Simplicity in the Source

April 25, 2024

mteich@google.com

Why?

- I want you to care about simplicity
- I want you to want to care about simplicity
- I want you to not have to rewrite your own project after 5 years

What is "the Source"?

- Source of features
- Source of bugs
- Source of complexity
- Source of documentation?!
- Source of **truth**

Use the force, Luke!

Read the source!

Economies of Scale - R/W Inequality

- Source is read ~10x more often than it is written to (by humans). Think review, debugging, understanding how something works (source is the source of truth, not documentation), and finally turndown.
- Keep the audience in mind, not everyone is as smartass as you are.
- If you spend 1 minute to clarify your source so that each reader needs 10 seconds less to understand it, that's a big win.

Measuring Complexity - Linecount

- Same functionality in less lines of source code
- Can be very problematic in the extreme (obfuscation, arcane language features)
- Heavily depends on programming language
- Count comments, empty lines for visual grouping, lines with only `{` / `}`?

Measuring Complexity - Cyclomatic, Cognitive

- score each function based on number and nesting levels of loops and conditions
- **can** be a good signal in many cases
- not so much for complex algorithms, repetitive conditionals
- extracting a new function that would only be used in one call site **can** hurt readability because now the reader needs to jump around to understand the full source.

Exponential Configuration Complexity

- Almost always when you add a new special configuration (setting, flag, experiment, hardware variant), you're not just adding new special configuration, you're adding a new dimension in your configuration space and with that you're multiplying the number of possible combinations.
- Impossible to test all combinations
- Interactions between configuration dimensions become harder to understand and reason about, it becomes frustrating to work on the project, starting its decay and demise -> abandoned, costly rewrite or turndown
- Practical advice: Think twice before adding a new flag. If possible, have a clear plan to remove it again asap.

Feature Creep? Complexity Creep!

- wego case study
  - personal fun project
  - went "viral"
  - lots of feature requests
  - nice guy me tried to make everyone happy by adding all the features
  - fulfill all niche requirements, but source became unmaintainable
  - project abandoned :(
  - -> question every user request critically. Is the change in scope for the project? Would the change benefit >80% of users? Would it be a reasonable default? If not, reject it.

Write for Humans, not for Computers!

- If your source is clear, you don't need a lot of comments explaining it.
- Compilers are extremely efficient at optimizing source for performance, don't try to beat them unless you have an actual performance issue.
- Simpler source is **easier to maintain** and thus has a **higher life expectancy** and **avoids costly rewrites**.

Tips: Write for Humans not for Computers

- Use long flag names in scripts so the reader doesn't have to open the man page.
  - `grep -B 5 -m 1 pattern`
  - `grep --before-context 5 --max-count 1 pattern`
  - Also helps with searching for flag usage.
- Prefer `--quiet` or `--silent` over `&>/dev/null`
- Avoid this style

Incident Response

- Time is critical.
- More time to understand the broken piece of source ->
  - More error budget consumed.
  - More money lost.
- Hyperlink from logs to source.
- Embed your stack trace in your error message.
  - Canonical in Go: `return fmt.Errorf("parse config: %w", err)`
  - Quickly reconstruct error condition with very basic and reliable tools (log message, source).

Delete

- Antoine de Saint Exupéry: "It seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away"
- Duplicates: Projects that achieve the same goal.
- Command line flags - most are never used
- Variables / constants only used once. Inline them so readers don't have to jump around the source to find their value.
- User interaction points - "Almost all user input is error"

Idempotency - Just Execute!

- Don't check for existence of a file / binary or state of something before acting. Just execute the step and handle the error (if any).
- Multiple problems with checking first:
  - race condition: state could be changed by another process between check and execution
  - harder to read for humans (and computers too)
  - Mostly no benefit -> dead weight
- Idempotency: Ignore current state, ensure intended state. Like a recipe: "Fill bowl with 1 liter of water", not "If the bowl is empty, fill it with 1 liter of water"
- Your enemies: `test -f`, `systemctl status | grep`
- Your friends: `rm -f`, `mkdir -p`, `os.MkdirAll`, `systemctl (re)start`

Consistency! Consistency everywhere!

- Helps with reading source, because our minds don't have to adapt to different styles
- When modifying existing source, adopt its style. Don't enforce your own preference.
- Local consistency (e.g. same file, same project) is much more important and easy to achieve than global consistency (e.g. programming language style guide, company level)

Locality

- Keep parts together that are relevant to each other
- Why: Avoids scrolling / jumping around when reading source
- Declare variables in as narrow a scope as possible
- Assign value to a variable as late as possible
- Reading source becomes easier because you don't need to keep as much state in your head.

Interfaces

- Make your library easy to use
- Make your library hard to use incorrectly
- Avoid complicated multi-step initialization sequences
- Use clear function and parameter names

Avoid Advanced Language Features ("Syntactic Sugar")

- Makes the source easier to read for people who are not familiar with the language already
- E.g.: Ternary operator in C: `int foo = bar ? baz : frob;`
- Perl secret operators
- Learn and stick to the "idiomatic" style of the language
  - `for (int i = 0; i < n; i++)` (idiomatic)
  - `int i = 1; for (; i <= n; ++i)` (non-idiomatic)

Before Asking For Review

- Detach yourself mentally from the source you've just written.
- Forget everything about it. Get coffee.
- Read it.
- Is it clear?
- Is it easy to understand?
- Is it fun?
- Take **one minute** to improve what you found.

Homework Thyme!

1. Open https://go.dev/play/p/XKbCdoxZRff
2. Simplify it!
3. Send it to me: mteich@google.com
4. Receive my simplified version in return

Takeaways

- Source is read ~10x more often than it is written.
- Keep the audience in mind, not everyone is as smartass as you are.
- Spend 1 minute to clarify your source so that each reader needs 10 seconds less to understand it.

References and Reading / Watching Material

- KISS - keep it simple, stupid
- UNIX Philosophy
- suckless.org
- clean coder - Bob Martin
- 5 Step Engineering Principles - Elon Musk

Answers