Code Quality

I have a mentimeter

https://www.mentimeter.com/app/presentation/albm89xigv5mfzmetb5cr5npagey5ajv/edit?source=share-modal

Different quality attributes for different kinds of quality

Functional quality:

- Accurate, Complete, Consistent, Useful, Precise, Informative

Different quality attributes for different kinds of quality

Functional quality:

- Accurate, Complete, Consistent, Useful, Precise, Informative

Deep quality:

- Having flow, Fitting to human needs, Being beautiful, Anticipates, Inspires, Surprisingly great (Stripe.com credo)

Different quality attributes for different kinds of quality

Functional quality:

- Accurate, Complete, Consistent, Useful, Precise, Informative

Deep quality:

- Having flow, Fitting to human needs, Being beautiful, Anticipates, Inspires, Surprisingly great (Stripe.com credo)

Functional quality constraints <-> Deep quality liberates

Another kind of quality

Process quality:

- Traceability, Training, SOPs, Limiting non-conformities

Another kind of quality

Process quality:

- Traceability, Training, SOPs, Limiting non-conformities

Product quality:

- Reliability, Feedback, Specifications, Limiting Defects

The relation between them

- Deep quality requires (a lot of) functional quality
- Product quality (often) improves with process quality

TOC

- Impact of code quality
- Let’s talk code
- How to assure code quality

1. Impact of code quality

Who?

Who?

- Users
- Developers
  - Other developers
  - External developers
  - Future developers
  - Future you (!), on a tight deadline
- Support Engineers
- AI?

Who?

- Users
- Developers
  - Other developers
  - External developers
  - Future developers
  - Future you (!), on a tight deadline
- Support Engineers
- AI?

“When the code was written, only you and God understood it. Now only God understands it”

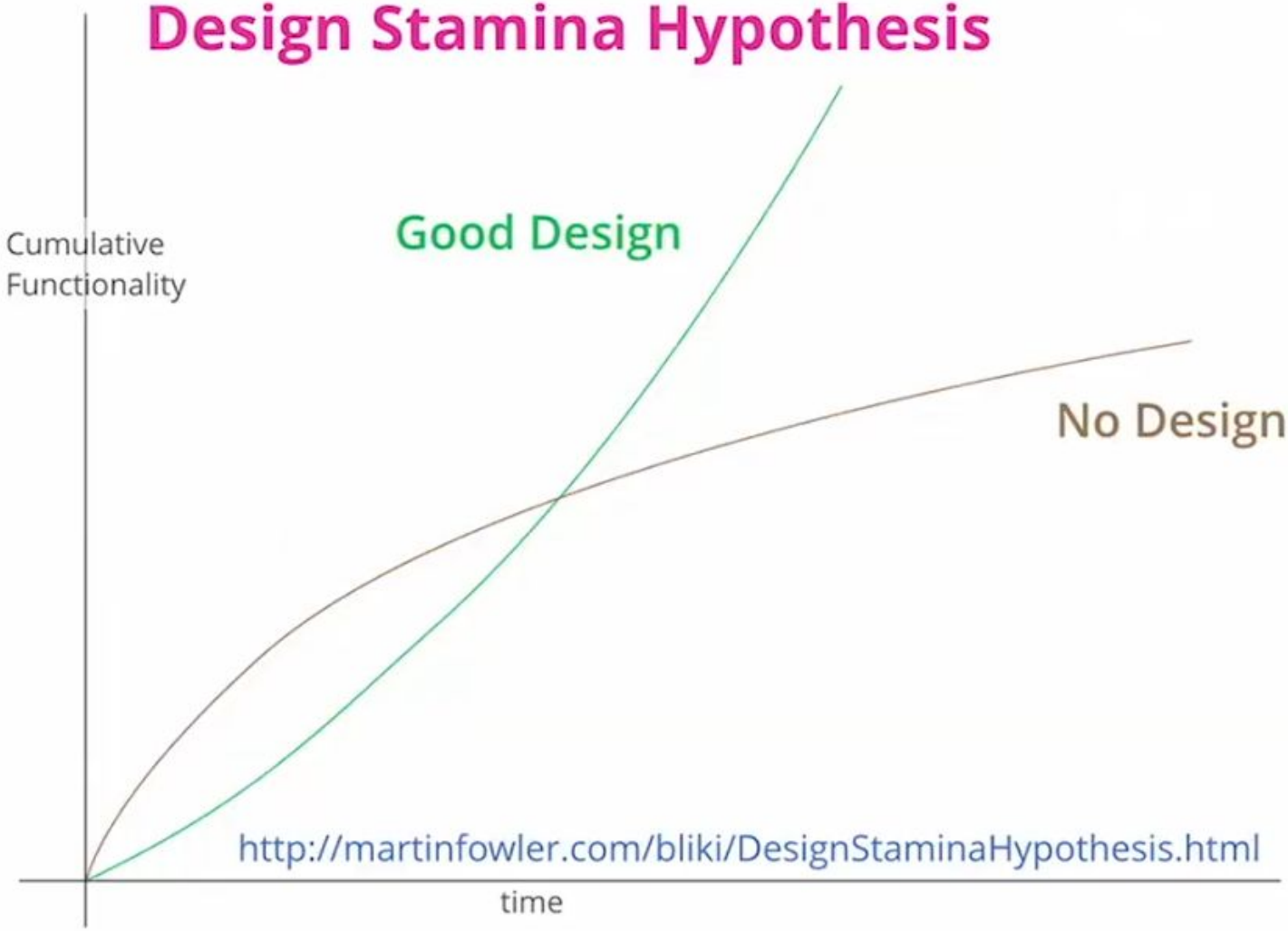
Why?

Why?

- Users
  - Want to do something
  - interacting
  - waiting
  - reads output (error messages)

- Developers
  - Extending
  - changing
  - fixing functionality
  - navigating codebase

Design Stamina Hypothesis



Why?

- Future Developers → Legacy maintenance
- Other developers → Can it cooperate?
- External developers → Can it collaborate?
- New developers → Onboarding friction

Why?

- Service engineers
  - Is problem human error or bug? (Error messages again)
  - Can problem be fixed from the database?
  - Restarting a failed system
  - Often not a lot of context (maybe not even the code)

- AI?
  - Also misses a lot of code (only has the code)

Why?

“When you wrote the code, Only you and God understood it. Now only God does.”

2. Let’s talk code

Subjects

- A core design philosophy: KISS
- Code is read more then written
- Software Architecture
- Software Testing
- “Talkative” software
- External factors: Humans and Security
- Some common pitfalls

A core design philosophy: KISS

- KISS: Keep It Stupid Simple

- DRY vs WET
- YAGNI
- Premature optimization
- Code golfing
- Cargo Culting

- Remember: Human readable

A core design philosophy: KISS

- KISS: Keep It Stupid Simple

- DRY vs WET
- YAGNI
- Premature optimization
- Code golfing
- Cargo Culting

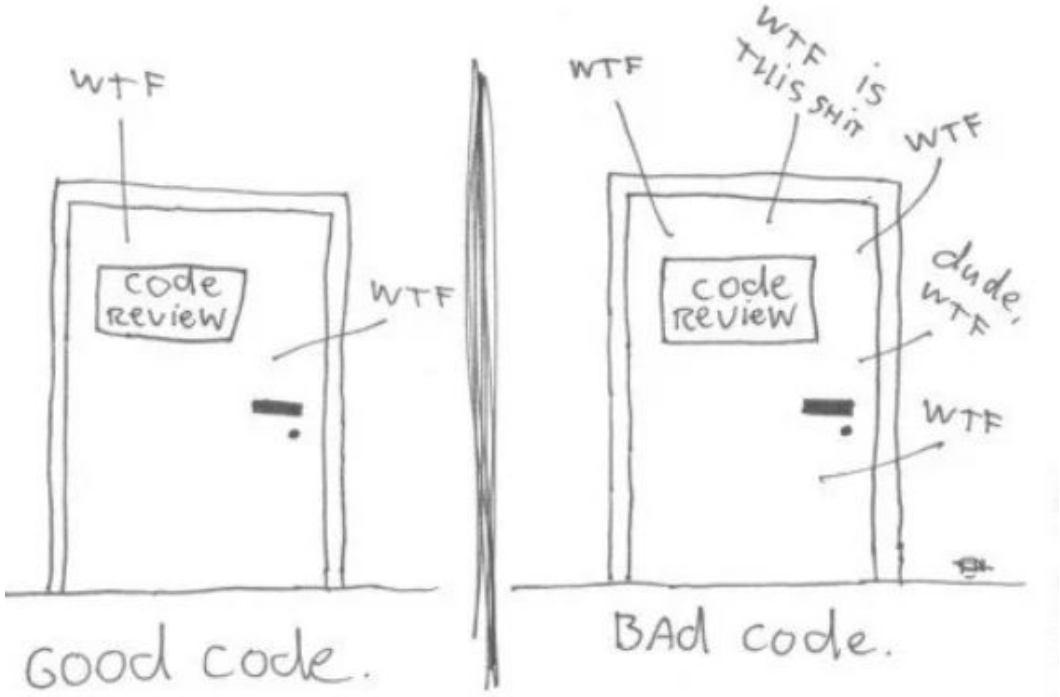
- Remember: Human readable

> “All problems in computer science can be solved by another level of indirection… except for the problem of too many layers of indirection.”

Code is read more then written

- Variable naming - hardest things in software engineering
- Comments for what is not obvious - Context or decisions
- POLA - Principle of Least Astonishment
- Docstrings on functions/API - Automatic reference documentation
- Technical documentation - Design/architecture choices

The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



Software Architecture

- Complexity reduction:

- Change amplification
- Cognitive load
- Unknown unknowns

Software Architecture

- Building blocks “seamed” together
  - Seams can easily be ripped apart and (re)constructed
  - Building blocks are harder to picked apart / understood

Software Architecture

- Building blocks “seamed” together
  - Seams can easily be ripped apart and (re)constructed
  - Building blocks are harder to picked apart / understood

- What and where are those seams?
  - REST API
  - Database layer
  - Datastructures
  - Files
  - BAM files
  - Sometimes already dedicated by technical limitations / requirements / framework

Software Architecture

- Building blocks “seamed” together
  - Seams can easily be ripped apart and (re)constructed
  - Building blocks are harder to picked apart / understood
- What and where are those seams?
  - REST API
  - Database layer
  - Datastructures
  - Files
  - BAM files
  - Sometimes already dedicated by technical limitations / requirements / framework
- Topic too big too cover fully here

Tactics in Software Architecture

- Consistent use of seams
  - REST API → Topic on itself
  - Document year seams

Tactics in Software Architecture

- Consistent use of seams
  - REST API → Topic on itself
  - Document year seams

- Idempotent
- Atomic
- Preserve invariants
- Fail fast
- Single source of truth (code/data/knowledge(!))

Martin Fowler on Software Architecture

“Software Architecture is a shared understanding”

Testing

- Not just for verification

- Explaining intent
- Easier to test → better seams → better code
- Gives the power of fearless refactoring

- Build test suite for each new bug / edge case
  - Prevent regressions!

Testing

- Not just for verification

- Explaining intent
- Easier to test → better seams → better code
- Gives the power of fearless refactoring

- Build test suite for each new bug / edge case
  - Prevent regressions!

- Again topic on itself

Software that talks - To the outside

- Error handling
- Logging

- For both users and developers (and support engineers)
- Showing intent
- Diagnostics

Error handling

- Input sanitization (fail fast)

- Additional information in your error messages
  - For what patient/analysis failed
  - What looked different then expected
  - What common solution is there for this problem?

Error handling

- Input sanitization (fail fast)

- Additional information in your error messages
  - For what patient/analysis failed
  - What looked different then expected
  - What common solution is there for this problem?

- Needs a mechanism in the first place to handle errors

Logging

- Using log levels
  - DEBUG
  - INFO
  - ERROR
- Using hierarchical component structure
  - Easy filtering
- Log handling
  - A stream can be corrected
  - How much do you keep?

External factors

- Control on external factors
  - A lot comes together: Robust architecture, Error handling, Logging

- Recover from unexpected external factors
  - System misuse (when it was slow)
  - System crash / reboot
  - External system down
  - Disk full
- User input sanitization
- Security
  - Permission system
  - Least privilege principle

Security

- Secure by design
- Audit trail (logging)
- Access control (permission system)

- Logging
  - Bug hunting
  - Monitoring
  - Postmortem (breaks and breaches)

User input sanitization

- Unreliable input

- Whitespace, meaningless characters (,/;)
- Duplicate data
- Enforcing with a regex

- Often target area of attack

Some common pitfalls

- Metrics miss units of measurement
  - Was it in milli or micro?

- Magic variable
  - Some important variable is deep in the code hidden
  - Could be configuration or class variable

- Edge cases
  - eg ZeroDivisionError
  - Bug or edge case or external factors?
  - This needs comments
  - And tests

3. What assures quality?

What assures quality

- Starts with the architecture
- Automate the simple stuff: CI : linters, formatters, testing
- Document decisions / agreements
- Compliance documents (or does it?)
- You

- But finally: It is a habit and culture

Starts with the upfront design (document)

- Style guide / Architectural documentation
  - Doing common patterns (correct “seams” use)
  - Howto logging / permission checks / implement X
  - Naming convention (to be consistent)

- Choices
  - Any automated CI checks
  - What to test / test coverage

Code review

- More then functional quality

- Goals
  - Teaching (both sides) of both
    - the code
    - common patterns
    - new techniques
    - Much more powerful first hand!

  - Code is understandable - Proof is in the pudding
  - Hawthorne effect - Better performance when observed
  - Finding bugs?

Code review

Different kind of review comments

- Nits
- Suggestions
- Questions
- Clarifications
- Blockers

Psychological safety

- About code, not people - for both sides
- If it is really bad - problem with upfront communication


YOUR CODE IS GOOD
BUT IT CAN BE BETTER
imgflip.com

Actionable items:

- Merge request template → Add your own checks which are often forgotten
- Acquire more skill → Framework documentation / Blogs

Actionable items:

- Merge request template → Add your own checks which are often forgotten
- Acquire more skill → Framework documentation / Blogs

- Plan around it
  → Cooldown weeks

  - Stuff not on the “roadmap”
  - Cleanup week
    - Documentation
    - Tests
    - Logging
  - Weeks dedicated to quality improvement

Take-aways

- Write your code
  - Simple
  - With the intent to be read
  - Not just to get it done
  - More than “the happy path”
  - Not for machines, but for humans

- Quality takes time and practise, but start where it counts the most


CODE QUALITY?
AIN'T NOBODY GOT TIME FOR THAT
imgflip.com

Discussion and Questions