Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Informatik

Software Architecture and Techniques

Errors, Vulnerabilities, Smells
In
Source Code

Marcel Baumann, tangly llc

CC BY

Clean Code

- Simplest step to improve your source code is to use tools
- Tools are cheap, fast, and do not require coordination with experts
- Tools can only find **non-quality**
- Practice daily to improve – same as to go to the gym -

Clean Code

- Compiler errors
- Compiler warnings
- Static checks
  - Bugs → high probability it will crash
  - Errors → it can crash
  - Vulnerabilities → it can be hacked
  - Smells → it will cost to maintain

Tools

- Analyze with your IDE functions
- Jacoco
- SpotBugs
- SonarLint and SonarQube
- Checkstyle
- PMD

Sonar Rules

- Around 500 rules only for Java code
- Subset of OWASP vulnerabilities
- De facto standard
- If you find a better tool, just use it

*Goal: Improve Quality of your **product** and source code*

OWASP

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities XEE
- Broken Access Control
- Security Misconfiguration
- Cross Site Scripting XSS
- Insecure Deserialization
- Using Components with known vulnerabilities
- Insufficient Logging and Monitoring

Why Tools?

- It is cheaper to use tool than to use humans to review code
- You can do it every few minutes
- Nobody is watching over your shoulder
- But tools can only find simple problems

The approach we recommend to code quality?

Manage it as a water leak, fix the leak before you mop the floor!

Goals

- No compiler errors
- No compiler warnings
- No Sonar, Spotbugs errors, vulnerabilities or smells
- Code coverage shall be higher than 60%
- **Every found bug has a test reproducing it before you correct the error**

Why Pair Programming?

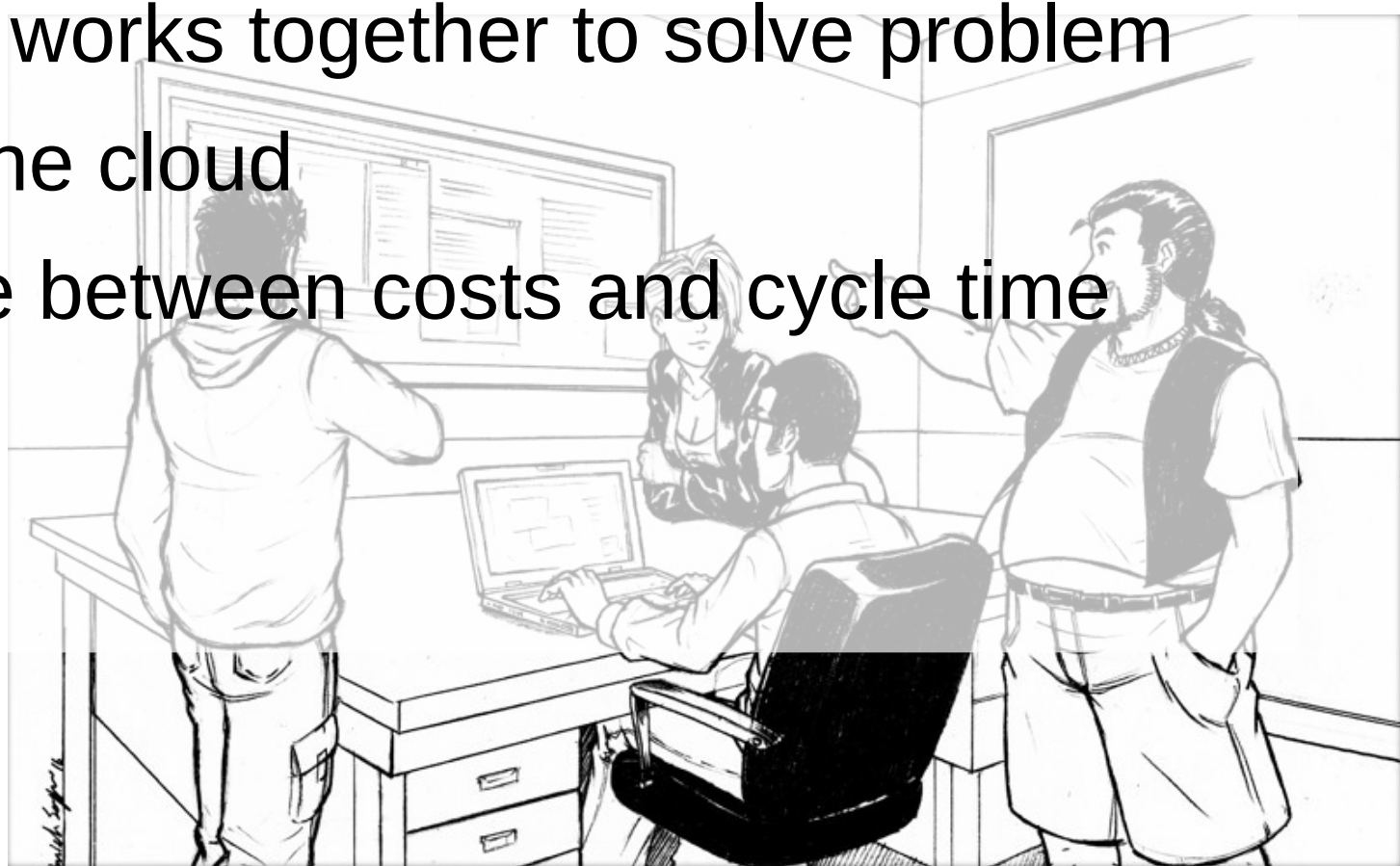
- Tools only detect simple semantic problems
- People help you to improve your design
- People help you to get started with architecture
- Multiple team members know the code

Wisdom of the crowd

pair programming, mob programming

Next Stage: Mob Programming

- Whole team works together to solve problem
- Wisdom of the cloud
- Compromise between costs and cycle time



Technical Meetings / Dojos

You still should hold coding dojos

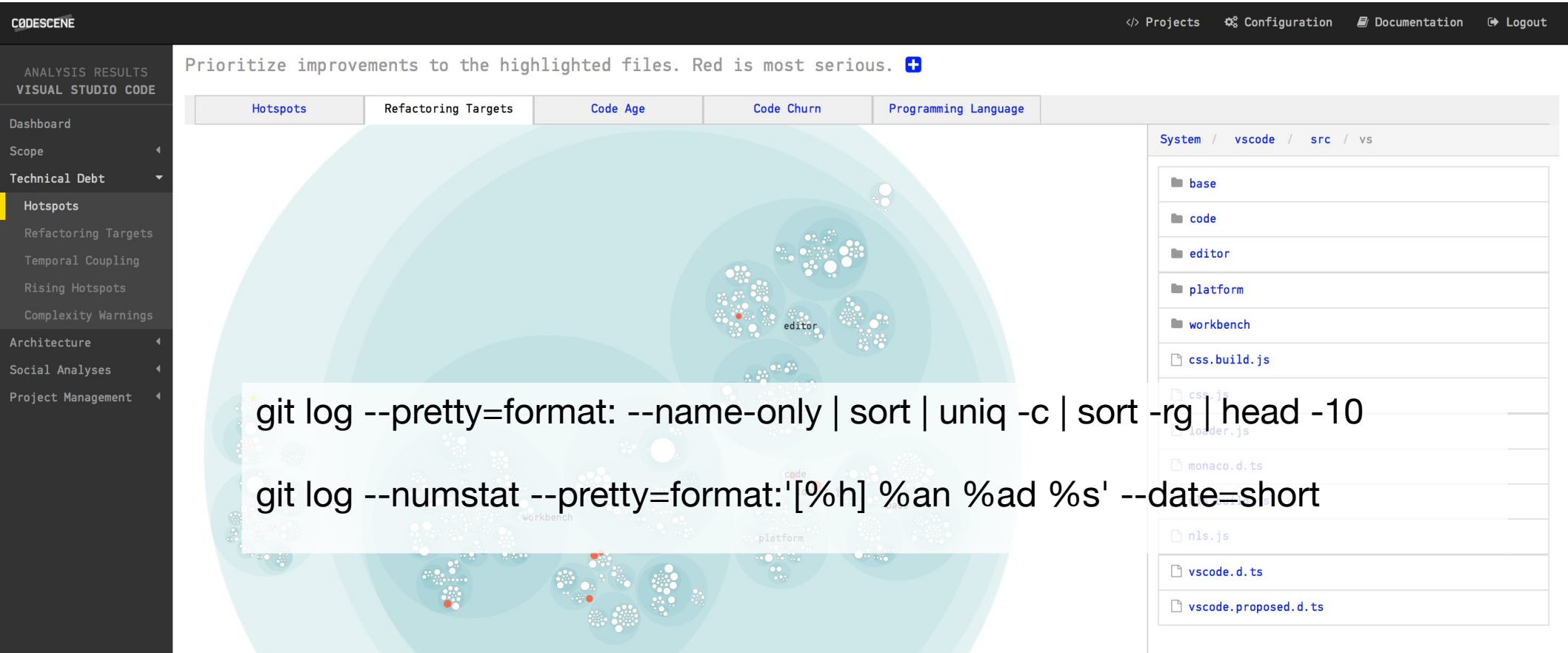
You still should do architecture workshop each sprint

Remember: Tools find non-quality, they currently cannot measure quality

SonarLint and SonarQube

- Work in pair
- Run SonarLint “Analyze with SonarLint”
- Read the generated report
- Study the rule description
- Repeat

Forensics Approach



Advanced Tools

- Module concept of Java 9
  - Compiler validation of dependencies and visibilities
- ArchUnit
  - Codify dependency rules as unit tests

Modules

- Huge impact on architecture
- Still “bleeding edge” in Java – Java communities are laggards -
- Formalize bounded domains
- Compiler validation

ArchUnit

- Good approach before modules
- Custom rules for specific needs
- “Bleeding edge” with Java – sometimes laggards with new features -

DevOps Approach

- Tools make only sense if they are automatically triggered in your CI/CD
  - They are part of your Gradle or maven build
- Quality gates are the corollary

Zero Bug Policy

- No Open Bug
- Fix it or forget it
- Deliver quality
- Have happy users
- No more bug board

Bad APIs

- Force clients to write bad code
- Lack of consistency in nomenclature
- Centralize access to the features in a single class
- Do not use immutable objects
- Do not document your API
- Use old Java style

Bad Scrum / Agile

- Missing Definition of Done
- Missing git training
- Missing coding guidelines
- Missing deployed application multiple times per week
- Missing DevOps discipline

Exercises (1/3)

- Read the cheat sheet “Clean Code”
- Code coverage with with IntelliJ and Jacoco
  - How to improve code coverage?
  - Why should you improve code coverage?
  - How much should you improve code coverage?
- Static checks with IntelliJ and SonarLint

Exercises (2/3)

- Coding dojos with student code examples
  - Remove smells
  - Refactor
  - Unit tests
  - Test driven tests
  - Always regularly commit to git with meaningful comments

Exercises (3/3)

- Optional exercise
  - Connect your gitlab project to SonarQube
  - Extend your pipeline to generate SonarQube metrics
  - Study your git history over time
    - Read Martin Fowler post
      Patterns for Managing Source Code Branches