

Teil 3.2: Software Qualität

Falk Howar

Softwarekonstruktion WS 2018/19

LS14

1 Motivation

2 Software Qualität

3 Code Qualität

4 Coding Standards

5 Code Reviews

6 Code Metriken

Qualität

The bitterness of poor quality remains long
after the sweetness of low price is forgotten

Benjamin Franklin

Motivation

Software ist omnipräsent in modernen Geräten!

Bugs auch ...

Bugs im Betrieb können wirklich teuer werden!

⇒ Methoden, um Fehler zu finden und zu verhindern

Klassisches Beispiel: Ariane 5

- Junfernflug 4. Juni, 1996
- Explodiert 40 Sekunden nach Start
- Lange teuerster Software Fehler : 500M USD



Fehlerhafte Integration von (unnötigem) Legacy Code:

- Unbehandelte Exception in floating-point Umwandlung
- Umwandlung von a 64-bit int to a 16-bit signed integer
- Funktioniert für Zahlen kleiner oder gleich 2^{15}
- Größere Zahl \Rightarrow Exception ...

Aktuellstes Beispiel

The screenshot shows a news article from The Register. The header features the site's logo 'The Register' with the tagline 'Biting the hand that feeds IT'. Below the header is a navigation bar with links: DATA CENTRE, SOFTWARE, SECURITY, TRANSFORMATION, DEVOPS, BUSINESS, and PERSONAL 1. The main title of the article is 'Pro tip: You can log into macOS High Sierra as root with no password'. A subtext below the title reads 'Apple, this is Windows 95 bad – but there is a workaround to kill the bug'. The author is listed as 'By Shaun Nichols in San Francisco 28 Nov 2017 at 20:15' with 140 comments and a share button. The article includes a photograph of two young children, a boy and a girl, looking excitedly at a laptop screen.

Updated A trivial-to-exploit flaw in macOS High Sierra, aka macOS 10.13, allows users to gain admin rights, or log in as root, without a password.

- Anmeldung als 'root' ohne Passwort nach Update
- Alle Administrator Rechte
- November 2017
- Auch aktuelles Update löst Problem nicht ganz

Qualitätssicherung

- Was ist Qualität von Software?
- Wie kann Qualität sichergestellt werden?

- Warum, Wer, Wann, Was, Wie?

Software Qualität

Software Qualität

Quality

The standard of something as measured against other things of a similar kind; the degree of excellence of something.

<http://www.google.com>

Software Qualität

Quality

The standard of something as measured against other things of a similar kind; the degree of excellence of something.

... the standard of something ...
... degree of excellence ...



Sehr unspezifisch:
Exzellenz in was?

Man kann Qualität messen!
Im Vergleich?

<http://www.google.com>

Aus Beispielen (Vorlesung I)

Wir wollen, dass unser Produkt ...

Aus Beispielen (Vorlesung I)

Wir wollen, dass unser Produkt ...

- ... nicht explodiert,
- ... Menschen nicht gefährdet,
- ... nicht ausfällt,
- ... Erwartungen der Benutzer erfüllt,
- ... kein Sicherheitsrisiko darstellt.

Software Qualität (zweiter Versuch)

Quality

The measurable standard of something **in some concrete respect**; the degree of excellence of something **in some concrete respect**.

Software Qualität (zweiter Versuch)

Quality

The measurable standard of something **in some concrete respect**; the degree of excellence of something **in some concrete respect**.

- Wie oft es explodiert,
- Verursachte Todesfälle,
- Anzahl Ausfälle,
- ...

Software Qualität (zweiter Versuch)

Quality

The **measurable** standard of something **in some concrete respect**; the degree of excellence of something **in some concrete respect**.

- Wie oft es explodiert,
- Verursachte Todesfälle,
- Anzahl Ausfälle,
- ...

Wir akzeptieren eine bestimmte Häufigkeit von Problemen
Oft ist die akzeptierte Häufigkeit 0.

Technische Definition

Quality (P. B. Crosby)

Conformance to requirements. The requirements may not fully represent customer expectations.

- Qualität relativ zu Anforderungen,
- Messbar als Conformance (Erfüllen von Anforderungen),
- Anforderungen drücken Erwartungen aus

Faults and Failures

Ausfall

Wenn eine Anforderung nicht (mehr) erfüllt wird.

Der Moment in dem das System explodiert ...

Faults and Failures

Ausfall

Wenn eine Anforderung nicht (mehr) erfüllt wird.

Der Moment in dem das System explodiert ...

Fehler

Ursache für einen Ausfall

Exception, die geworfen wurde bei 64-bit float nach 16-bit Umwandlung ...

Qualitätssicherung

Qualitätssicherung

Verhinderung von Ausfällen in Produkten oder Services (durch das Verhindern oder Finden von Fehlern).

Qualitätssicherung

Qualitätssicherung

Verhinderung von Ausfällen in Produkten oder Services (durch das Verhindern oder Finden von Fehlern).

- Nur Ausfälle können beobachtet werden
- Fehler finden ist oft schwierig

Qualitätssicherung

Qualitätssicherung

Verhinderung von Ausfällen in Produkten oder Services (durch das Verhindern oder Finden von Fehlern).

- Nur Ausfälle können beobachtet werden
 - Fehler finden ist oft schwierig
-
- Verschiedene Strategien: Ausfälle durch Testen finden, Fehler verhindern, ... \

Warum, Wer, Wann, Was, Wie?

Warum

Weil wir gute Menschen sind?

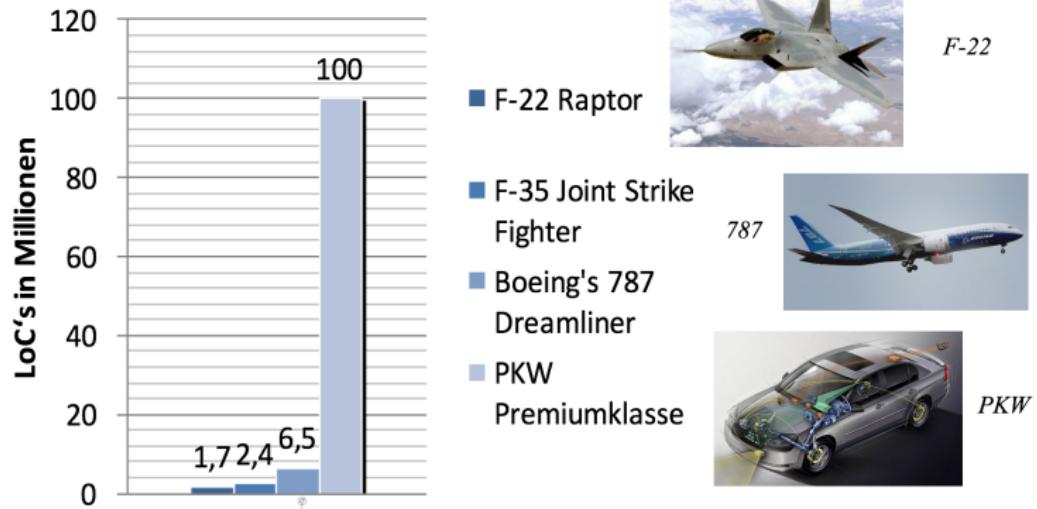
Warum

Weil wir gute Menschen sind?

- Um Kunden zufriedenzustellen (damit sie Kunden bleiben)
- Produkthaftung (damit wir nicht verklagt werden)
- Gesetzlich verlangt für bestimmte Produkte (Autos, Flugzeuge, etc.)
- ...

Wer?

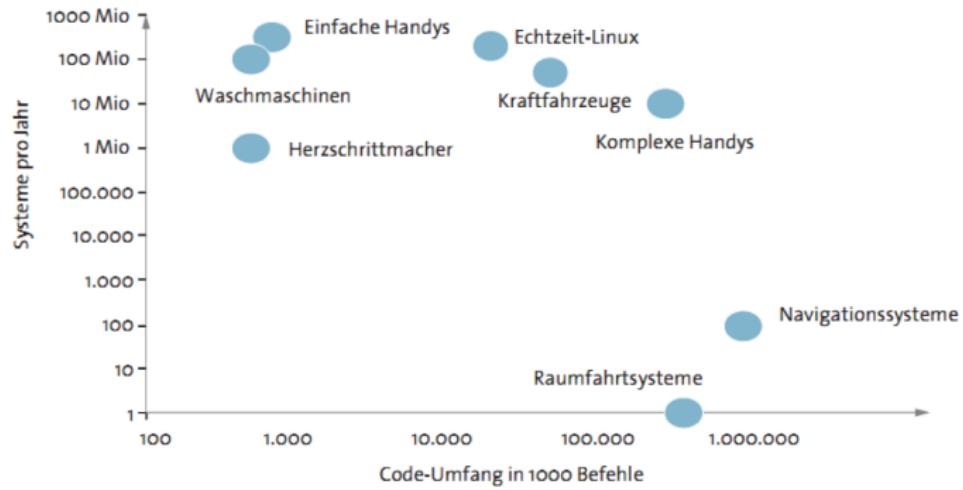
Die offensichtlichen Kandidaten: Sicherheitskritische Systeme.



Wer (II)

Zwei Gruppen:

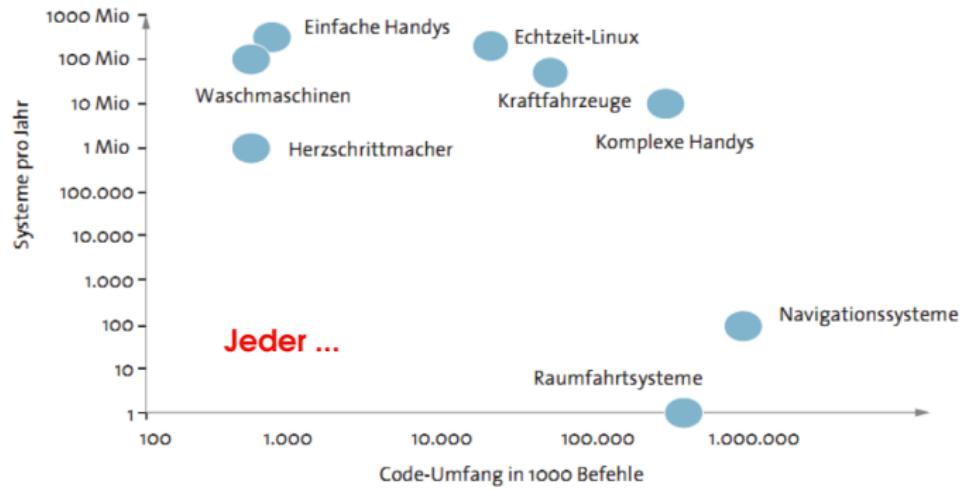
Sicherheitskritische, teure Systeme und Systeme mit vielen Installationen.



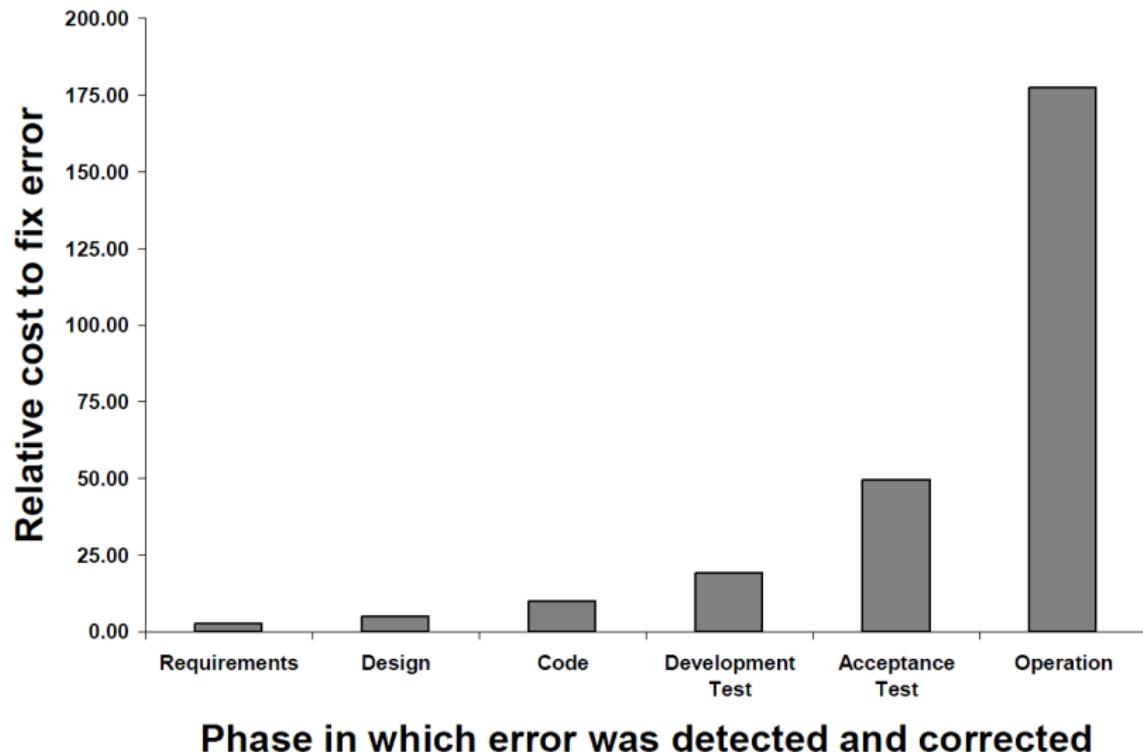
Wer (II)

Zwei Gruppen:

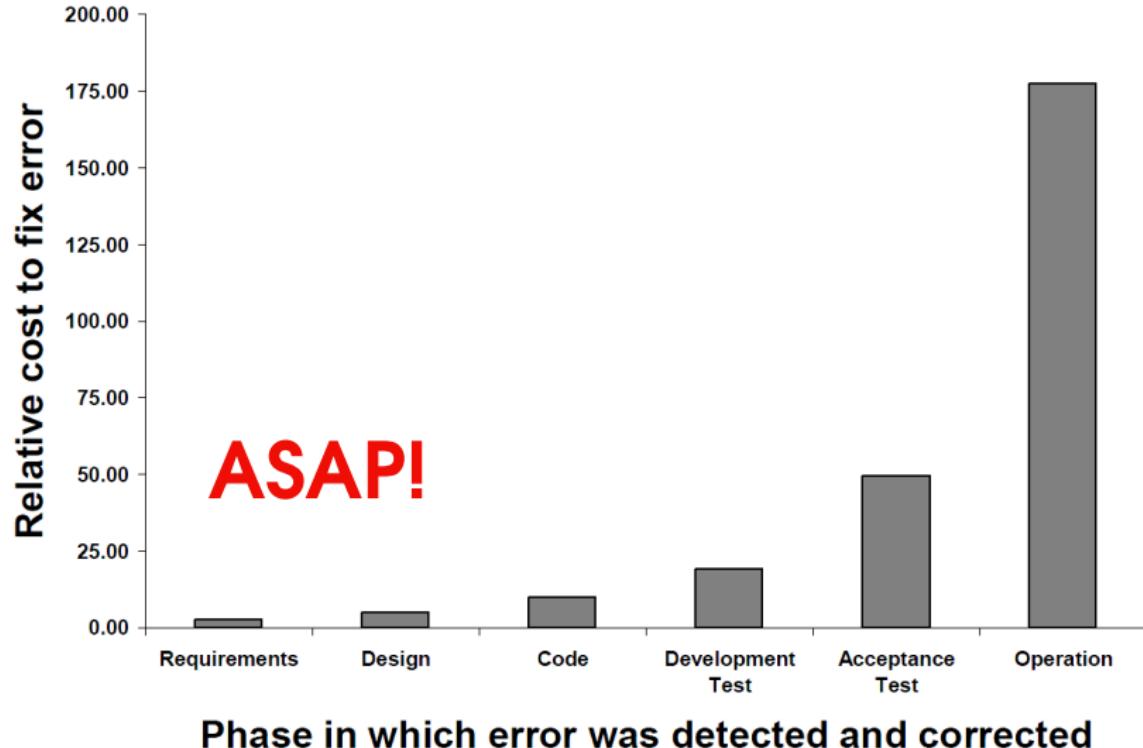
Sicherheitskritische, teure Systeme und Systeme mit vielen Installationen.



Wann?



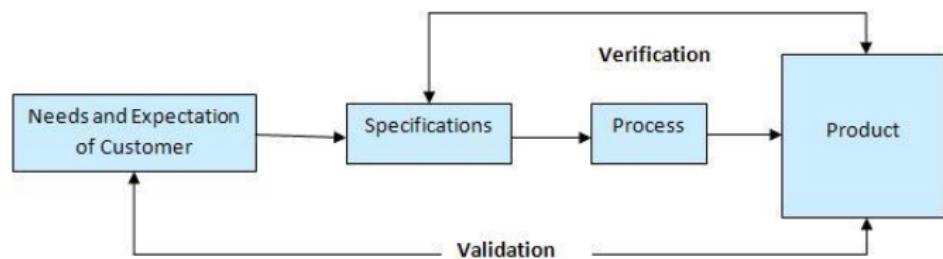
Wann?



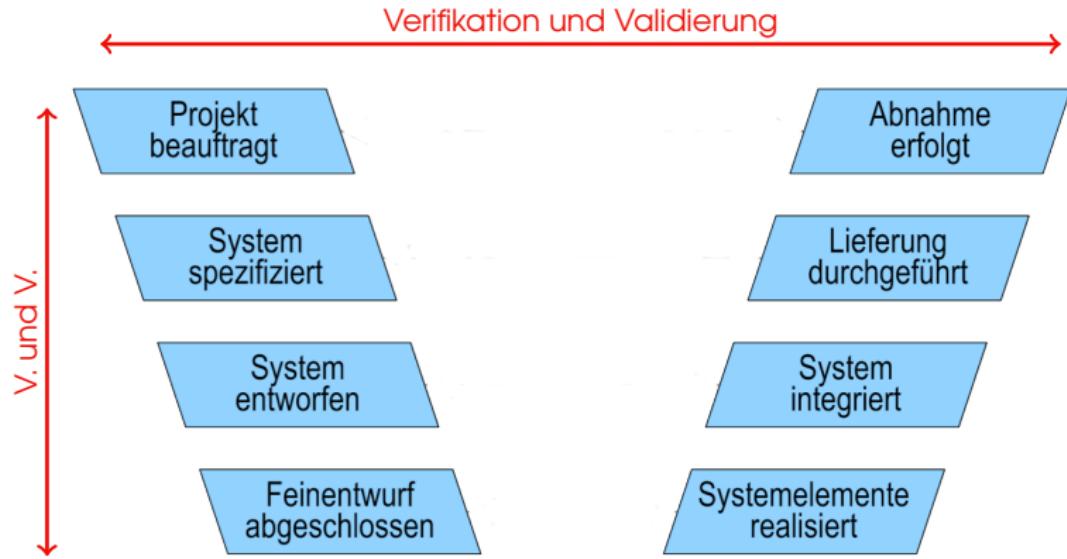
Was?

Verifikation: Bauen wir das Produkt richtig?

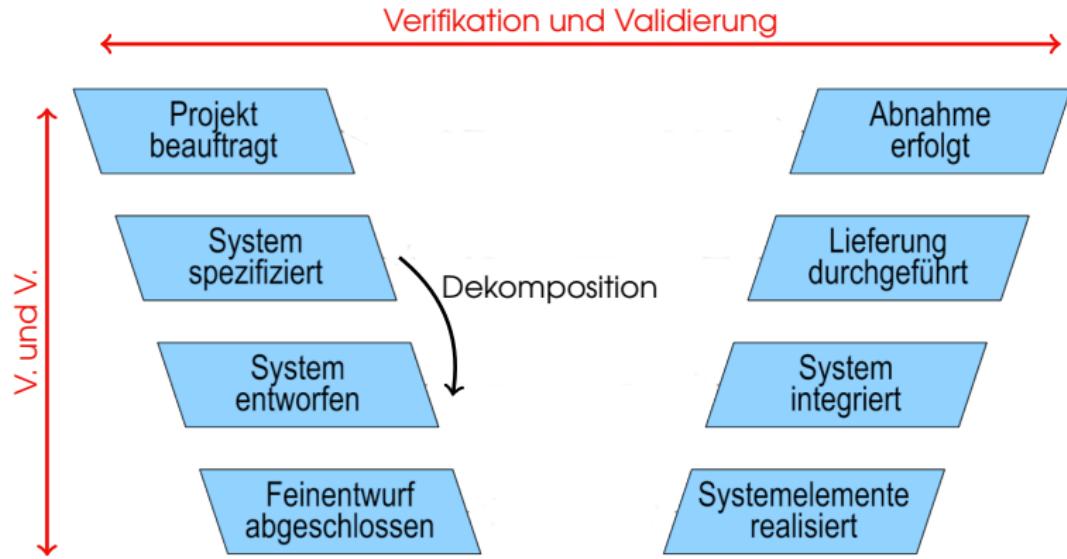
Validierung: Bauen wir das richtige Produkt?



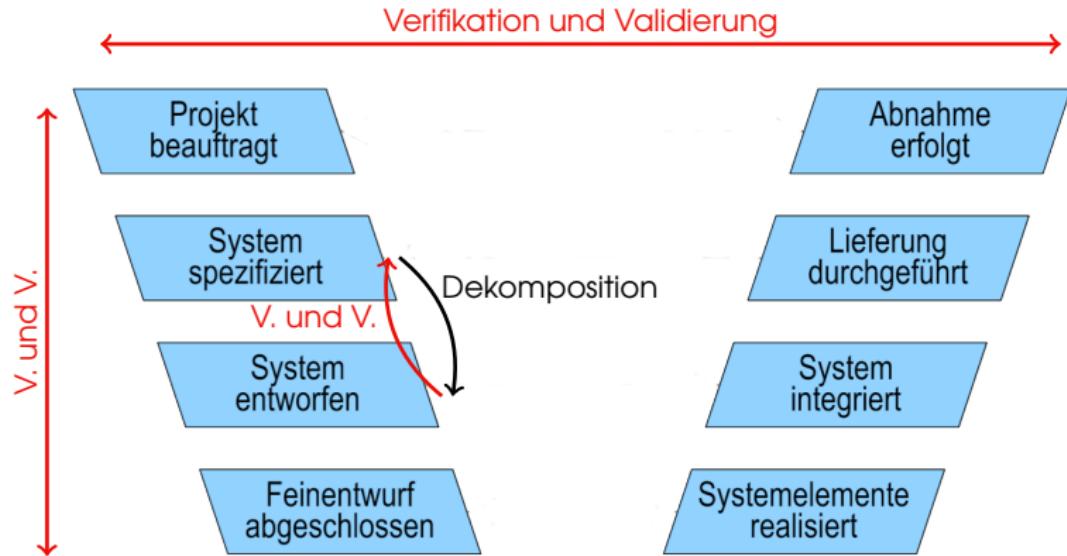
V & V im Entwicklungsprozess



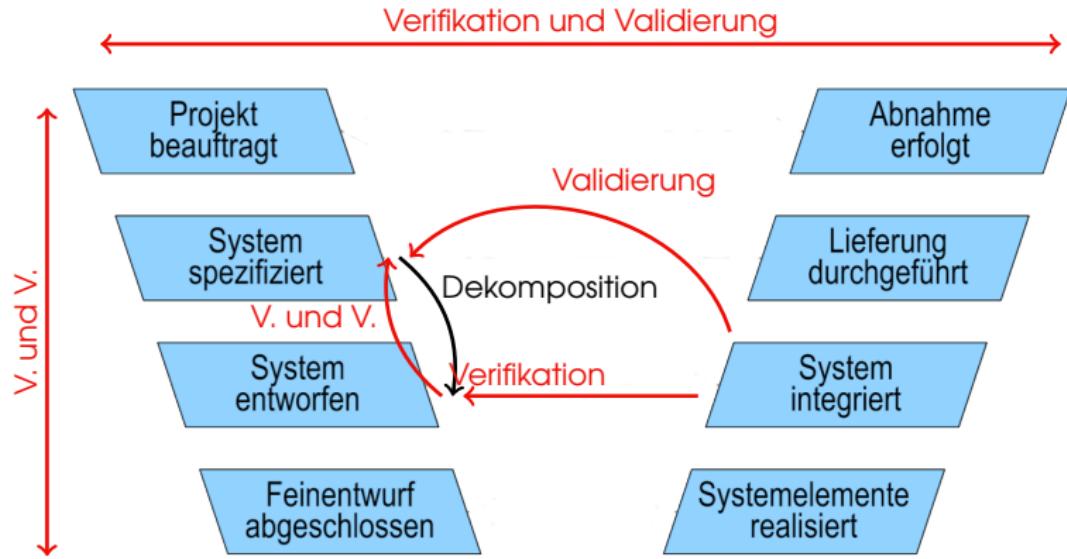
V & V im Entwicklungsprozess



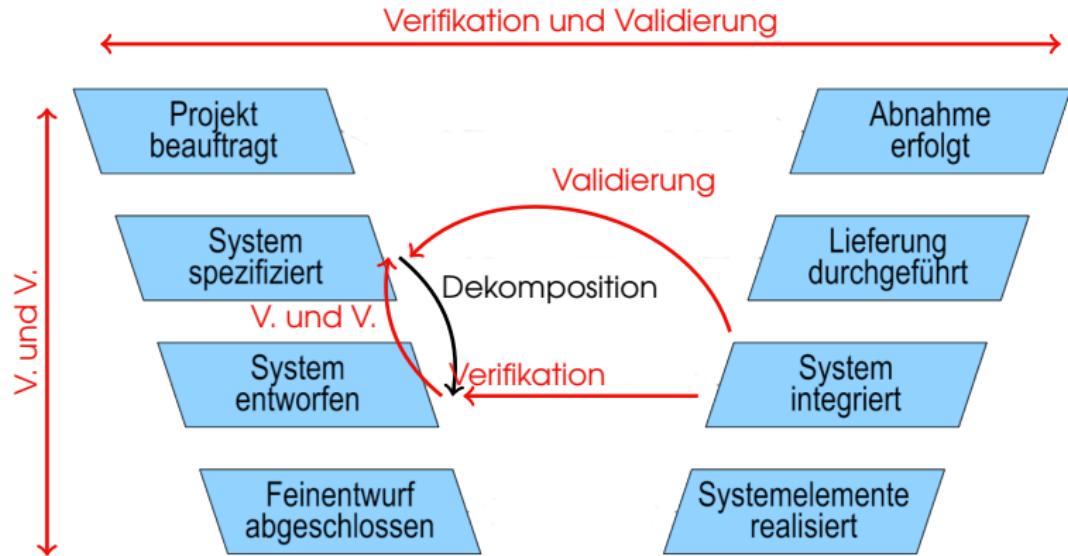
V & V im Entwicklungsprozess



V & V im Entwicklungsprozess



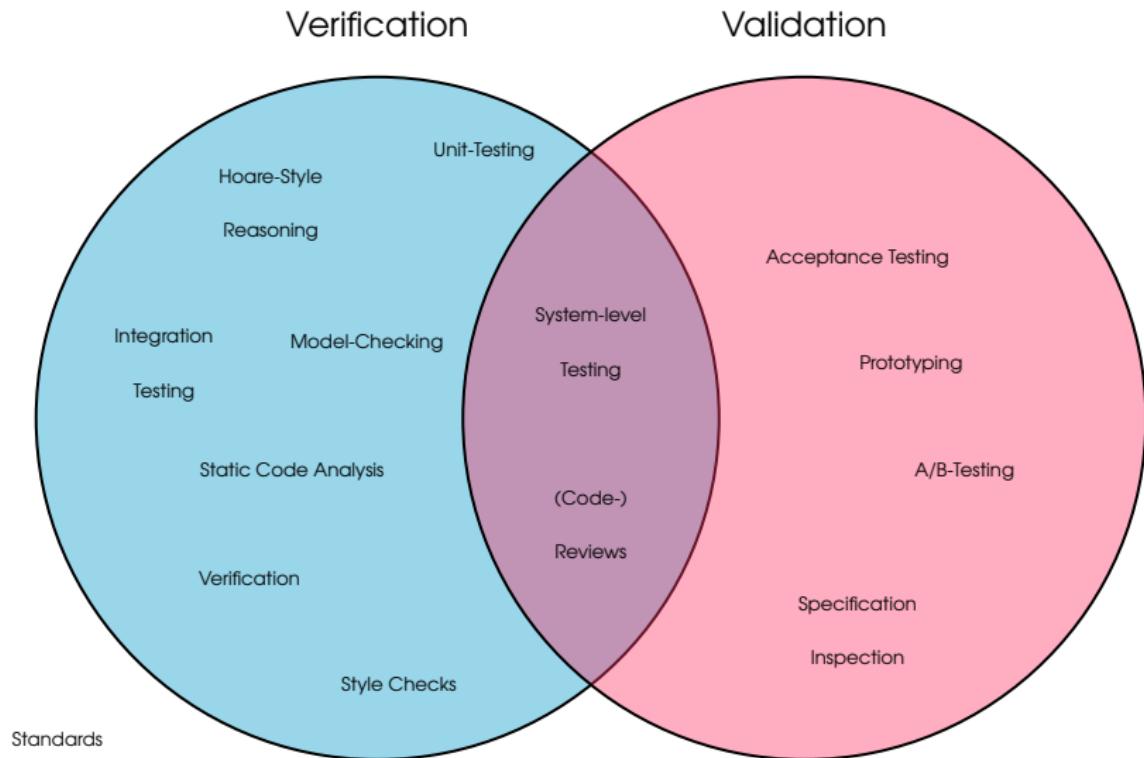
V & V im Entwicklungsprozess



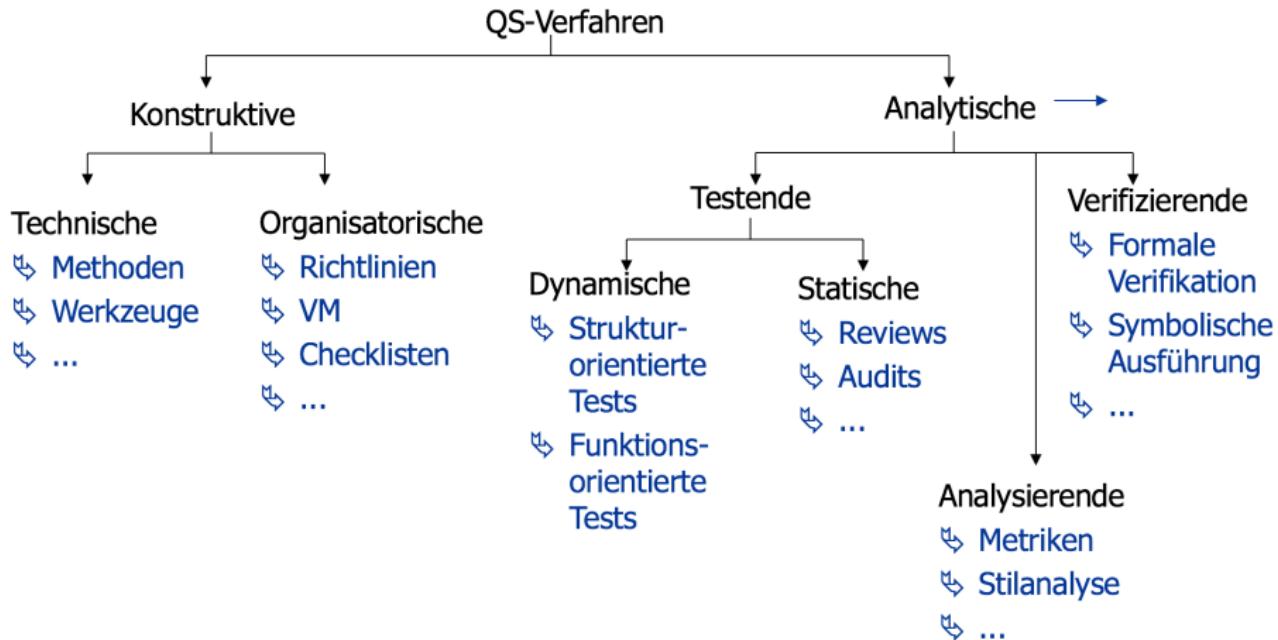
⇒ Wann? In jedem Prozessschritt!

Jedes Artefakt des Entwicklungsprozesses kann verifiziert und validiert werden.

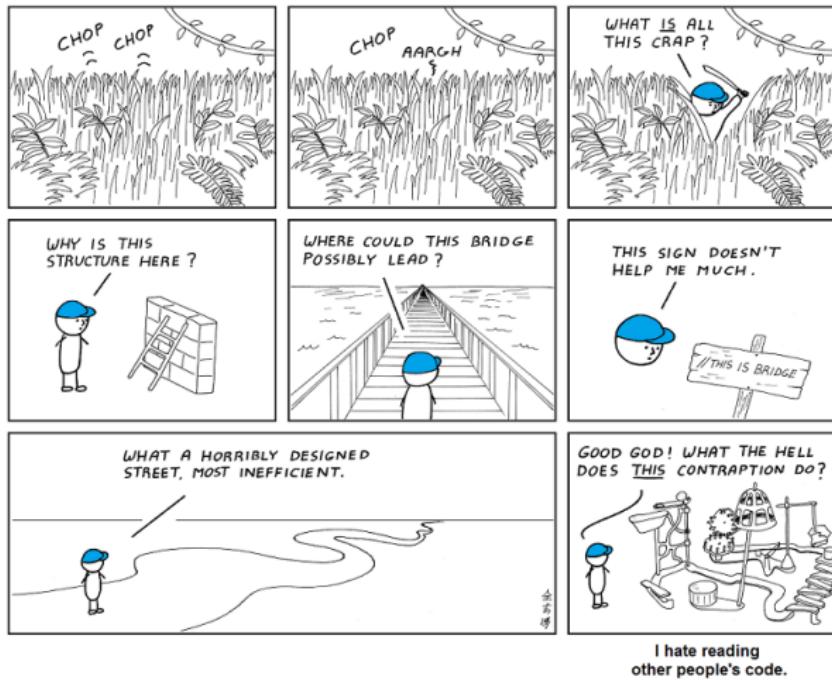
Wie? - Verifikation vs. Validierung



Wie? - Klassifizierung von Ansätzen



Code Qualität



Code Qualität: Motivation

In großen Software Projekten ...

- ... Entwickler verteilt über **verschiedene Standorte**,
- ... Viele verschiedene Entwickler, **geteilter Code** (über Zeit bzw. nebeneinander),
- ... **Code aus Jahrzehnten**, auch wenn Entwickler wechseln.

⇒ Entwickler müssen in der Lage sein
den Code von anderen zu lesen und zu verstehen

Code Qualität: Definition

Code Quality or Software Structural Quality

How well a software meets non-functional requirements that support the delivery of its functional requirements:

- Reliability / Resilience: How well the software deals with faults.
- Performance / Efficiency: How efficiently the software performs.
- Security: How secure a software is for a user.
- Maintainability: How easy/hard it is to maintain the software.

(Zuverlässigkeit / Fehlertoleranz, Performanz / Effizienz, Sicherheit, Wartbarkeit)

Im Gegensatz zu: funktionaler Qualität die beurteilt, wie gut Software funktionale Anforderungen erfüllt.

Code Qualität: Agenda

Ansätze zur Sicherstellung von Code Qualität:

- (1) **Coding Standards** stellen sicher das Code wartbar / lesbar ist und ggf. andere nicht-funktionale Anforderungen erfüllt,
- (2) **Code Reviews** stellen gemeinsames Verständnis sicher und finden Fehler möglichst früh,
- (3) **Code Metrics** helfen bei der Bewertung der Qualität von Code.

Coding Standards



The Ten Commandments, George Bannister, <https://www.flickr.com/>

Coding Style Examples

Some Examples

And better alternatives

```
if (foo)
    // bar();
doSomethingElse();
```

Coding Style Examples

Some Examples

```
if (foo)
    // bar();
doSomethingElse();
```

And better alternatives

```
if (foo) {
    // bar();
}
```

Coding Style Examples

Some Examples

```
if (foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

And better alternatives

```
if (foo) {
    // bar();
}
```

Coding Style Examples

Some Examples

```
if(foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

And better alternatives

```
if(foo) {
    // bar();
}
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++) {
        doThis();
    }
    doThat();
}
```

Coding Style Examples

Some Examples

```
if(foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

```
boolean b = x.equals("");
```

And better alternatives

```
if(foo) {
    // bar();
}
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++) {
        doThis();
    }
    doThat();
}
```

Coding Style Examples

Some Examples

```
if(foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

```
boolean b = x.equals("");
```

And better alternatives

```
if(foo) {
    // bar();
}
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++) {
        doThis();
    }
    doThat();
}
```

```
boolean b = "".equals(x);
```

Coding Style Examples

Some Examples

```
if(foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

```
boolean b = x.equals("");
```

```
int doIt(int x) {...}
```

And better alternatives

```
if(foo) {
    // bar();
}
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++) {
        doThis();
    }
    doThat();
}
```

```
boolean b = "".equals(x);
```

Coding Style Examples

Some Examples

```
if(foo)
    // bar();
doSomethingElse();
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++)
        doThis();
        doThat();
}
```

```
boolean b = x.equals("");
```

```
int doIt(int x) {...}
```

And better alternatives

```
if(foo) {
    // bar();
}
```

```
for (int i = 1; i < a; i++) {
    for (int j = 1; j < b; j++) {
        doThis();
    }
    doThat();
}
```

```
boolean b = "".equals(x);
```

```
/**
 * computer a random prime
 * number between
 * 0 and upperBound.
 */
int findPrime(int upperBound) {...}
```

Coding Standards: Definition

Coding Standards

A coding standard is a set of rules prescribing a certain programming style. Rules usually cover:

Indentation, Organization, Declaration, Error Handling, Conditional Statements,
Test Coverage ...

Example: All Variables shall be declared at the top of a class.

Coding Standards: Motivation

Code standards are useful for a number of reasons:

- Maintainability: Code standards can improve the readability of code,
- Security: Code standards can help enforce security requirements,
- Reliability: Code standards can limit the use of potentially “dangerous” practices.
- Efficiency: Code standards can help enforcing efficient code.

Coding Standards: Examples

Maintainability

- Documentation required for all public methods.
- Placement of braces: 1TBS.

Security

- SQL Statements are not built from Strings but using library ...
- Only hashes are stored for passwords.
- All code needs to be tested.

Reliability

- All exceptions must be caught and handled.
- All parameters of methods must be validated.

Efficiency

- Use StringBuilder instead of repeated concatenation.

Coding Standards: Examples

Maintainability

- Documentation required for all public methods.
- Placer **Bad Example!**
How much?, When?, ...

Security

- SQL Statements are not built from Strings but using library ...
- Only hashes are stored for passwords.
- All code needs to be tested.

Reliability

- All exceptions must be caught and handled.
- All parameters of methods must be validated.

Efficiency

- Use StringBuilder instead of repeated concatenation.

Coding Standards: Discussion

Some Good Practices:

- Rules have to be specific: E.g., what does “Don’t use Hacks” mean?
- Less is more: Nobody will remember 50 rules,
- Enforcement is essential,
- Tool support is valuable (\Rightarrow static code analysis).

Example: NASA JPL

Rules used at Jet Propulsion Laboratory's (JPL) Laboratory for reliable software (specifically written for C).

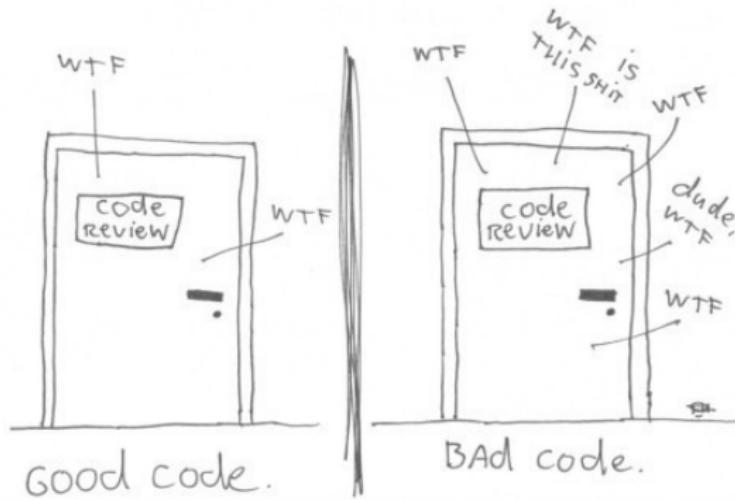
- (1) Restrict all code to very simple control flow constructs – do not use goto ...
- (2) All loops must have a fixed upper-bound. It must be trivially possible for a checking tool to prove statically that a preset upper-bound on the number of iterations of a loop cannot be exceeded. If the loop-bound cannot be proven statically, the rule is considered violated.
- (3) Do not use dynamic memory allocation after initialization.
- (4) No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than about 60 lines of code per function.
- (5) The assertion density of the code should average to a minimum of two assertions per function. Assertions are used to check for anomalous conditions that should never happen in real-life executions. ...
- (6) The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.
- (7) All code must be compiled, from the first day of development, ... All code must be checked daily with at least one, ..., state-of-the-art static source code analyzer ...
- (8) ...

Summary: Coding Standards

- Rules / Guidelines for Code (or other documents)
- “Codex”
- Goal: High-quality code without “code smells”
- High-quality: maintainable, safe, secure, and fast

Code Reviews

The only valid measurement
of code quality: WTFs/minute



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

Code Reviews: Definition

Code Review

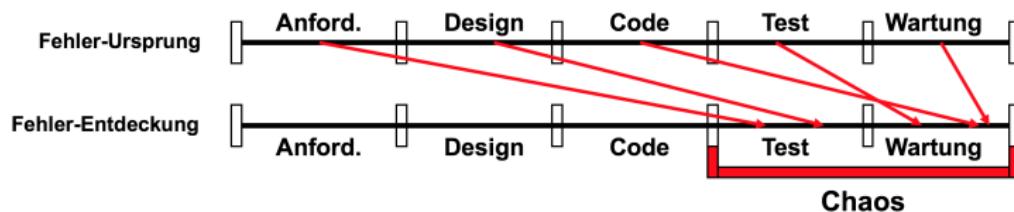
Systematic examination of source code — often by others than the author.

Code reviews can help

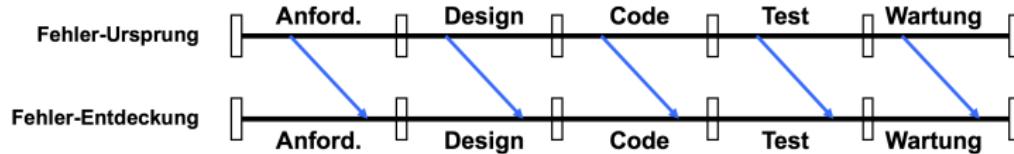
- finding bugs early,
- improve the quality of software,
- sharing knowledge.

Code Reviews: Desired Effect

Ohne Reviews

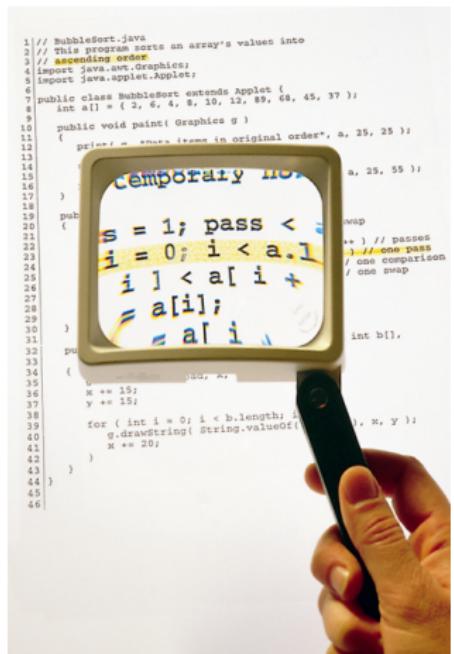


Mit Reviews (ideal)

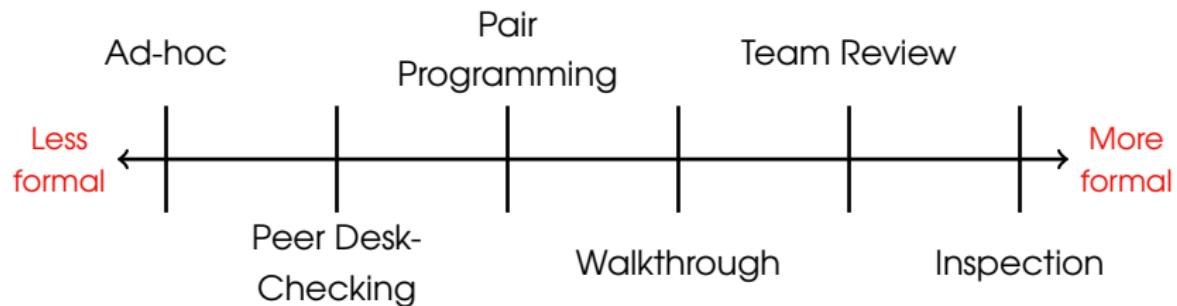


Code Reviews: Discussion

- Often used in safety critical domains
- Reviews can be done early in the process
- Reading, discussing, and presenting code helps better understanding



Code Reviews: Overview



Empirical studies show that Inspection is most effective.

- Inspections lead to 50% more errors discovered than in walkthroughs (Ford)
- Inspections lead to 6x more errors discovered than in ad-hoc reviews

Code Reviews: Methods (1)

Ad-hoc

- Ask colleague for help with problems
- Only problems reviewed
- Outcome depends on who you ask

Peer Desk-Checking

- Similar to ad-hoc review
- Reviewer executes product to be reviewed (e.g., code) on paper

Code Reviews: Methods (2)}

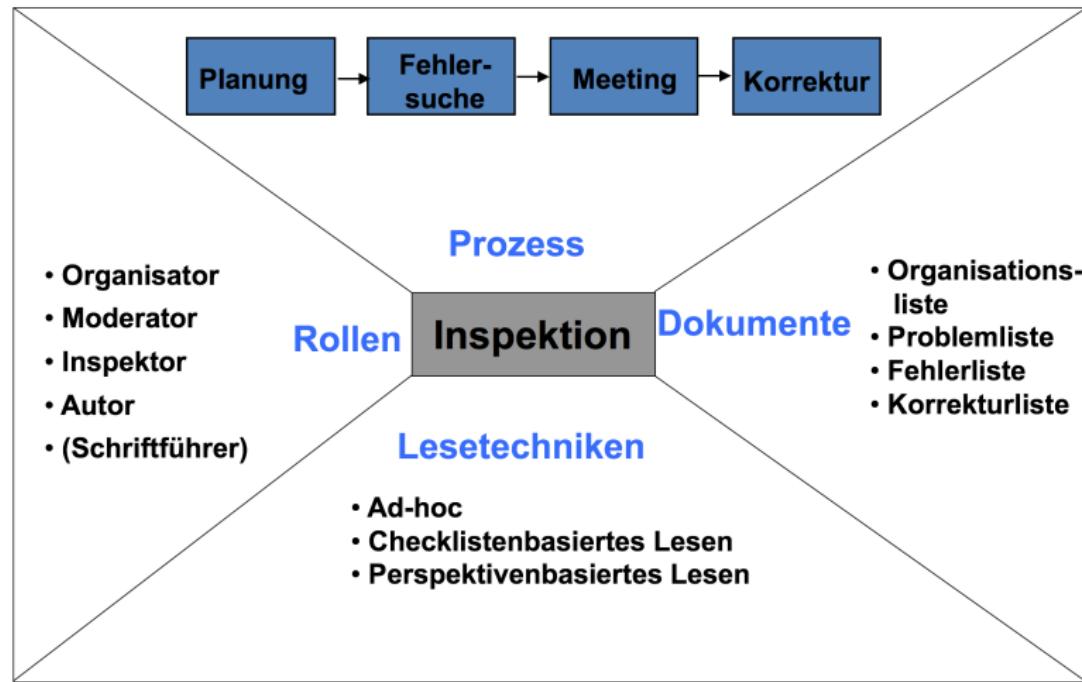
Pair-Programming

- Developers work in pairs of two. One person types and explains, the other person listens and checks
- one technique of “eXtreme programming”

Walkthrough

- Author presents code or document to reviewers
- No defined process or guideline for finding faults
- Risk: author controls focus of review

Code Reviews: Inspection



Inspection is often used for software requirements specifications and test plans.

Code Reviews: Team Review

Team-Review

- Similar to Inspection but less formal
- Several people produce individual reviews
- Results are discussed in a meeting with the author

Examples: Google

A screenshot of a GitHub pull request interface. On the left, there is a code editor showing Java code from the file `gerrit/gerrit-server/src/main/java/com/google/gerrit/server/change/PatchSetInserter.java`. The code defines a class `PatchSetInserter` with various private fields and methods. A specific line of code, `private boolean runHooks;`, has a green highlight. A comment from Stefan Belli is visible above this line, and a reply from Dave Borowitz is shown below it. The GitHub interface includes standard navigation buttons at the top right.

```
gerrit / gerrit-server/src/main/java/com/google/gerrit/server/change/PatchSetInserter.java
106 private PassDescriptor patchSet;
107 private ChangeMessage changeMessage;
108 private SshInfo sshInfo;
109 private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
110 private boolean draft;
111 private boolean runHooks;
112
113 private boolean sendMail;
114 private Account.Id uploader;
115 private BatchRefUpdate batchRefUpdate;
116
117 /**
118 * @Inject
119 public PatchSetInserter(ChangeHooks hooks,
120     ReviewDb db,
```

110 private PatchSet patchSet;
111 private ChangeMessage changeMessage;
112 private SshInfo sshInfo;
113 private ValidatePolicy validatePolicy = ValidatePolicy.GERRIT;
114 private boolean draft;
115 private boolean runHooks < true;
Stefan Belli Why do you move this out of the constructor? Initially I assumed this... Jan 28 2:55 PM
Dave Borowitz Because it would be identical between the two constructors, so it sa... Jan 28 3:19 PM

Email- / Web-based Team Review

- Everyone works on HEAD revision
- Every single change to code-base is reviewed before commit
- Custom web-based system: Mondrian

Examples: NASA JPL

Tool-assisted Inspection

- Tool-based review / response process
- Tool reports added to reviews
- Agrees and disagrees are tracked
- Short meeting (60-90 min) for each module to review comments and responses
- Only disagrees have to be discussed in a meeting

The screenshot shows a software interface for reviewing code. At the top, there's a menu bar with 'File', 'View', 'Exit', 'Review Wrapup', 'Send report', 'Release', 'newest', 'Module', 'mfs', 'Lines: 1', and 'Find'. Below the menu is a toolbar with buttons for 'File', 'Review', 'Save', and 'Analyses' (set to 'stats'). A status bar at the bottom right shows 'Action: None' and 'Fix Code' buttons, along with 'Save' and 'Cancel'.

The main area displays a code editor with the following C code:

```
mfs_tree.c
0102 } mfs_box_ctor(ptr, nboxes, pau, rem
0103 }
0104 STATIC NodeError mfs_tree_hdr_verify(hdr
0105 {
0106     if (FSAL_HDR_VTR(troot != NULL) && (path_is
0107         FSAL_HDR_VTR(troot == NULL)) {
0108         if (path->type == MFS_DIRECTORY) as
0109             return -EDBFR;
0110         else if (path->count < 0) {
0111             return -EDBFR;
0112         } else if (path->count < 0) {
0113             return -EDBFR;
0114         } else if (path->type == MFS_DIRECTORY) {
0115             if (path->count < troot->count) {
0116                 return -EDBFR;
0117             } else {
0118                 // Fall through
0119             }
0120         } else if (path->count > mfs_get_dsz(
0121             return -EDBFR;
0122         } else {
0123             // Fall through
0124         }
0125 }
```

Below the code editor, there are two lists of errors:

- Agree**:
 - .../mfs/mfs_tree.c:113: missing else
 - .../mfs/mfs_tree.c:113: disagreement: The code is slightly confusing, but adding an extra else would not help.
- Disagree**:
 - .../mfs/mfs_posix.c:731: missing else

Summary: Code Reviews

Goals:

- Finding defects (failures and/or faults) as early as possible
- Increase quality of code
- Decrease costs of quality
- Idea: Two pairs of eyes catch more than one
- Shared knowledge (increasing bus factor)
- Better understanding through discussion / presentation

Methods:

- Can be more or less formal
- Today often tool-based
- Ad-hoc, Pair Programming, Inspection, ...

Software Metrics



<http://software-cities.org/>

Software Metrics

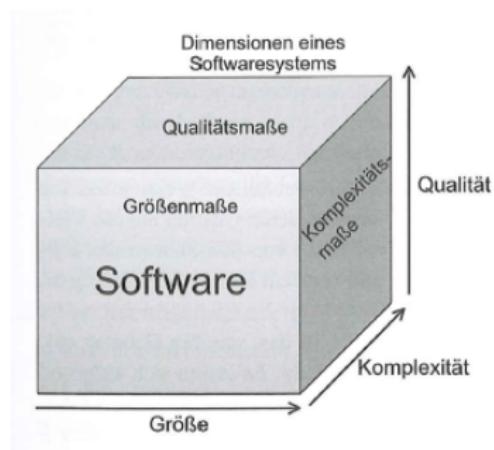
Software Metrics

A software metric is a **standard of measure** of a **degree** to which a software system or process possesses some property.

- A metric is not a measurement!
- Measurements are the numbers obtained by the application of metrics

Software Metrics: Outline

- LOC: Lines of Code
- LCOM: Lack of Cohesion
- CC: Cyclomatic Complexity



Lines of Code (LOC)

Lines of Code (LOC)

LOC counts the number of lines in the text of a program's source code.

- Simplest indicator for project complexity.
- Similar Metrics: Number of Classes, Number of Methods, Number of ...

Lack of Cohesion Metric (LCOM)

Cohesion: Degree to which things belong together.

Lack of Cohesion Metric (LCOM)

For a class with a fields, and m methods, let $n(A_j)$ count the number of methods that access field A_j . Lack of cohesion is computed as

$$LCOM = \frac{\left(\frac{1}{a} \sum_{i=1}^a n(A_i)\right) - m}{1 - m}$$

- “Belong together” measured by counting access
- High LCOM value \Leftrightarrow low cohesion

Lack of Cohesion Metric (LCOM) - Example

Dreieck
-a : int
-b : int
-c : int
+setKanten(a:int, b:int, c:int)
+zeichne()
+umfang()
+flaeche()

Kreis
-r : int
+setRadius(r:int)
+zeichne()
+umfang()
+flaeche()

Figur
-a : int
-b : int
-c : int
-r : int
+zeichneDreieck()
+zeichneKreis()
+flaecheDreieck()
+flaecheKreis()
+umfangDreieck()
+umfangKreis()
+setRadius(r:int)
+setKanten(a:int, b:int, c:int)

Classes Dreieck and Kreis

access every field in every method.

Dreieck:

$$a = 3, m = 4, A_1 = 4, A_2 = 4, A_3 = 4$$

$$\Rightarrow LCOM = \frac{(1/3 \cdot 12) - 4}{1 - 4} = 0$$

Kreis:

$$a = 1, m = 4, A_1 = 4$$

$$\Rightarrow LCOM = \frac{(1/1 \cdot 4) - 4}{1 - 4} = 0$$

Figur = Dreieck + Kreis:

$$a = 4, m = 8, A_1 = 4, A_2 = 4, A_3 = 4, A_4 = 4$$

$$\Rightarrow LCOM = \frac{(1/4 \cdot 16) - 8}{1 - 8} = \frac{4}{7}$$

Cyclomatic Complexity (CC)

In the next deck of slides ...

Tools

PMD

PMD scans Java source code and looks for potential problems like:

- Duplicate code - copied/pasted code means copied/pasted bugs
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Many other things (we will come back to PMD later)

The screenshot shows a web browser window displaying the PMD results for a Java project. The URL in the address bar is `file:///Users/falk/Workspace/teaching/sse-examples/target/site/pmd.html`. The page title is "PMD Results". It indicates the document was last published on 2016-01-17 and is version 0.1-SNAPSHOT. A "Built by Maven" badge is present. The page lists violations across several Java files:

- de/tuc/ifi/sse/handsfree/ecusw/cdl/CdlComponent.java**
 - Violation Line
Avoid empty catch blocks 30-32
- de/tuc/ifi/sse/handsfree/ecusw/tel/TelApiModule.java**
 - Violation Line
Avoid unused imports such as 'java.util.ArrayList' 8
- de/tuc/ifi/sse/handsfree/ecusw/tel/api/TelInterfaceForCan.java**
 - Violation Line
Avoid modifiers which are implied by the context 14
- de/tuc/ifi/sse/handsfree/ecusw/tel/api/TelInterfaceForCdl.java**
 - Violation Line
Avoid modifiers which are implied by the context 18
 - Avoid modifiers which are implied by the context 25

At the bottom, a copyright notice reads "Copyright © 2016. All Rights Reserved."

<https://pmd.github.io/>

How to decide what to measure

Goal Question Metric Paradigm (GQM)

Goal: What is the goal?

Example: Improving Code-Design

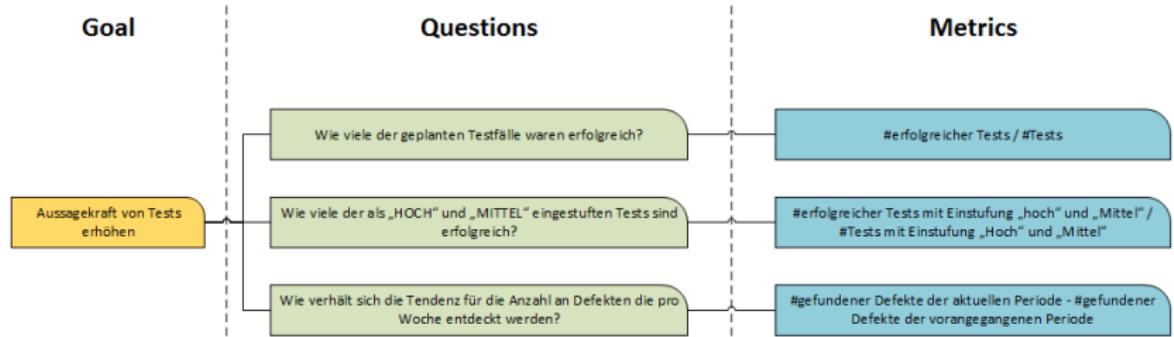
Question: What needs to be analyzed?

Analyze: Nesting level, cohesion, ...

Metric: What is a relevant software metric?

Metric: Lack of Cohesion (LCOM)

GQM Example 2



Summary: Code Metrics

Metric:

- Standard of Measure
- Can be computed for Software
- Examples: LOC, LCOM, CC, ...

Goal:

- Quantify some Aspect of Product

Careful!

- Only Indicators
- Relationship between Goal and Metric?