Rule of Software "Engineering"

How to not go crazy developing software in the real world.

Benjamin Shropshire
benjamin@precisionsoftware.us

September 29, 2023

git:7d712680d949a4b603ff57851e0b3c19753d7069

Background

- Institutional.
- Stable foundation.
- Ideals (with real-world compromises).
- Having a target gives a direction to move towards.

Foreground: So what's the point?

- **Sustainability**: IOUs.
- **Maintainability**: Escape its legacy and choices.

- Interrupts: e.g.
  - Upstream library updates.
  - "Requests" from the PHB.
- Opportunities: e.g.
  - New requirements & feature requests.
  - Exploit newly acquired insight, experience and capabilities.
- Unexpecteds: e.g.
  - Late breaking "issues"[1].

[1] e.g. Meltdown, Spectre, Heartbleed, log4shell

Introduction

- Rules for the *build process*.
- Rules for the *code review process*.
- Rules for *testing*.
- Rules for the *release process*.

Rules for the build process:

Rule #0: Have a build process.

What is *not* a build process?

- A source code directory.
- A `README.txt` file.
- A shell script.

What is a build process?

- A stable, consistent, portable, supported interface.
- A toehold into new and unfamiliar code.

Rules for the build process:

Rule #1: Use the same build process for "everything".

Okay; wherever it makes sense.

Benefits of using a single build process uniformly:

- Starting on code is always the same.
- Portability/reuse.
- Reduce the number of things being supported.

Rules for the build process:

Rule #2: Use a scalable build process.

It needs both the *capability* and *capacity* to get the job done.

- Reliable.
- Capable & flexible.
- Fast.
- Provision the capacity it needs.

The choice of build tool is critical.[2]

"Build everything and run all the test to see what happens" should be an easy reflexive part of development.

[2] My personal, heavily biased, recommendation is Bazel.

Rules for the build process:

Rule #3: Use a hermetic build process.

Track everything the build uses.

- Project source code.
- Dependent libraries (compiled or source code).
- Configuration(s).
- Tool chain(s) (configurations and binaries).
- *EVERYTHING!!!*[3]

- Everyone gets the same results, failures, successes.
- Handoffs just work.
- Dev environment is trivial.

[3] With apologies to Luc Besson.

Rules:

A quick aside ...

Rules:

Rule #0: Use source control.

Rules for the build process:

Rule #4: Use a reproducible build process.

No changes when re-building from history.[4]

Without this: Finding breaking changes is expensive.

With this: No uncontrolled changes.

[4] Excluding metadata, build stamps, etc.

Rules for the build process:

Rule #5: Commit to keeping the build clean.

This is not free.

- Buy-in from the people who use it.
- Prioritize it. (Who sets priorities?)
- Requires ownership.
- Maintenance before development.

Rules for the build process:

Rule #0 Have a build process.

Rule #1 Use the same build process for “everything”.

Rule #2 Use a scalable build process.

Rule #3 Use a hermetic build process.

Rule #4 Use a reproducible build process.

Rule #5 Commit to keeping the build clean.

Rules for the code review process:

Rule #0: Have a code review process.

What is *not* a code review process?

- "*Yo, I just submitted some code if anyone want to look at it.*"
- The design process.[5]
- Just for important/complex code.
- Elitism, rubber stamping, hazing.

What is a code review process?

- A second perspective.
- Distributing expertise.
- Maintaining awareness.
- A demonstration of the importance placed on code quality.

[5] Design is important, but it's different than review.

Rules for the code review process:

Rule #1: Maintain a style guide.

A good plan violently executed right now
is far better than a perfect plan executed next week.

— George S. Patton

Benefits of a style guide:

- Avoids debates about trivia.
- Acts as an authority.
- Starting point for improvements.
- Distribution hub of policy changes.
- A knowledge base.

Don't be too dogmatic.

Rules for the code review process:

Rule #2: Maintain defined owners for all code.

- Unowned code: "*Someone else's problem.*"
- Too many owners: "*Indecision by committee.*"
- The owner is responsible for understanding that code.
- The owner is responsible for curating that code.
- Non-owners *can* touch code.

Any reasonable sized code base will be too large for any one person to know in detail.

Rules for the code review process:

Rule #3: Maintain automatic checks.

Don't ask developers or reviewers to do task than can be done by computers.

- Auto-formatters.
- Lint.
- Static analysis.[6]
- Presubmit checks.
- ...

Note: The automated check must have very few false positives so that developers do not grow accustomed to ignoring everything.

[6] E.g. Clang-Tidy.

Rules for the code review process:

Rule #4: Maintain those checks at head.

- Run automatic check at HEAD.
- Fixing preexisting problems.

If it's worth testing for, it's worth fixing.
Spending effort fixing things shows people it's important.

Rules for the code review process:

Rule #5: Commit to doing code reviews.

This is not free.

- Buy-in from reviewers and reviewees.
- Make reviewing code a priority.
- Make good code a priority.
- Code owners must schedule time for this.
- Code review is part of the job.

Rules for the code review process:

Rule #0 Have a code review process.

Rule #1 Maintain a style guide.

Rule #2 Maintain defined owners for all code.

Rule #3 Maintain automatic checks.

Rule #4 Maintain those checks at head.

Rule #5 Commit to doing code reviews.

Rules for testing:

Rule #0: Have tests.

What is *not* testing?

- "Build it and try a few things."
- A list of things to try.
- A testing department.[7]

What is testing?

- Available to every developer.
- A formal definition of passing.
- Part of the build process (preferably).

Improvements are $O(n)$ not $O(n^2)$.
Try it and see what breaks!

[7] Not that having one is a bad thing, that just something else.

Rules for testing:

Rule #1: Make it easy to find and run the tests.

Make finding them the same everywhere.

- Easy to run tests get run more regularly.
- Testing is part of the development process.
- A definitive answer to “what are the test?”.

Rules for testing:

Rule #2: Make the tests run quickly.

Short edit/test/debug cycles accelerate productivity.

- "*Did an edit fix the one thing or break another?*"
- Quickly and definitively answers changes processes.
- Fast tests give value even in a *known broken* state.

Rules for testing:

Rule #3: Make the tests run automatically during code review.

- No need to remember.
- Canonical answer for: "*What are the right tests?*"

Rules for testing:

Rule #4: Make the tests run automatically *after* review.

Trigger *all* the tests on a scheduled basis.

- Anything that is not monitored, **is** broken.
- Flaky tests will get thought.
- Testing logs help locate complex failures.[8]

[8] This assumes that testing logs are being kept... You are, right?

Rules for testing:

Rule #5: Commit to keeping the test passing.

This is not free.

- Buy-in from code owners and users.
- Prioritize it.
- Failing or missing tests should be unacceptable. [9]
- Broken tests, broken windows.
- Tests are part of the job.

[9] But don't go too far. People make mistakes; so plan for, and deal with that.

Rules for testing:

Rule #0 Have tests.

Rule #1 Make it easy to find and run the tests.

Rule #2 Make the tests run quickly.

Rule #3 Make the tests run automatically during code review.

Rule #4 Make the tests run automatically *after* review.

Rule #5 Commit to keeping the test passing.

Rules for the release process:

Rule #0: Have a release process.

What is *not* a release process?

- A documented list of manual steps.
- Another team's responsibility.
- The last step of the project.

What is a release process?

- A formal well defined process.
- Everything; checked in code $\rightarrow$ artifacts ready to ship:
  - Build & test.
  - More tests.[10]
  - Deploy as canary.
  - Generate change-logs.
  - Push the docs.
  - Deploy to production.
  - etc.

[10] including stuff that isn't hermetic.

Rules for the release process:

Rule #1: Ensure that the release process is easy to use.

If it's hard to do on **your** timeline when things are going correctly, then it will be impossible under time pressure when it's most need.

Rules for the release process:

Rule #2: Ensure that the release process is effective.

- Don't over engineer it.
- Keep it simple.
- Keep if flexible.
- Keep it transparent.
- Don't incentives bypassing it.

  Things will go wrong: Allow manual flexibility in the process.
  Things will change: Allow flexibility in the automation.

The Release process is code, follow the same design practices.

Rules for the release process:

Rule #3: Ensure that the release process is automatic.

- Just need to kick off the process.
- Schedule regular releases.
- Keeps it working. Keeps it fresh.
- Breaks get noticed (and fixed) sooner.

Rules for the release process:

Rule #4: Ensure that the release process is safe.

- Full automated test coverage... ideally.
- Tests failing should block a release.
- Overriding the process should be an exception, not the norm.

Basically; CI/CD

Rules for the release process:

Rule #5: Commit to keeping the release ready.

This is not free.

- Buy-in from developers and operations.
- Prioritize it.
- Product owners must schedule time for this.
- Releases are a part of the job.

Rules for the release process:

Rule #0 Have a release process.

Rule #1 Ensure that the release process is easy to use.

Rule #2 Ensure that the release process is effective.

Rule #3 Ensure that the release process is automatic.

Rule #4 Ensure that the release process is safe.

Rule #5 Commit to keeping the release ready.

Rules #0:

- Have a build process.
- Have a code review process.
- Have tests.
- Have a release process.

Rules #1: Good defaults:

- Use the same build process for “everything”.
- Maintain a style guide.
- Make it easy to find and run the tests.
- Ensure that the release process is easy to use.

Rules #2: Avoid blockers:

- Use a scalable build process.
- Maintain defined owners for all code.
- Make the tests run quickly.
- Ensure that the release process is effective.

Rules #3: Automate correctness:

- Use a hermetic build process.
- Maintain automatic checks.
- Make the tests run automatically during code review.
- Ensure that the release process is automatic.

Rules #4: Maintain the invariants:

- Use a reproducible build process.
- Maintain those checks at head.
- Make the tests run automatically *after* review.
- Ensure that the release process is safe.

...

Rules #4: Maintain the invariants:

- Use a reproducible build process.
- Maintain those checks at head.
- Make the tests run automatically *after* review.
- Ensure that the release process is safe.

Rule #4.1: Put the project status on a wall where everyone can see it.

Be careful what you measure,
because that is what you will get.

Rules #5: Plan for the costs:

- Commit to keeping the build clean.
- Commit to doing code reviews.
- Commit to keeping the test passing.
- Commit to keeping the release ready.

The End

Principles

- Any process that depends on humans doing a job perfectly will fail.
- Rarely will a single mistake kill anything. Often however, the first mistake is choosing to be in a position where one more mistake will kill you.
- Be careful what you measure, because that is what you will get.
- For any important part of a system, there should always be (at least) three owners:
  - One to do the work.
  - One to review the work.
  - One who can be on vacation.

Extra

- Link: Impact on maintainability and refactoring for higher-level design features