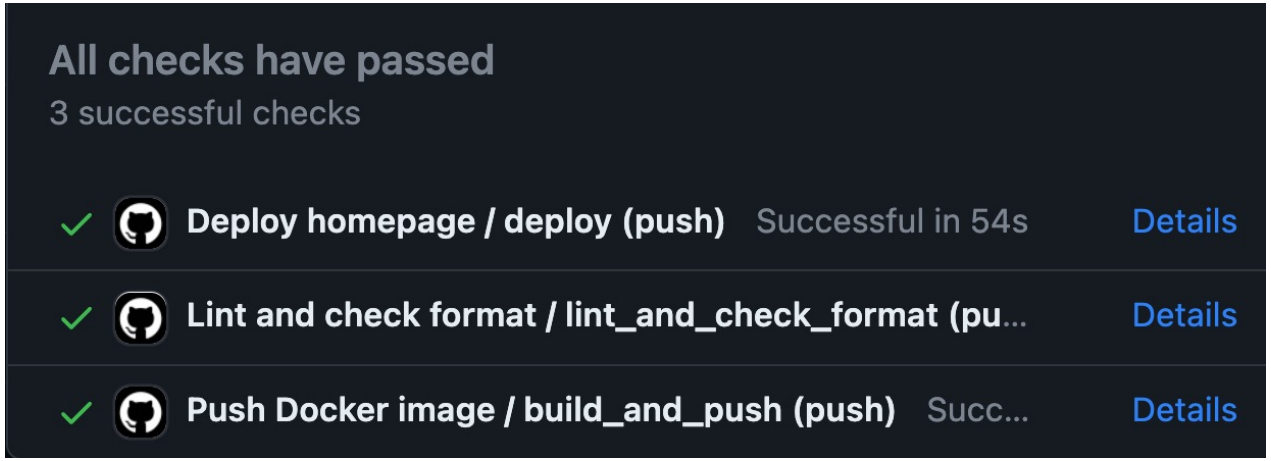
# “I promise I won’t make mistakes”

*Continuous Integration*

- Running pre-defined commands on events (e.g., push, PR)
- Automate format checking, linting, and testing

*Continuous Deployment*

- Useful for deploying project homepages and documentation



# “RTFM.”

***Some (if not most) people like diving into example code directly***

- Docs/Manuals can be too large to quickly digest
- Examples are basically top-down explanations whereas docs are bottom-up explanations

***One example may not be enough***

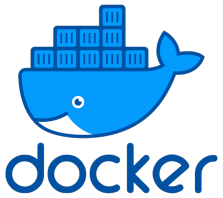
- One for the simplest use case
- A couple more if more complex/advanced API usages exist

# "They should experience at least half our pain"

***First impressions matter***

- Many open-source software don't even run
- If it doesn't run immediately, users will think it's one of them
- People want to get things done without doing anything

***Provide Docker images through Docker Hub***




- Many modern open-source software provide Docker images
- Package everything (including example scripts) in the image
- Just do your research inside containers from the beginning

# “Landing features trump everything”

***Enforcing high code quality slows down feature implementation***

- Degree of slowdown depends on the complexity of the project and the developers’ skill level
- There’s a point where slowdown is no longer slowdown

***Is the slowdown worth?***

- How long will you work with the codebase?
- How much slowdown will there be?
- Do you want people to use and build on what you did?

# Jae-Won's Final Recommendation

- Make it a habit to always write good code
- Just spend five minutes before typing to think about design
- Read the manual end to end
- Take an investment mindset
- Creating a good open-source repo goes a long way

*"It solves a problem people care about." – Ion Stoica*