



Foto: Thomas Josek

Software Engineering

QA + Testing I

Software & Systems Engineering | Prof. Dr. Andreas Vogelsang | 04.12.2023



@andivogelsang



vogelsang@cs.uni-koeln.de

Learning Goals for Today

- Know what Quality Assurance is and what qualities of software need to be ensured
- Know the general strategies to quality assurance
- Know what code review is and how to apply it
- Know what testing is, what its main goal is, what types of testing exist



Software Quality and its Assurance



Andy Hunt

“No one in the brief history of computing has ever written a piece of perfect software. It’s unlikely that you’ll be the first.”



Donald Trump (May 2020)

“If we didn’t do any testing, we would have very few cases.”

Software Quality

Quality [Ludewig and Lichter]

Quality is the entirety of properties and characteristics of a product or process that indicate adequacy with respect to given requirements.

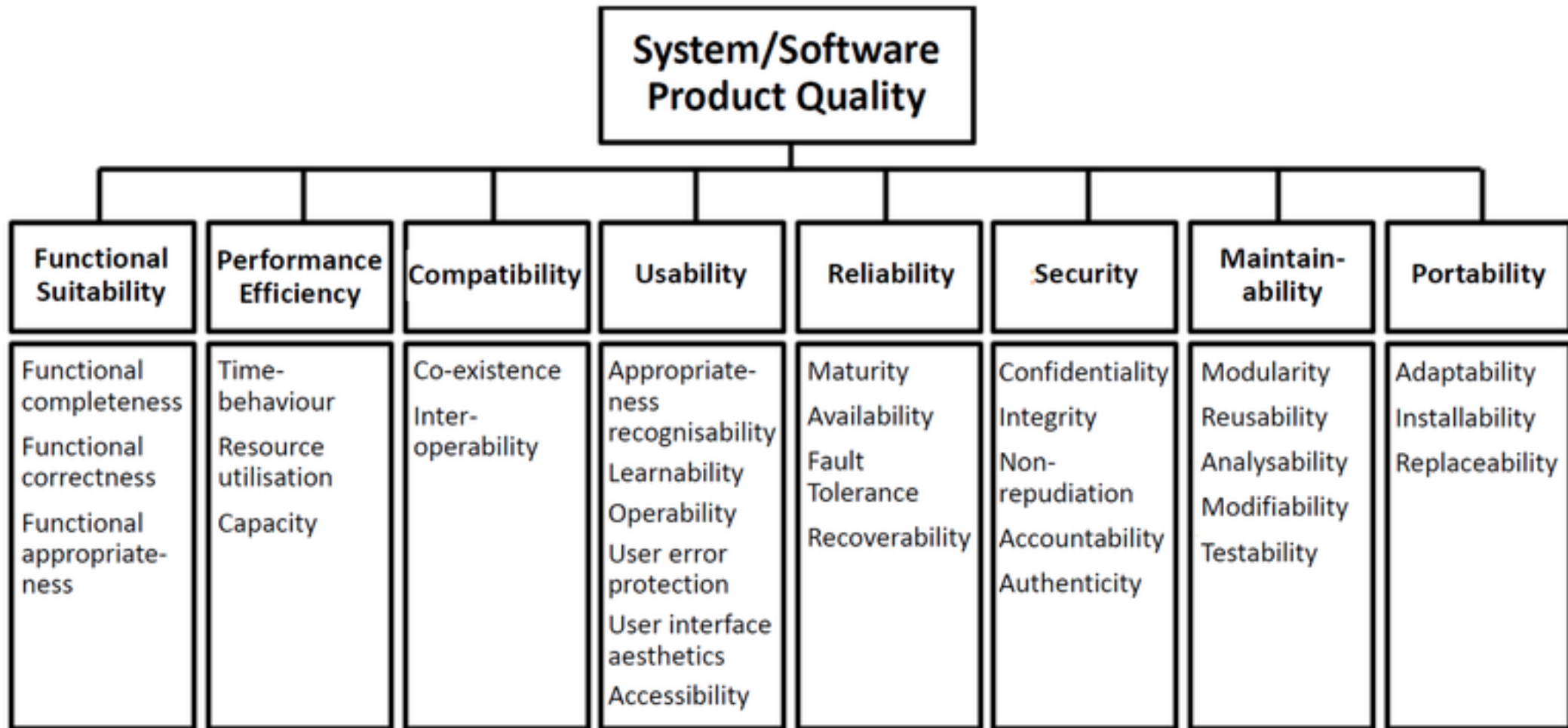
Quality Assurance [Ludewig and Lichter]

Quality assurance are all activities with the goal to improve the quality.

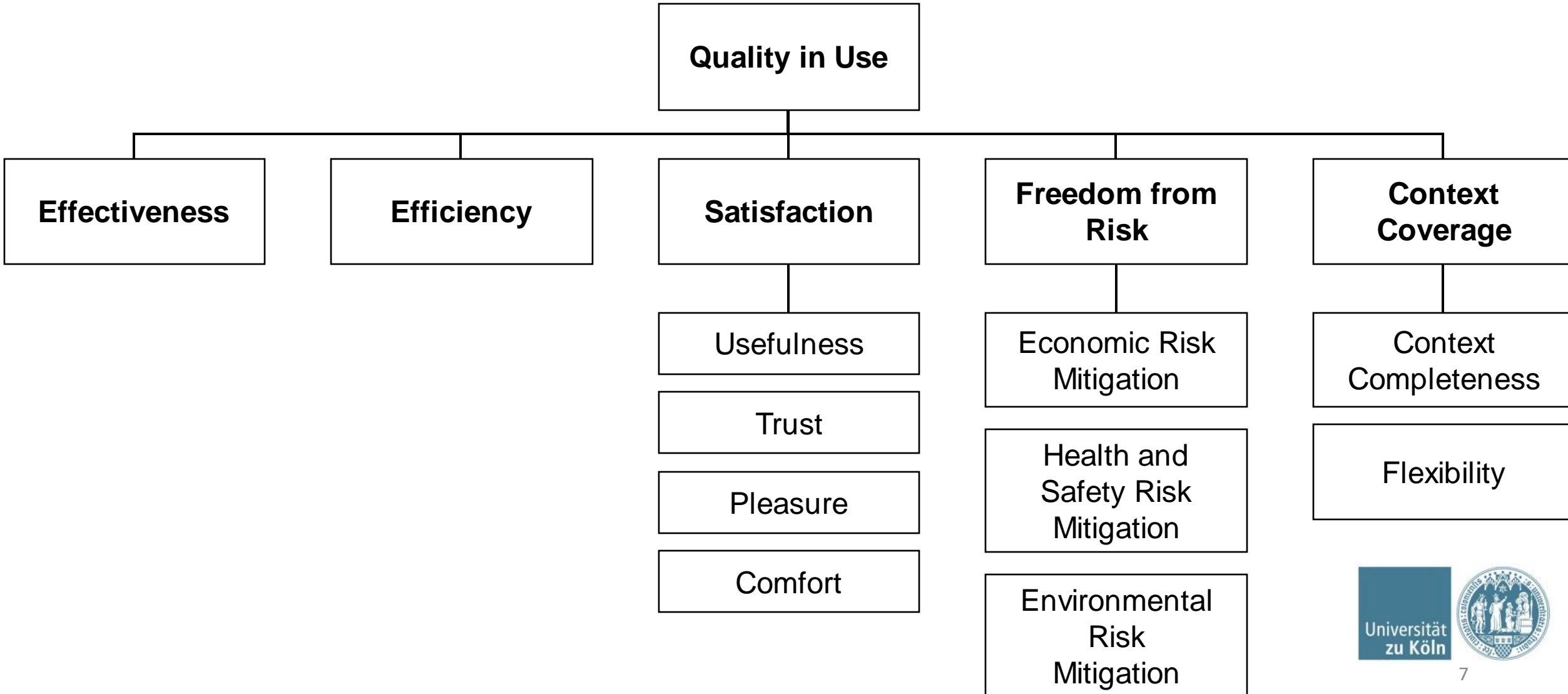
Expectations on Quality [Sommerville]

“Because of their previous experiences with buggy, unreliable software, users sometimes have low expectations of software quality. They are not surprised when their software fails. When a new system is installed, users may tolerate failures because the benefits of use outweigh the costs of failure recovery. However, as a software product becomes more established, users expect it to become more reliable. [...] If a software product or app is very cheap, users may be willing to tolerate a lower level of reliability. [...] Customers may be willing to accept the software, irrespective of problems, because the costs of not using the software are greater than the costs of working around the problems.”

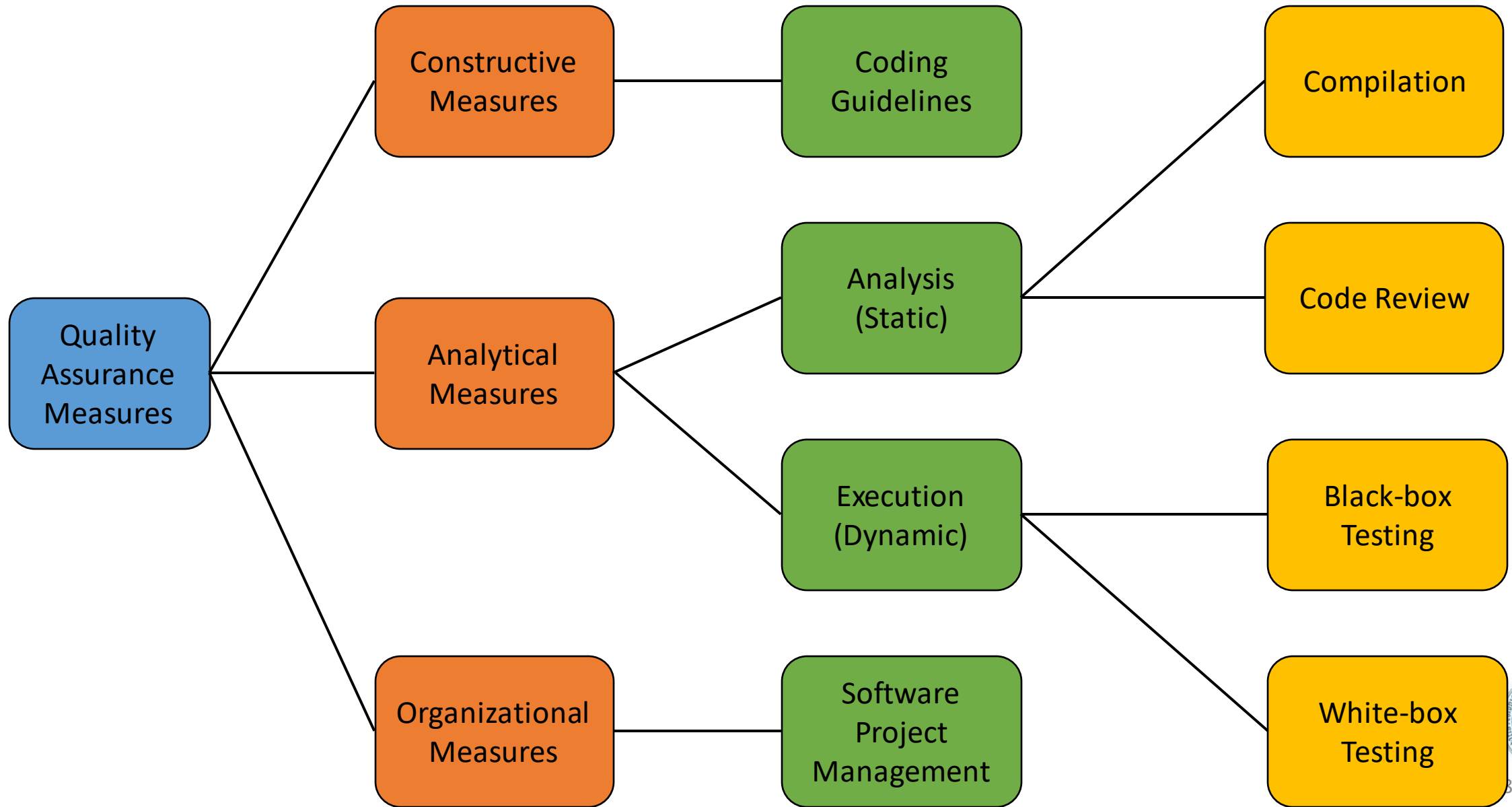
Product Quality [ISO/IEC 25010]



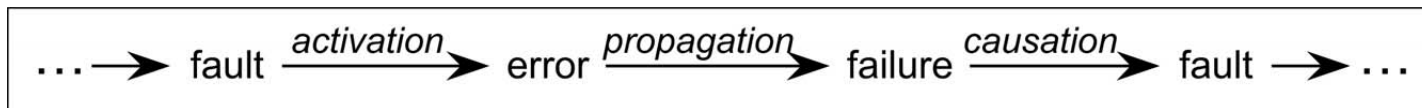
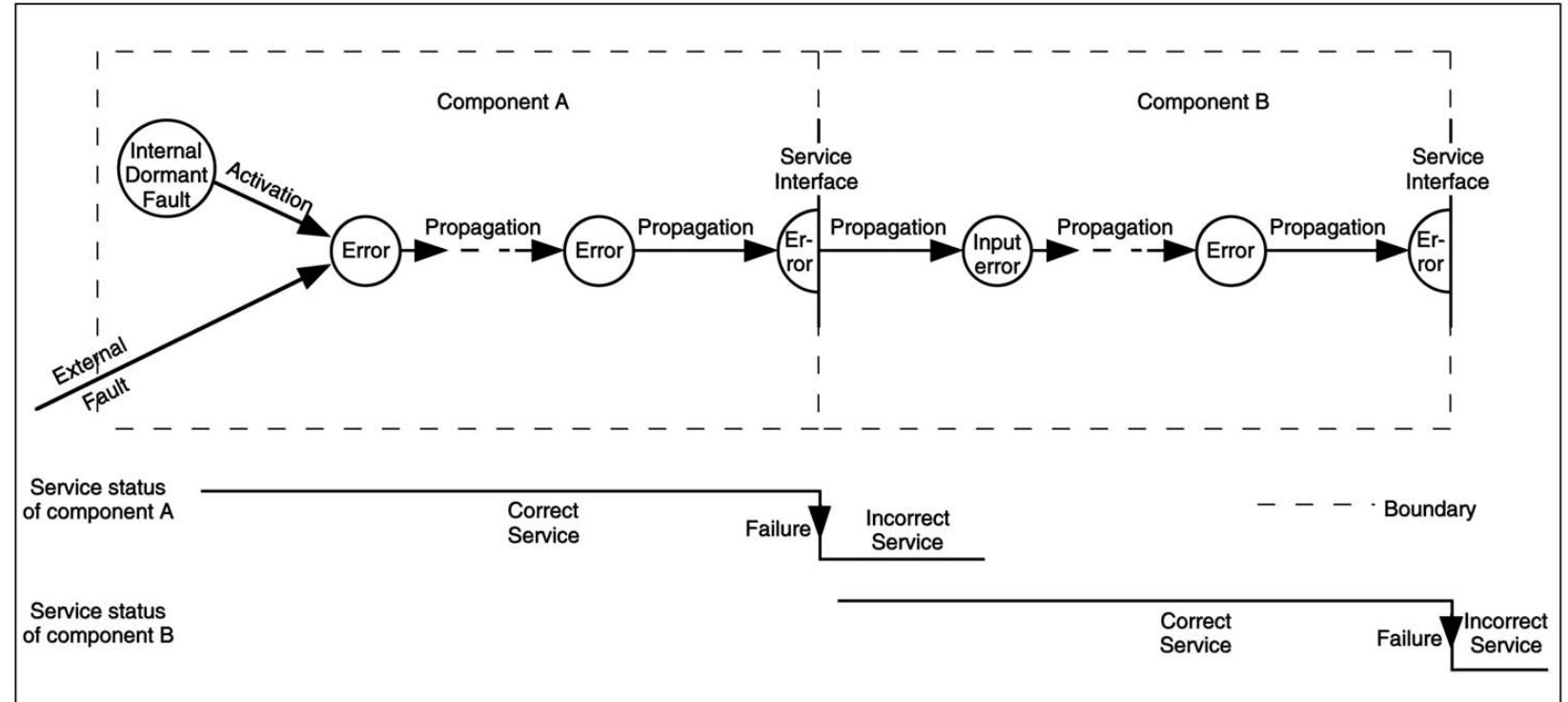
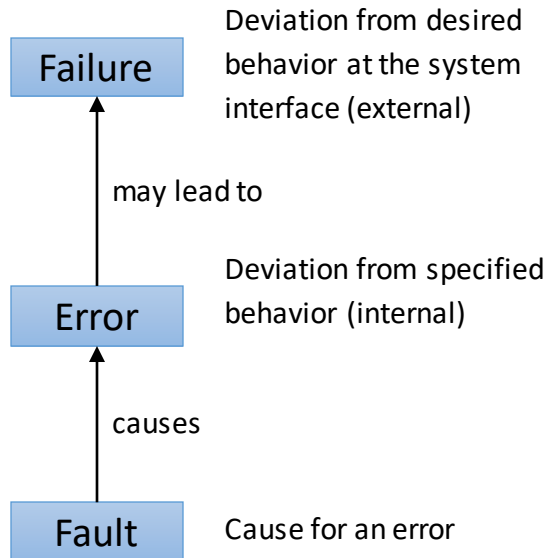
Quality in Use [ISO/IEC 25010]



Quality Assurance [Ludewig and Lichter]



Terminology: Errors, Faults, and Failures





Code Review

Code Reviews

Code Reviews

Idea: improve quality by asking other programmers for feedback. Typically applied with a quality checklist

Quality criteria: functionality, comprehensibility, maintainability, coding guidelines, design patterns, ...

Reviewer selection: based on familiarity with code, availability, expertise

Remarks

- Cannot be done by yourself
- Reviewers need programming experience and knowledge of the code (mutual feedback)
- Feedback should be timely and constructive
- Only changes reviewed, not too many

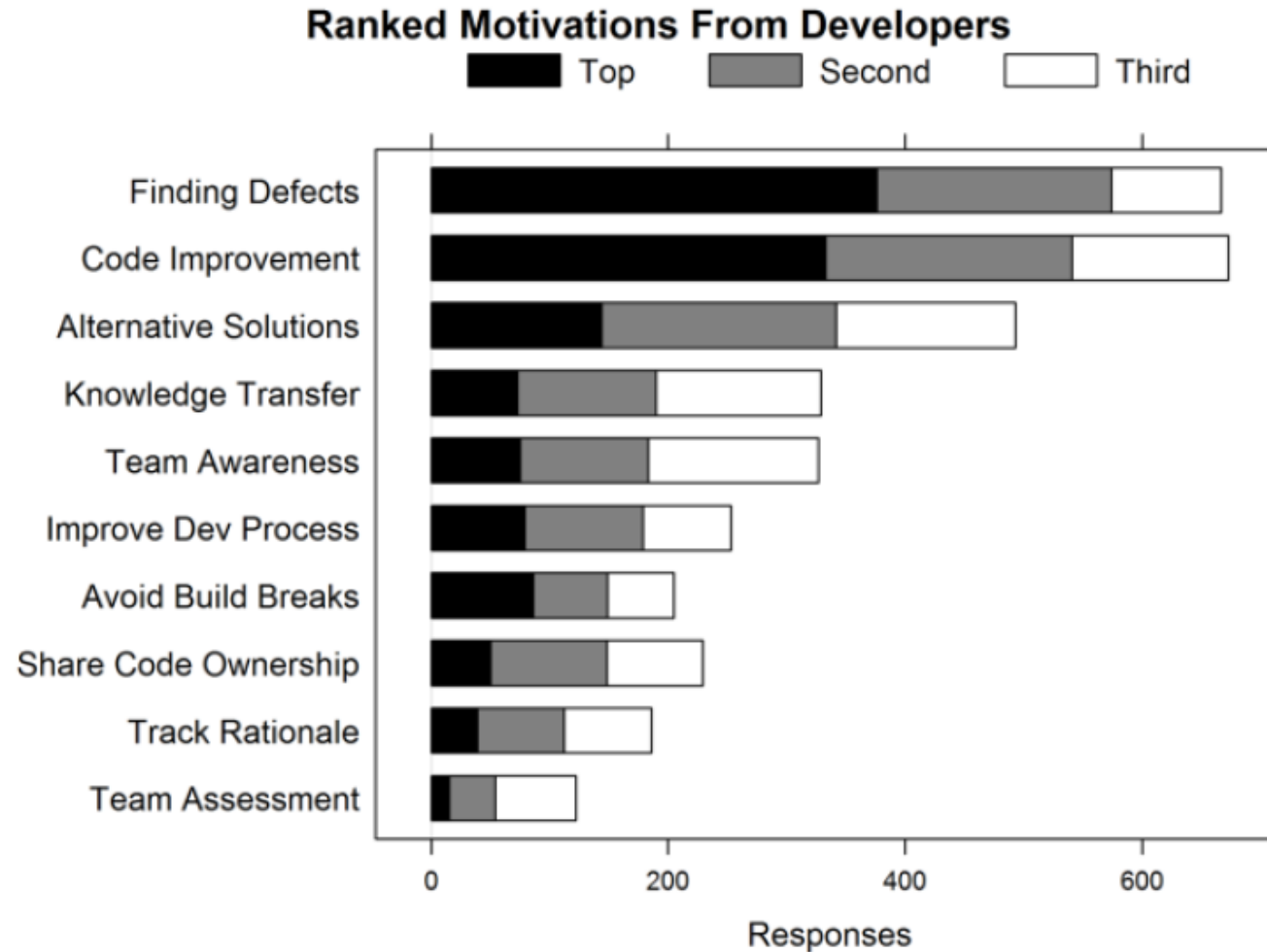
Reasons for Code Reviews

- Find bugs
 - Low-level and High-level Bugs
 - Bugs in requirements, architecture, code
 - Quality bugs (Security/Performance/...)
- Improve code
 - Readability, formatting, comments, consistency, naming
 - Enforce coding standards
- Discuss alternative solutions
- Transfer knowledge
 - Learn about API usage, available libraries, best practices, code conventions, system design, “tricks”, ...
 - “Train” developers

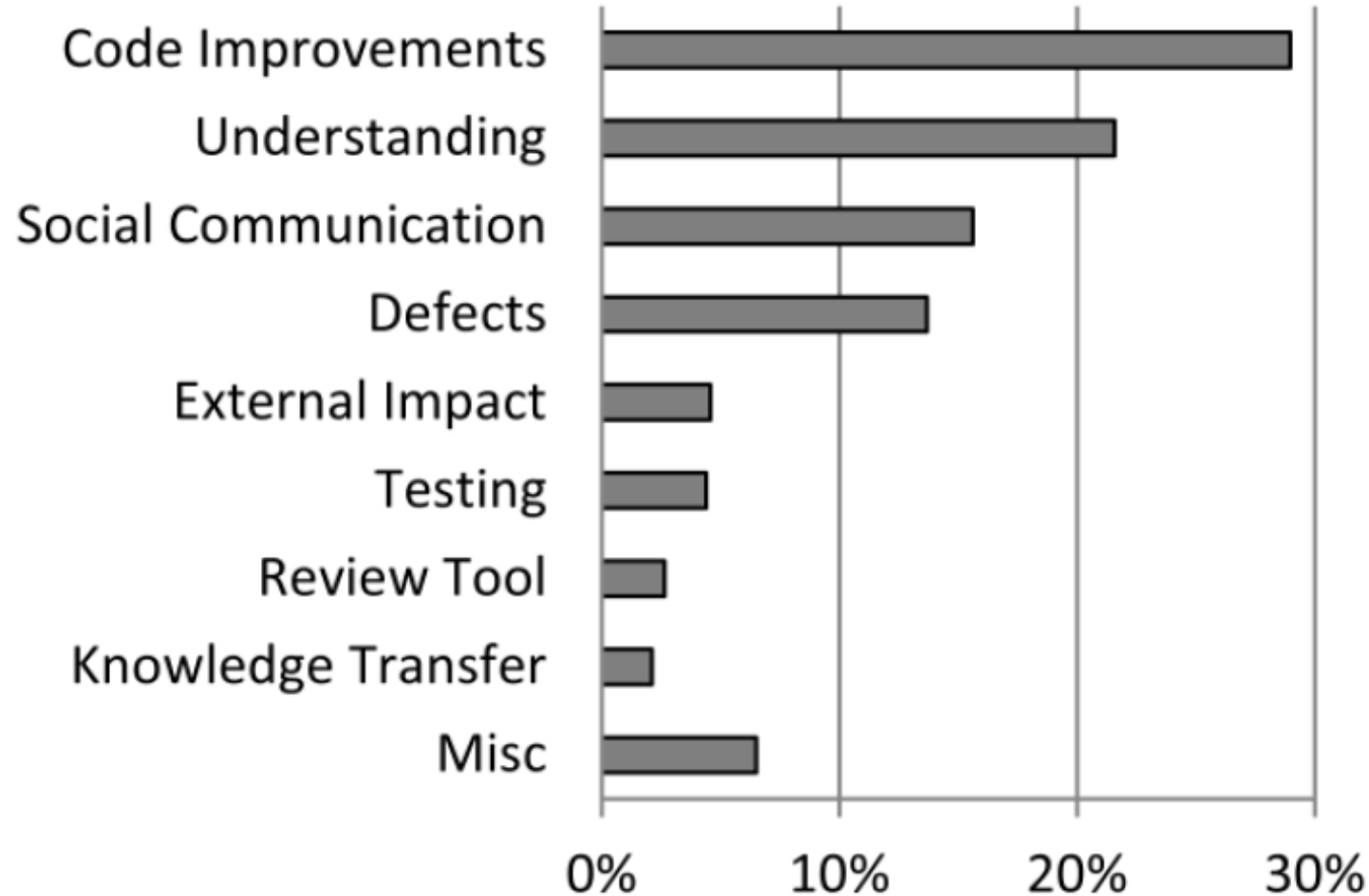
Reasons for Code Reviews (continued)

- Awareness and transparency
 - Changes are double-checked
 - Changes are announced
 - General awareness of changes and new features
- Shared code ownership
 - Common understanding of larger parts of the code
 - Openness w.r.t. feedback and change requests
 - Developers don't see their code as "their baby"

Code Reviews at Microsoft



Findings in Code Reviews at Microsoft



200 reviews with
570 findings

Different Types of Code Reviews

“Modern“ Code Review

- Lightweight, tool-based code review
- Review of every code change
- Asynchronous process
- Usually only one reviewer but open to everyone
- Part of the “definition of done”

Formal Code Inspection

- Predefined and structured process
- Review initiated based on releases
- Synchronous process (review meeting)
- Different roles: Moderator, several reviewers,...
- Explicitly planned in the project plan

Modern Code Reviews



Let's implement a new feature

OK, I think my code is ready now. @Betty: Can you have a look?

Great, thanks. Better now?



Sure. Let's see... Here are my comments.

Awesome. I'll approve the merge.

The screenshot displays the Azure DevOps Code Review interface. On the left, a code editor shows the content of `Site.css`. The CSS includes rules for validation helpers, error messages, and form elements. A comment is visible in the left margin: `Use #FF8C00 instead.`. The right-hand pane, titled "Team Explorer - Code Review", provides details about the review process. It shows the review is for "Fabrikam Fiber" and was requested by "Jamal Hartnett". A "Send Comments" button is highlighted with an orange circle. Below this, it lists two reviewers: "Johnnie McLeod" and "Raisa Pokrovskaya", both with a status of "Requested". The "Comments (2)" section shows one overall comment. The "Files" section lists the reviewed file, `Site.css`, within the path `...m Fiber/HelloWorld/HelloWorld/Content`. At the bottom of the right pane, a "Save (Ctrl+Enter) | Cancel | Line 16" bar is visible. In the background, a "New Code Review" dialog is partially visible, showing a "Submit" button highlighted with an orange circle. The University of Cologne logo is in the bottom right corner.

```
ing-top: 60px;
ing-bottom: 40px;

s for validation helpers */
validation-error {
  r: #b94a48;

validation-valid {
  lay: none;

put-validation-error {
  er: 1px solid #ddd;

pe="checkbox"].input-validat
er: 0 none;

ion-summary-errors {
  r: #b94a48;

ion-summary-valid {
  lay: none;
```

Code Review - Site.css

Team Explorer - Code Review

Code Review | Fabrikam Fiber

Hello World border color

Requested by Jamal Hartnett.

Send Comments

View Shelveset | Close Review | Actions

Reviewers (2)

Add Reviewer

Johnnie McLeod - Requested

Raisa Pokrovskaya - Requested

Related Work Items

Comments (2)

Overall (1)

Add Overall Comment

Files

...m Fiber/HelloWorld/HelloWorld/Content

Site.css

Use #FF8C00 instead.

Save (Ctrl+Enter) | Cancel | Line 16

New Code Review

Submit

Universität zu Köln

GitHub

This repository Search

[Explore](#) [Features](#) [Enterprise](#) [Blog](#)[Sign up](#)[Sign in](#) [ckaestne](#) / [TypeChef](#)

★ Star

20

🍴 Fork

12

Refactorings #28

[New issue](#)[Merged](#) joliebig merged 17 commits into [liveness](#) from [CallGraph](#) 9 months ago Conversation 3 Commits 17 Files changed 97

+1,149 -10,129



ckaestne commented on Jan 29

Owner

@joliebig

Please have a look whether you agree with these refactorings in CRewrite

key changes: Moved ASTNavigation and related classes and turned EnforceTreeHelper into an object

Labels

None yet

Milestone









No milestone

Assignee

No one assigned

2 participants

 ckaestne added some commits on Jan 29

-   remove obsolete test cases 82dddb6
-   refactoring: move AST helper classes to CRewrite package where it is ... f8fc311
-   improve readability of test code 7e61a34
-   removed unused fields ✓ f35b398



ckaestne commented on Jan 29

Owner

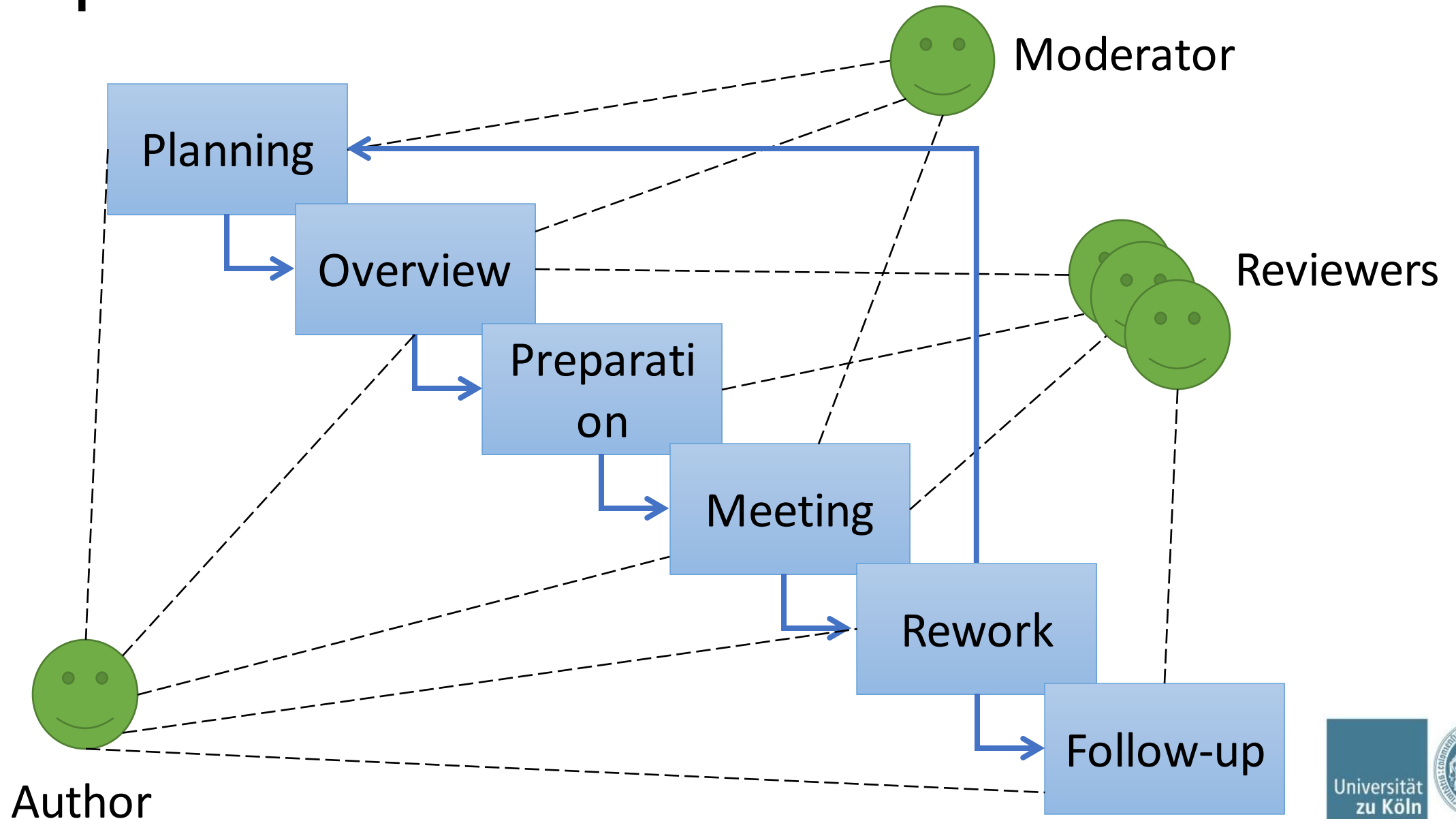
Can one of the admins verify this patch?

<https://help.github.com/articles/using-pull-requests/>

Formal Inspection (aka Fagan Inspection)

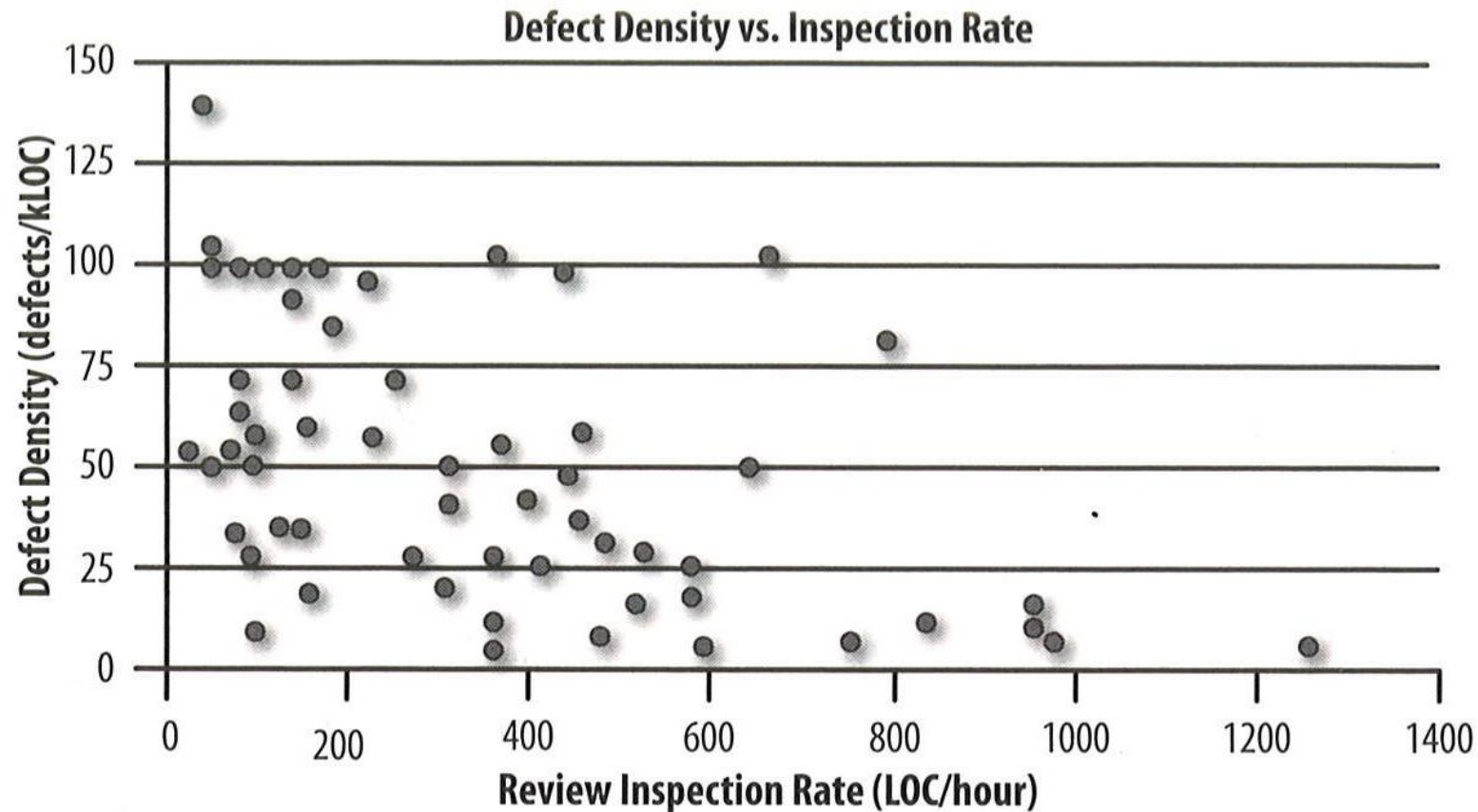
- Idea popularized in 70s at IBM
- Broadly adopted in 80s, much research
 - Sometimes replacing component testing
- Group of developers meet to formally review code or other artifacts
- Most effective approach to find bugs
 - Typically, 60-90% of bugs found with inspections
- Expensive and labor-intensive

Inspection Process



Inspection Speed

- More than 400 LOC/h make reviews shallow
- Recommendation: <400 LOC for a 1h inspection meeting.



Code Review Checklists

Implementation

- Does this code change do what it is supposed to do?
- Can this solution be simplified?
- Does this change add unwanted compile-time or run-time dependencies?
- Was a framework, API, library, service used that should not be used?
- Was a framework, API, library, service not used that could improve the solution?
- Is the code at the right abstraction level?
- Is the code modular enough?
- Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?
- Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?
- Are there any best practices, design patterns or language-specific patterns that could substantially improve this code?
- Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-close Principle, Liskov Substitution Principle, Interface Segregation, Dependency Injection?

Dependencies

- If this change requires updates outside of the code, like updating the documentation, configuration, readme files, was this done?
- Might this change have any ramifications for other parts of the system, backward compatibility?

Security and Data Privacy

- Does this code open the software for security vulnerabilities?
- Are authorization and authentication handled in the right way?
- Is sensitive data like user data, credit card information securely handled and stored?
- Is the right encryption used?
- Does this code change reveal some secret information like keys, passwords, or usernames?
- If code deals with user input, does it address security vulnerabilities such as cross-site scripting, SQL injection, does it do input sanitization and validation? Is data retrieved from external APIs or libraries checked accordingly?

Logic Errors and Bugs

- Can you think of any use case in which the code does not behave as intended?
- Can you think of any inputs or external events that could break the code?

Error Handling and Logging

- Is error handling done the correct way?
- Should any logging or debugging information be added or removed?
- Are error messages user-friendly?
- Are there enough log events and are they written in a way that allows for easy debugging?

Usability and Accessibility

- Is the proposed solution well designed from a usability perspective?
- Is the API well documented?
- Is the proposed solution (UI) accessible?
- Is the API/UI intuitive to use?

Testing and Testability

- Is the code testable?
- Does it have enough automated tests (unit/integration/system tests)?
- Do the existing tests reasonably cover the code change?
- Are there some test cases, input or edge cases that should be tested in addition?

Performance

- Do you think this code change will impact system performance in a negative way?
- Do you see any potential to improve the performance of the code?

Readability

- Was the code easy to understand?
- Which parts were confusing to you and why?
- Can the readability of the code be improved by smaller methods?
- Can the readability of the code be improved by different function/method or variable names?
- Is the code located in the right file/folder/package?
- Do you think certain methods should be restructured to have a more intuitive control flow?
- Is the data flow understandable?
- Are there redundant comments?
- Could some comments convey the message better?
- Would more comments make the code more understandable?
- Could some comments be removed by making the code itself more readable?
- Is there any commented out code?

Experts Opinion

- Do you think a specific expert, like a security expert or a usability expert, should look over the code before it can be committed?
- Will this code change impact different teams? Should they have a say on the change as well?

Find more at michaelgreiler.com

- Code Review Best Practices
- Code Review Pitfalls
- Code Reviews at Microsoft
- Code Reviews at Google

Social Aspects of Code Reviews

- Be aware: Author's self-worth is probably in the code
- Do not criticize authors
 - “you didn't initialize variable a” →
“I don't see where variable a is initialized”
- Meetings should not include management (it's a **peer** review)
- Do not use review results for HR evaluation
 - “finding more than 5 bugs during inspection counts against the author”
 - Leads to avoidance, fragmented submission, not pointing out defects, holding pre-reviews

Dos and Don'ts in Code Review

As a review receiver (author)

- Don't rely solely on reviewers; review your code first (incl. testing)
- Don't take feedback personally, do embrace feedback
- Don't “defend” your code
- Do address all comments
- Do decide how to resolve faults

Dos and Don'ts in Code Review

As a reviewer

- First of all, be polite and constructive
- Do get an understanding about the code's context
 - what is it about?
 - what belongs to the code under review (code, tests, configurations, documentation)?
 - is there a related issue/specification/discussion?
- Do try to run the code; don't just look at the code without testing it.
- Don't discuss style if there are no guidelines
- Do highlight the good parts of the code as well, not only the mistakes
- Don't blindly accept code from more experienced developers; do ask for clarification if you don't understand things
- Don't “show off” that you are better/smarter
- Do provide the review quick



Software Testing

Software Testing

Software Testing [Sommerville]

Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.

Validation Testing [Sommerville]

Demonstrate to the developer and the customer that the software meets its requirements.

Defect Testing (Verification) [Sommerville]

Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specification.

V&V [Boehm1979]

Validation: Are we building the right product?

Verification: Are we building the product right?

Kinds of Software Testing

Stages of Testing [Sommerville]

Development testing, where the system is tested during development to discover bugs and defects

Release testing, where a separate testing team tests a complete version of the system before it is released to users

User testing, where users or potential users of a system test the system in their own environment

Testing Levels

Unit testing: test specific lines of code (a unit)

Integration testing: test integrated components

System testing: test entire system

Acceptance testing: check compliance with business goals

Manual vs. automated testing [Sommerville]

In **manual testing**, a tester runs the program with some test data and compares the results to their expectations. [...]

In **automated testing**, the tests are encoded in a program that is run each time the system under development is to be tested.

Test Cases (Testfälle)

Systematic Test [Ludewig and Lichter]

A systematic test is a test, in which

1. the setup is defined,
2. the inputs are chosen systematically,
3. the results are documented and evaluated by criteria being defined prior to the test.

Test Case [Ludewig and Lichter]

In a test, a number of test cases are executed, whereas each test case consists **input values** for a single execution and **expected outputs**. An **exhaustive test** is a test in which the test cases exercise all the possible inputs.

Exhaustive Testing in Practice?

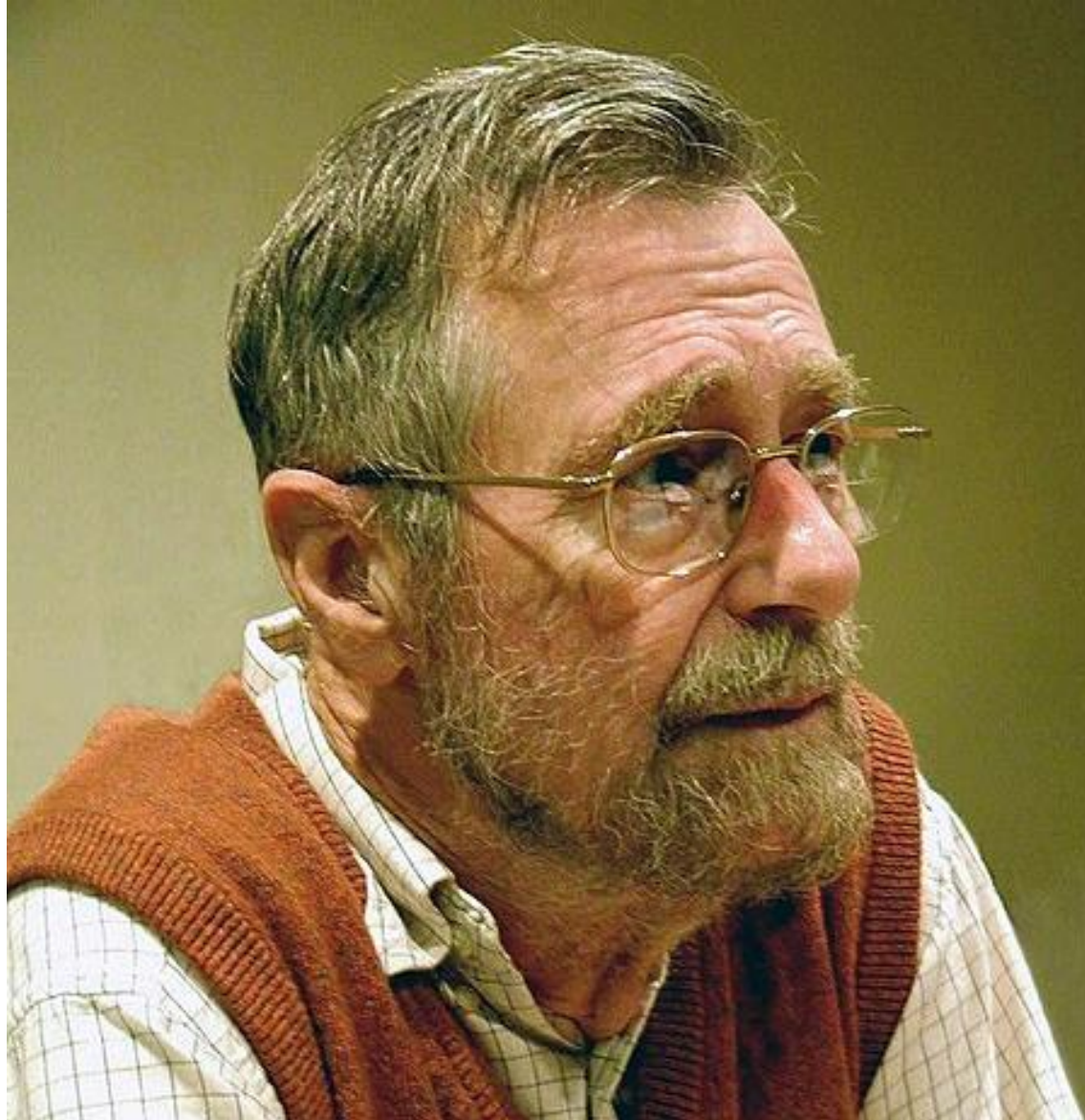
```
boolean a, b, c;
int i, j;
```

bla(a,b,c) has $2^3 = 8$ possible inputs

blub(i,j) has $(2^{32})^2 = 2^{64} \approx 10^{19}$ inputs

- assuming 10^9 test cases can be executed in 1 second (cf. CPU with more than 1 GHz)
- exhaustive test of blub takes ≈ 585 years
- testing for a day would cover less than 0.0005 % of the inputs

How to test thousands of such methods several times a day?



Edsger W. Dijkstra (1972)

“Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.”

Test Case Design (Testfallentwurf)

Goal [Ludewig and Lichter]

Detect a large number of failures with a low number of test cases.

An ideal test case is... [Ludewig and Lichter]

- **representative**: represents a large number of feasible test cases
- **failure sensitive**: has a high probability to detect a failure
- **non-redundant**: does not check what other test cases already check

