

UNIVERSITEIT VAN AMSTERDAM

Code Quality

Software Construction 2018

Dr. Vadim Zaytsev aka @grammarware

raincode

LABS
— compiler experts —

Test

- Link in today's email
- Next week: last lecture!
- Prepare a **screen**cast (~30min)

Screencast

- Technically any platform
- Identification (accounts + photos?)
- Language + frameworks
- Parsing (syntactic analysis)
- AST (design)
- Static analysis (type check)
- Interpretation (rendering)
- Styling (QLS)

Code Quality

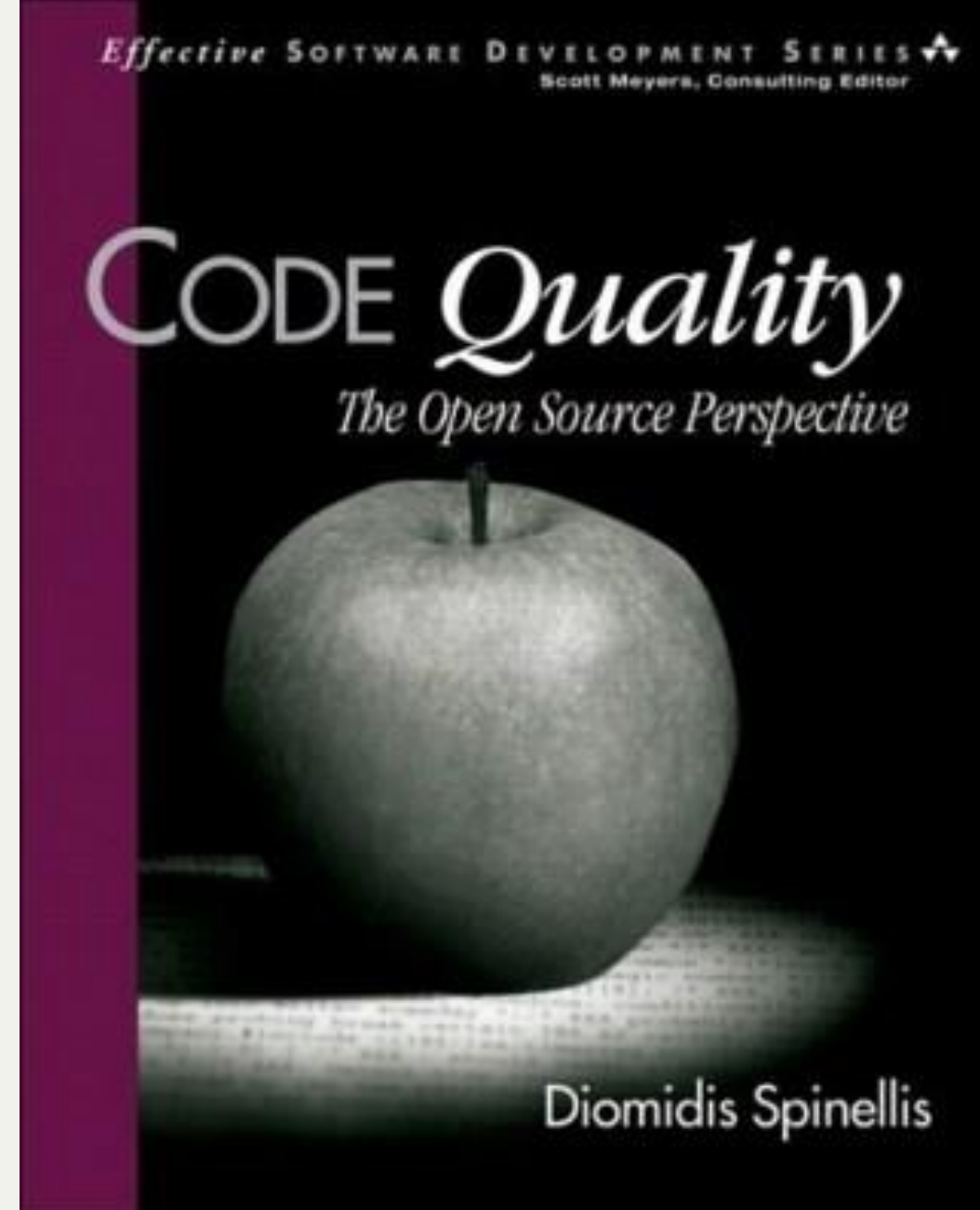
- Robustness of code
- Amount of flaws / defects
- Lack of errors / less bugs
- Maintainability / readability
- Requirements satisfaction / meeting standards
- Meeting functional req / nonfunction req
- How well software is designed / complies to design
- All sorts of things

(your test results)

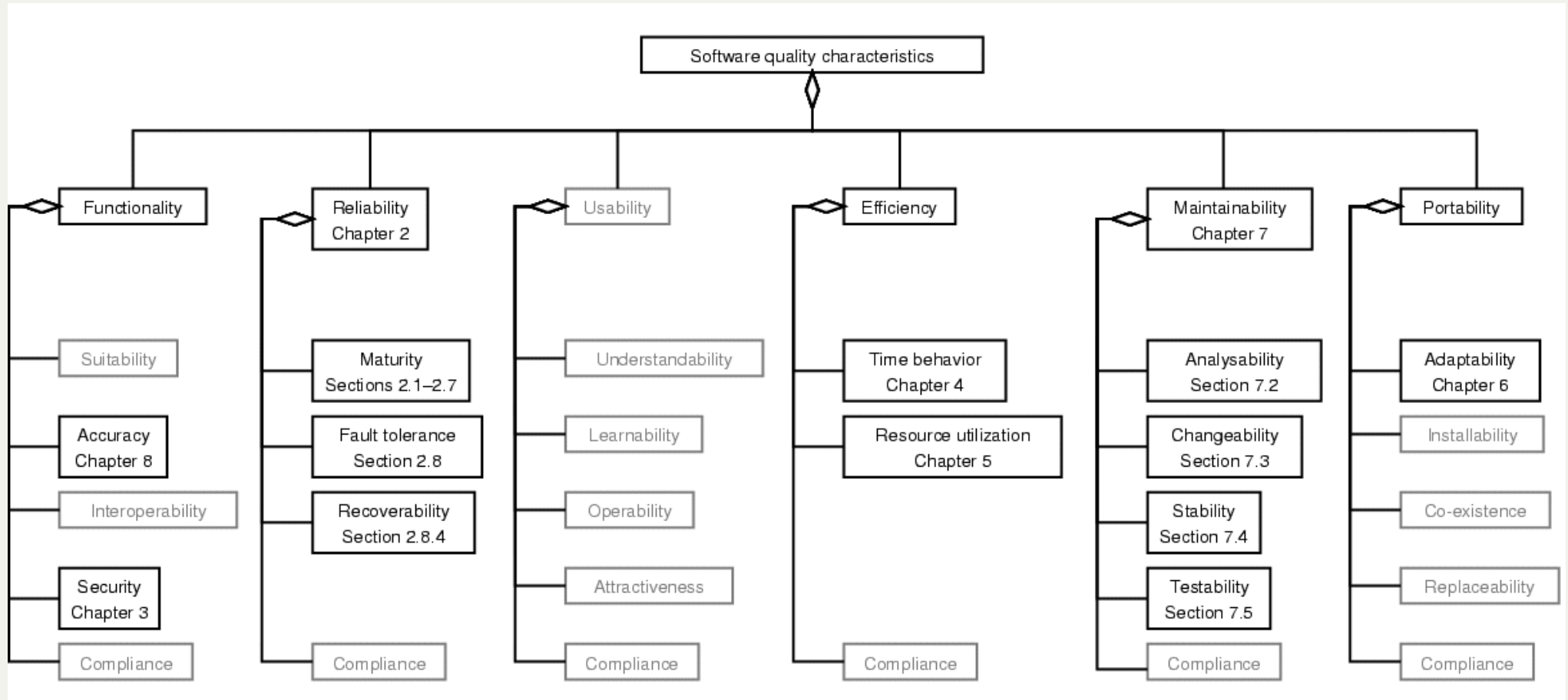


Code Quality

- Quality in use
 - bugs that manifest
- External quality
 - benchmarked
- Internal quality
 - examine, not run
- Process quality
 - accident?



Software quality characteristics



Functional suitability

- Obvious
- Completeness
 - cover all tasks & objectives
- Correctness
 - accuracy of results
- Appropriateness
 - suitability

Correctness example: floating point

- Inherently imprecise?
- ANSI/IEEE 754-1985 aka IEC 60559:1989
- Integers from $[-2^{53}, 2^{53}]$ are exact
 - represented by bit sequences
- $0.5 = 2^{-1}$ is exact; $0.0126953125 = 2^{-7} + 2^{-8} + 2^{-10}$ is exact
- $0.2 \simeq 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-11} + 2^{-12} + \dots + 2^{-54}$
 $= 0.19999999999999999555910790149937383830547332763671875$
- Error is measured in **ULP**
- Rounding? Memory format? Implied 1? Overflow? Cancellation? Absorption?

Performance & Efficiency

- Time behaviour
 - latency (response time), processing time
- Resource utilisation
 - humans included
- Capacity
 - bandwidth, throughput, database size

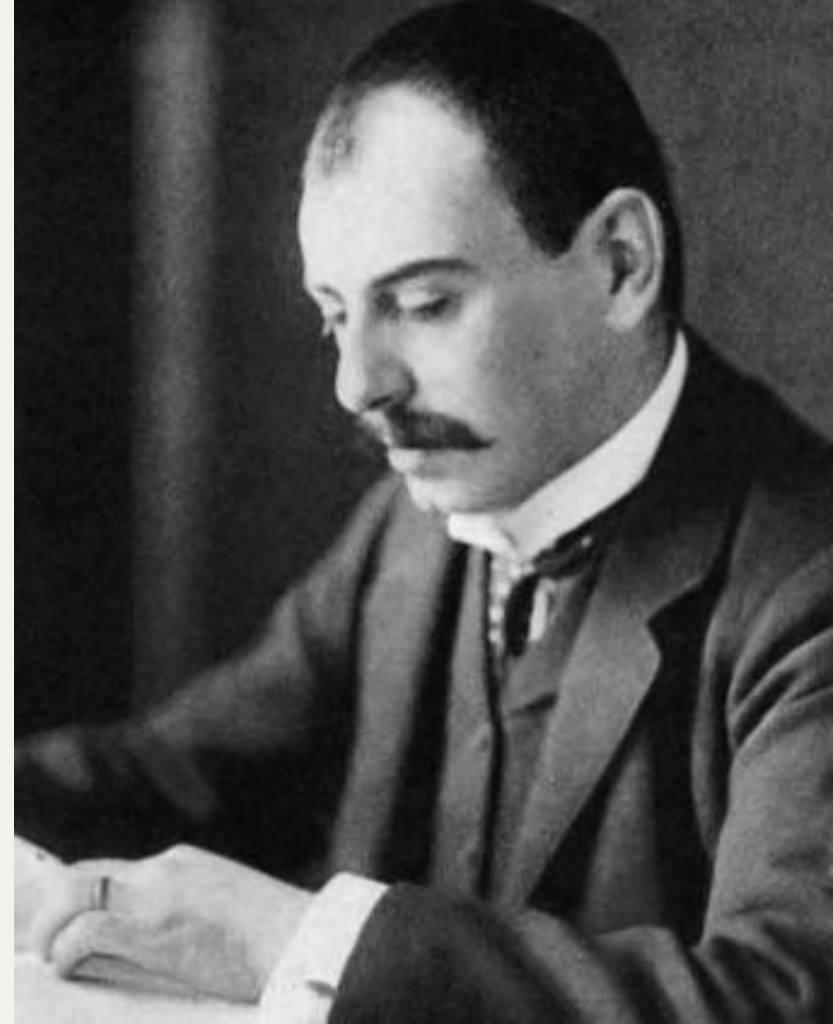
Quotes about efficiency



- *Do not strive to write fast programs—strive to write good ones.*
(Joshua Bloch)
- *Premature optimization is the root of all evil (or at least most of it) in programming.*
(Donald Knuth)
- *A fast program is just as important as a correct one—false!*
(Steve McConnell)
- *Optimizations always bust things, because all optimizations are, in the long haul, a form of cheating, and cheaters eventually get caught.* (Larry Wall)
- *The key to performance is elegance, not battalions of special cases. The terrible temptation to tweak should be resisted unless the payoff is really noticeable.*
(Jon L. Bentley & M. Douglas McIlroy)

Perf analysis

- Workload
 - user time; kernel time; idle time
- Profiles
 - running time \sim user time?
 - kernel time $>$ user time?
 - running time \gg user time + kernel time
- Algorithm complexity
 - $\text{const} < \log < \text{linear} < \text{loglinear} < \text{quadratic} < \text{cubic} < \text{exp} < \text{fac}$
- Average complexity vs worst-case complexity

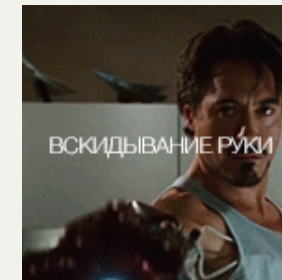
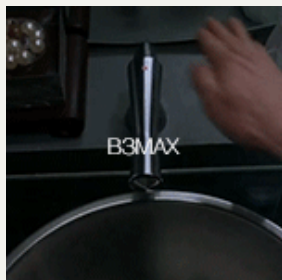


Compatibility

- **Co-existence**
 - reach goals without bad impact on others
- **Interoperability**
 - systems can collaborate to reach goals
 - seamlessness?
- **When the concern was raised?**
 - early: common central design
 - mid: mappings to/from common schema
 - late: megamodelling & synchronisation

Usability

- Appropriateness recognisability
 - Learnability
 - Operability
 - User error protection
 - UI aesthetics
 - Accessibility
-
- Questionnaires!



N. Shedroff, C. Nossel, *Make It So*, 2012.
J. Kosinski, *Oblivion*, 2013.

Reliability

- **Maturity**
 - reliable in normal circumstances
- **Availability**
 - is up
- **Fault tolerance**
 - reliable despite faults
- **Recoverability**
 - reestablish program state
- *Investment in reliability will increase until it exceeds the probable cost of errors, or until someone insists on getting some useful work done.*
(Gilb's Fourth Law of Unreliability)

Maturity up close

- **Input**
 - lexer/parser, XML lib, validating widgets; shotgun parsing
- **Output**
 - incompleteness or wrong format
- **Logic**
 - off-by-one, neglected extremes, forgotten cases, missing methods
- **Computation & data handling**
 - algorithm, operand, operators, uninitialised, null, type cheating
- **Timing**
 - race conditions
- **Interfacing**

Security

- Confidentiality
 - only authorised data access
- Integrity
- Non-repudiation
 - actions can be proven to have taken place
- Authenticity
 - provable identity
- Accountability
 - actions traced back to the entity



OF PASSWORDS

Some security advice

- Ignore vulnerable code
- Most common vulnerability?
- Race conditions are insecure
 - TOC2TOU
- Some API are more secure than others
 - strcpy vs strncpy, gets vs fgets
- Shell
 - metachars, .com vs .exe
- Temporary files and other forms of leakage

Maintainability

- **Modularity**
 - changes do not propagate, coupling & dependencies, separation
- **Reusability**
 - can be reprofiled
- **Analysability**
 - consistency, conventions, indentation, size
- **Modifiability**
 - changeability, stability, identification, patterns, encapsulation
- **Testability**
 - unit, integration, system, incidental, logging,

Portability

- **Adaptability**
 - can it coevolve with the environment?
- **Installability**
 - how to fit in an environment?
- **Replaceability**
 - can it replace an alternative product?

GUI portability strategies

- Unportable
 - application uses OS directly
- Portability layers
 - app is built on top of interchangeable PLs
- Emulation layer
 - native calls + foreign OS emulation
- Portable platform
 - JVM, .NET Core, JS, Tcl/Tk
- Internationalisation!

Conclusion

- ISO 9126 or 25010 is useful
- Software product quality has many aspects
 - functionality, performance, reliability, ...
 - each requires its own approach
- Analysing with metrics is OK, but...
- Start wrapping up at the lab
- Refactor!
- Prepare to share

