

PLM and Innovation
Excellence

Learning Campus

Your partner for
Business Learning

Siemens
Core
Learning
Program

Internal Quality

Authors: Peter Zimmerer, CT | Horst Sauer, CT | Christian Hahn, CT

Internal quality

Learning objectives

- Learn about the negative effects of neglecting internal quality
- Understand what internal quality is and how to measure it
- Understand the connection between internal quality and technical debt
- Learn how to drive internal quality and manage technical debt
- Understand the role of the test architect for internal quality

Scope of *Internal Quality*

As a **Test Architect**
You have **one Role** – but you are wearing **two Hats**

Remember WS1

SIEMENS
Ingenuity for life



Test Expert

for the system under test (SUT)

- Design the test approach
- Apply innovative test technologies
- Drive the quality of the SUT



***Internal Quality
of the SUT (production code)***



Software / System Architect

for the test system

- Design and realize the test architecture
- Apply innovative software technologies
- Drive the quality of the test system

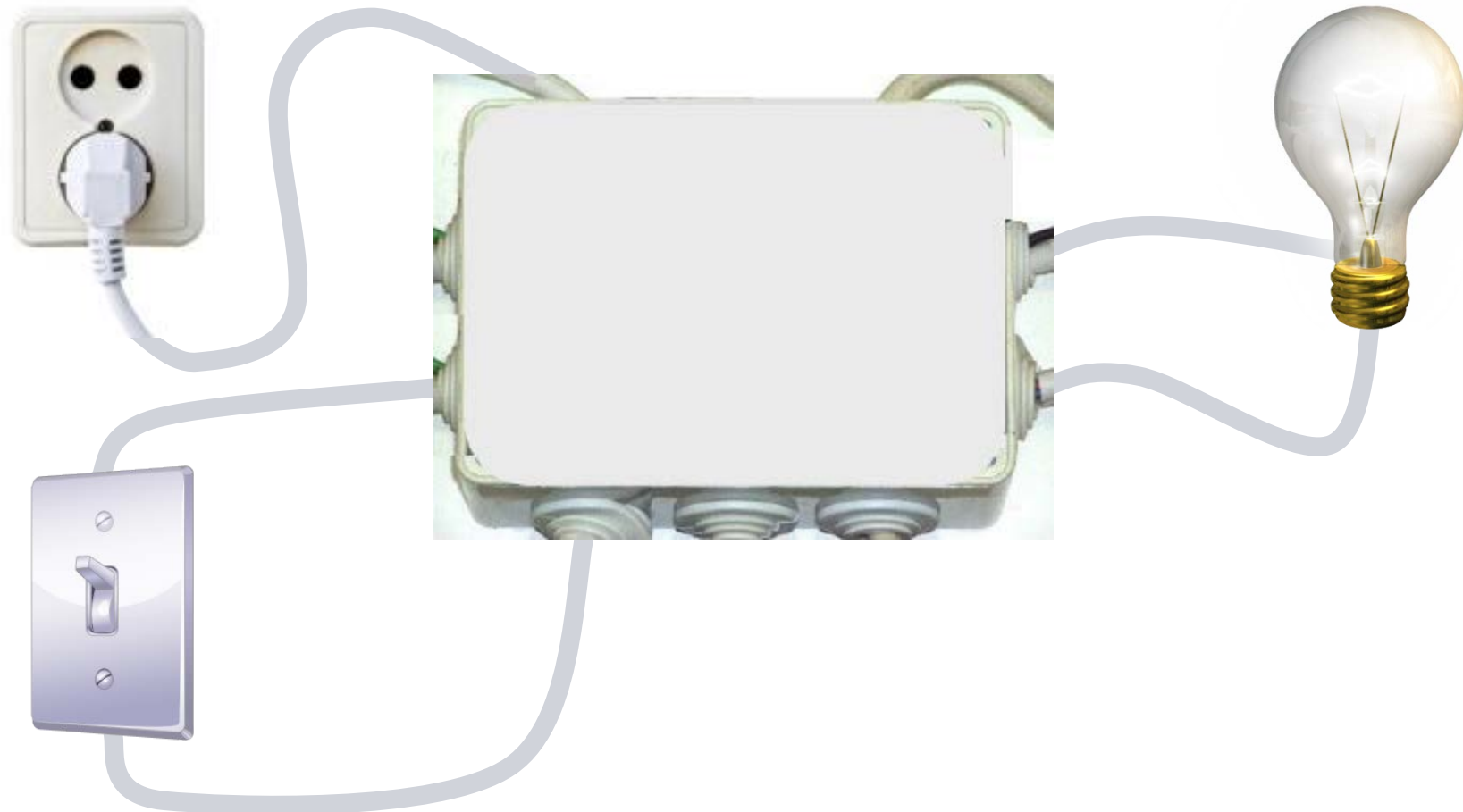
This is the
architect's job!



***Internal Quality
of the test system (test code)***

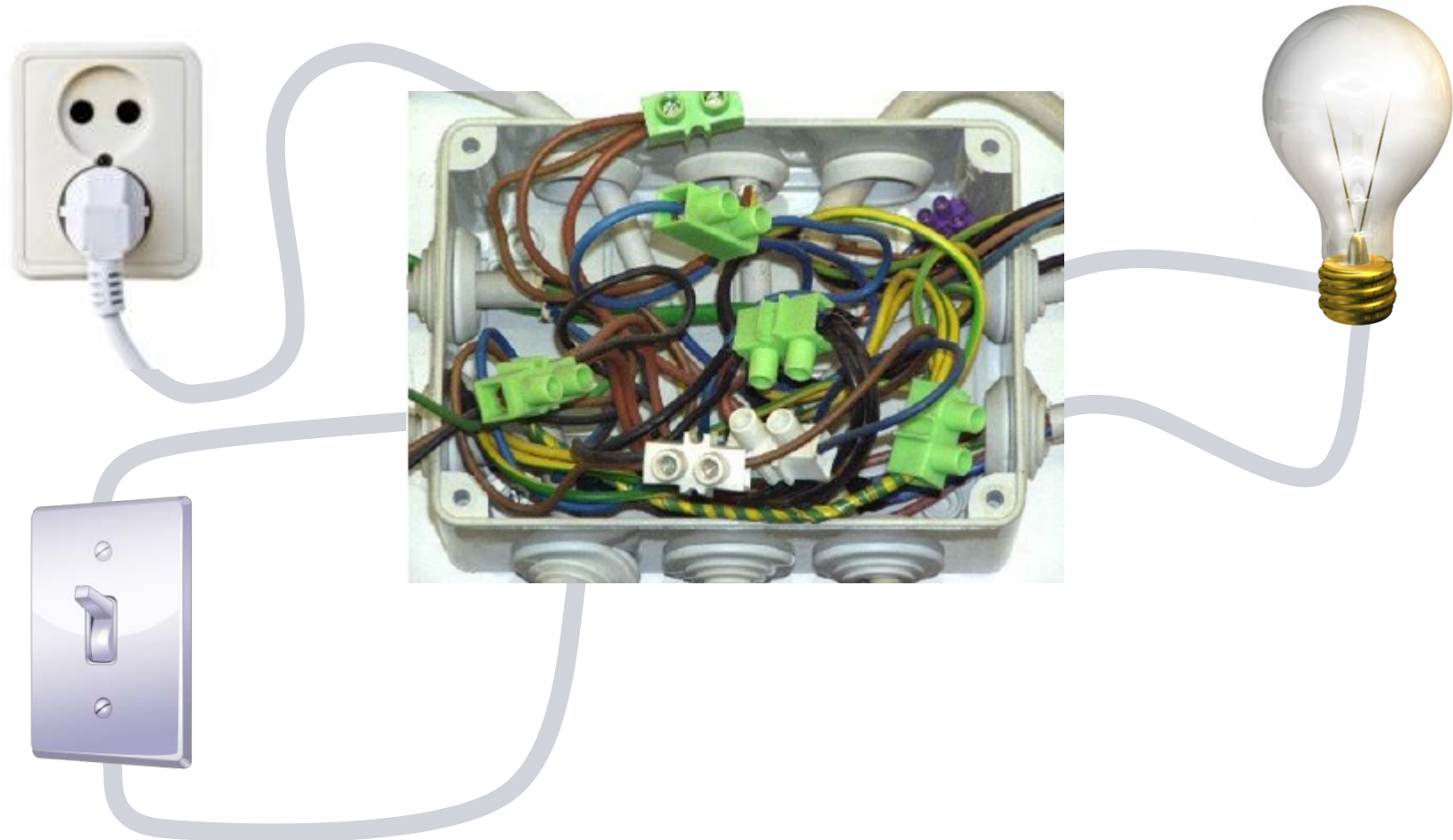
Internal quality

Perfect functionality is important, but not the only aspect to consider.



Internal quality

Even with perfect functionality there can be huge maintenance risks.



Internal Quality

Agenda

Internal Quality / External Quality / Quality in Use

Understand and Drive Internal Quality

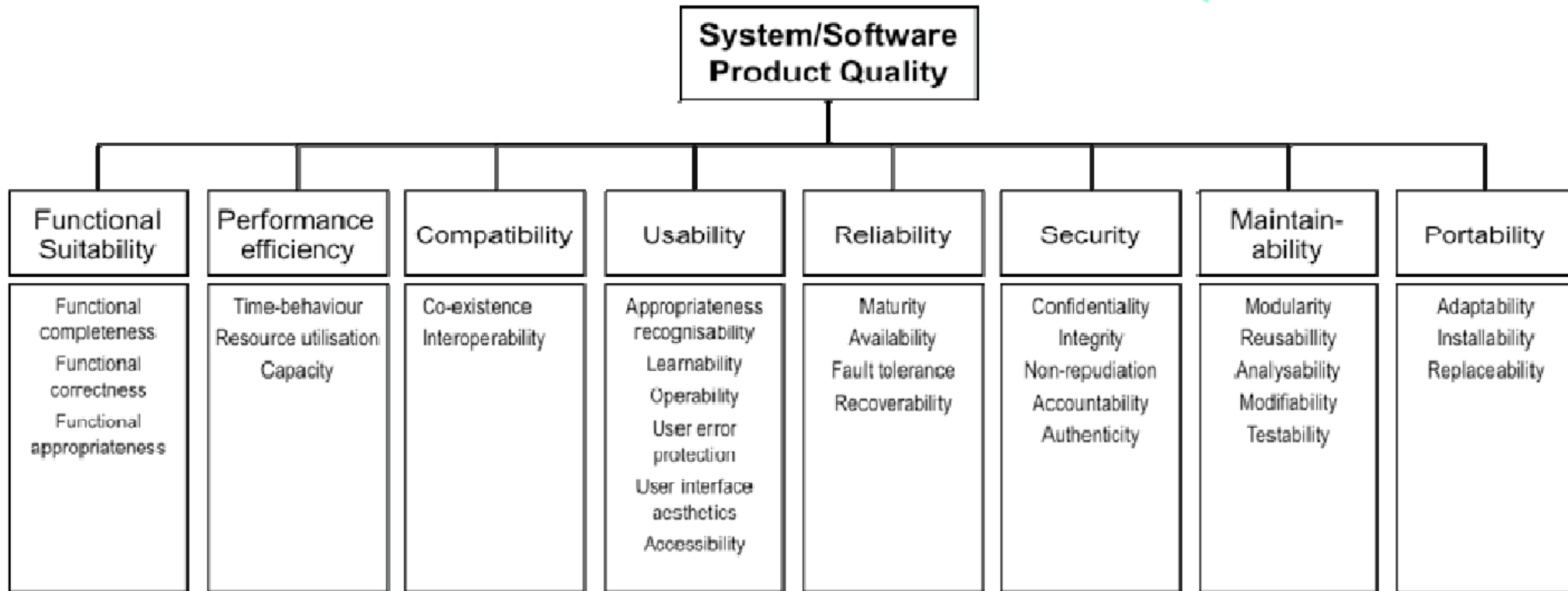
Understand and Manage Technical Debt

(Test) Code and Architecture Quality Management

Summary

ISO/IEC 25010 Product quality model

Remember WS2



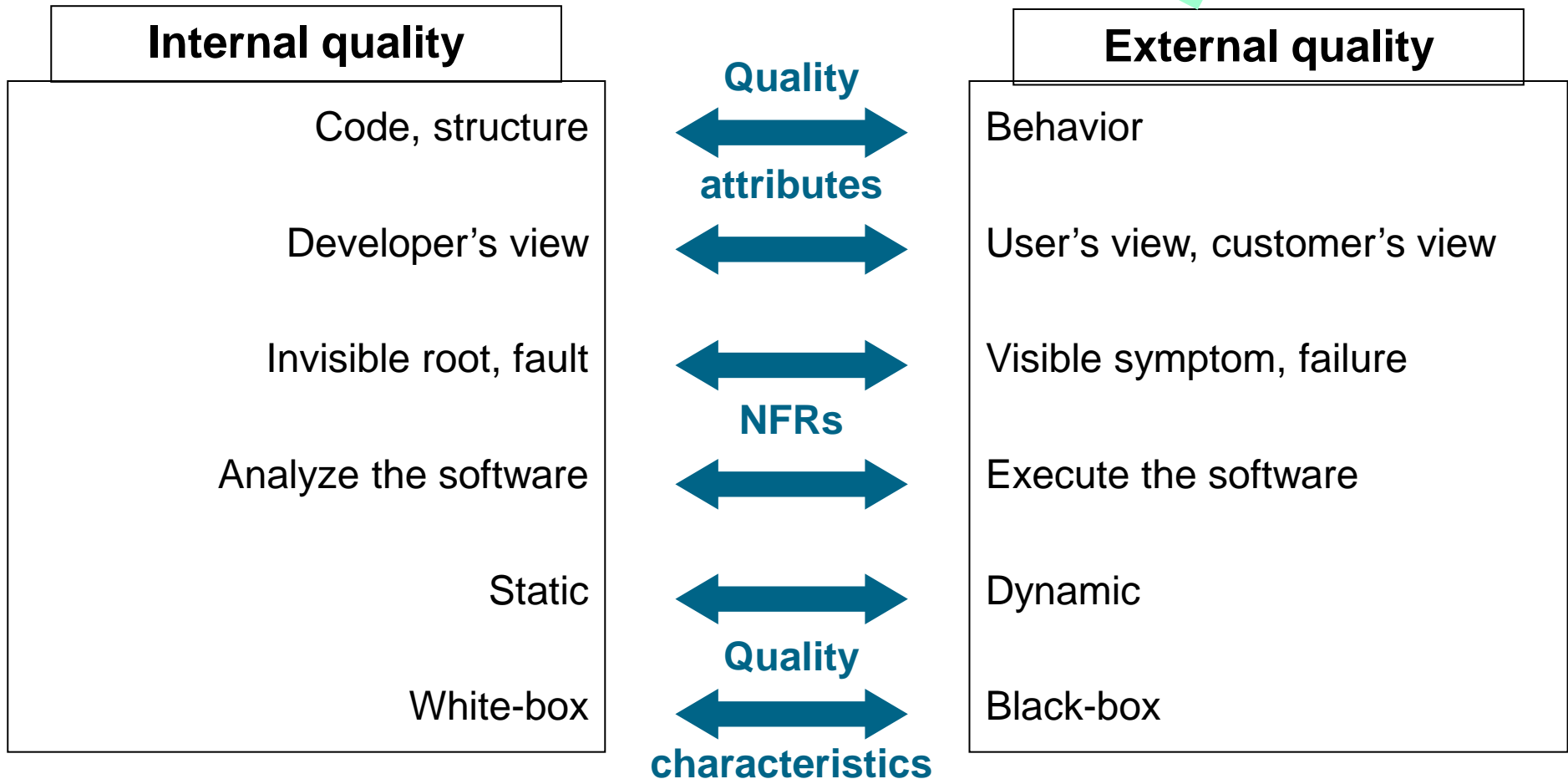
Internal measures characterize software product quality based upon static representations of the software, **external measures** characterize software product quality based upon the behaviour of the computer-based system including the software, and **quality in use measures** characterize software product quality based upon the effects of using the software in a specific context of use. (ISO/IEC 25020)

<http://www.iso.org/>

Close relation between internal and external quality

Remember WS2

SIEMENS
Ingenuity for life



Internal metrics measure the software itself, **external metrics** measure the behavior of the computer-based system that includes the software. (ISO/IEC 9126-1)

Exercise

Discussion on examples, consequences / effects, and root causes

Discuss and collect

Examples, consequences / effects, and root causes

- What are examples of poor internal quality?
 - What are the effects of neglecting internal quality?
 - How do these effects show up?
 - What are the root causes behind?
-
- Have you experienced such effects in your current project?
 - How do you deal with it?



Internet blog "QA Rocks" by Martijn de Vrieze, 2008

External quality is that which can be seen by customers and which is traditionally tested. Bad external quality is what can be seen: system crashes, unexpected behavior, data corruption, slow performance.

Internal quality is the hidden part of the iceberg, i.e. program structure, coding practices, maintainability, and domain expertise. Bad internal quality will result in lost development time; fixes are likely to introduce new problems and therefore require lengthy retesting. From a business point of view, this will invariably result in loss of competitiveness and reputation.

External quality is a symptom whereas the root problem is internal quality. Poor internal quality leads to high maintenance costs. In order to improve software quality, internal quality must be improved.

http://www.qa-rocks.com/index.php?option=com_content&task=view&id=14&Itemid=28

Close relation between internal and external quality – Examples

Performance issues:

- Are visible through behavior
- Can be detected with dynamic analysis, for example using profiling tools which point to specific critical statements in the code
- Can be detected with static analysis tools, for example bad coding practices like String concatenation with + operator in Java

Call dependencies:

- Potential calls can be detected with static analysis
- Actual calls can be detected with dynamic analysis
- Only the combination of both provides solid information

Unit testing, tracing, debugging:

- One focus is on the external behavior → Black-box
- Another focus requires internal knowledge about the code → White-box

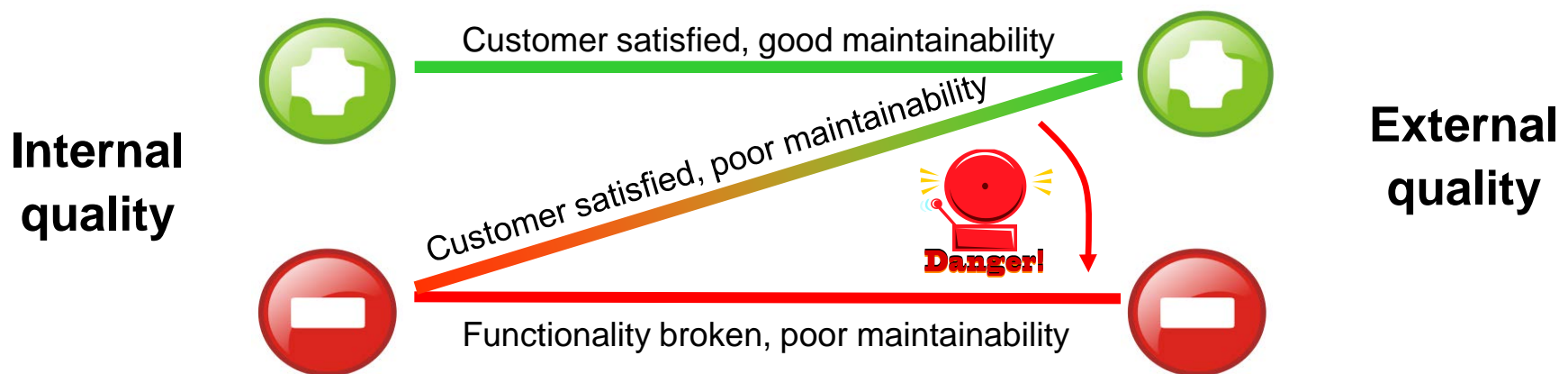
Possible gap between internal and external quality

Good external quality, bad internal quality

- happens often in younger projects with a small developer team; tends to get a nightmare later on during maintenance (technical debt)

Good internal quality, but the system does not fulfill the functional requirements

- rather theoretical situation which practically does not occur



Further negative effects of neglecting internal quality

Slow down in **development progress**

Many **unplanned activities** interrupting development

Adding **new features gets slower** with every release

Rising **cost of change**; bad adaptability/extensibility

Increasing amount of **rework** necessary

Integration becomes **complex and risky**

Expensive **release stabilization**

High efforts for **defect analysis and correction**

High **maintenance and sustainment cost**

Increased cost for **regression testing**

System Testing is overloaded with hot fix testing

High **training efforts** necessary, e.g. for service group

**Today's convenient
solutions, are
tomorrow's tasks.**

[Chinese Wisdom]

Internal Quality

Agenda

Internal Quality / External Quality / Quality in Use

Understand and Drive Internal Quality

Understand and Manage Technical Debt

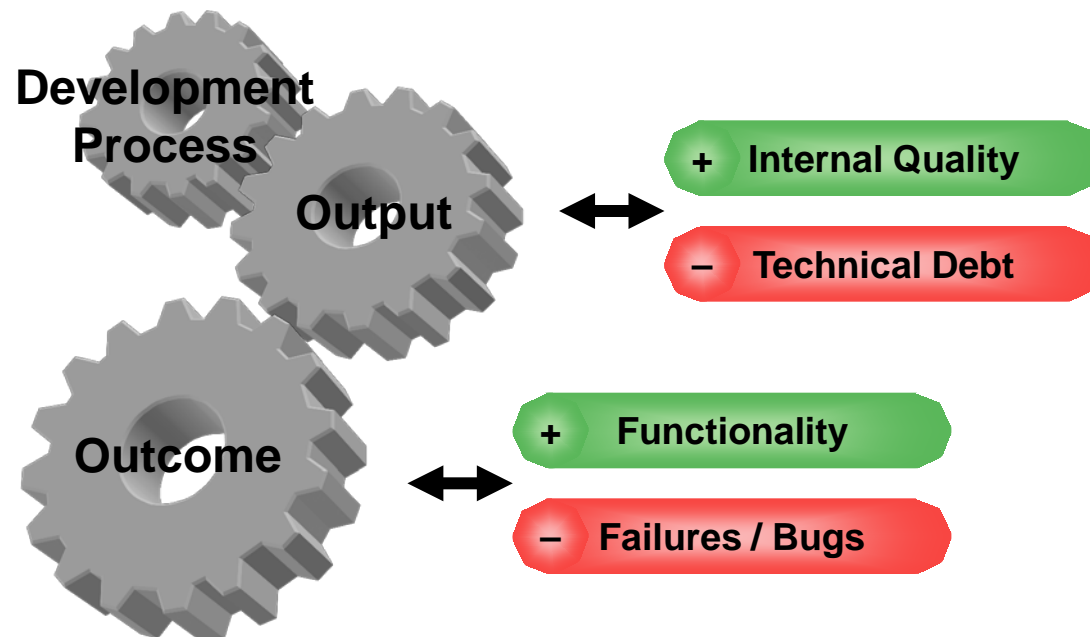
(Test) Code and Architecture Quality Management

Summary

Internal quality

Internal quality relates to the quality of engineering artifacts.
Letting internal quality slip, shows up as technical debt.

Internal quality is about **developing the system right**,
Not about **developing the right system** (external quality).



Driving internal quality (IQ)

To sustain internal quality, teams must approach development in a disciplined way. This includes practices in the process, quality and architecture area.

IQ management must be part of daily development work

- Continuous monitoring of IQ
- Early identification of emerging internal quality problems
- Architecture and design reviews
- Refactoring / Redesign for (internal) quality
- IQ should be part of “Technically Done” definition (→ quality gates)

Development process must support effective IQ management

- Work in potentially shippable product increments
- Small batches of work
- Sustainable pace
- Single work queue / backlog (also including the IQ sustaining tasks)
- Close collaboration

Visualizing internal quality – Reporting & Trends

Internal quality cockpit



Exercise – Discussion

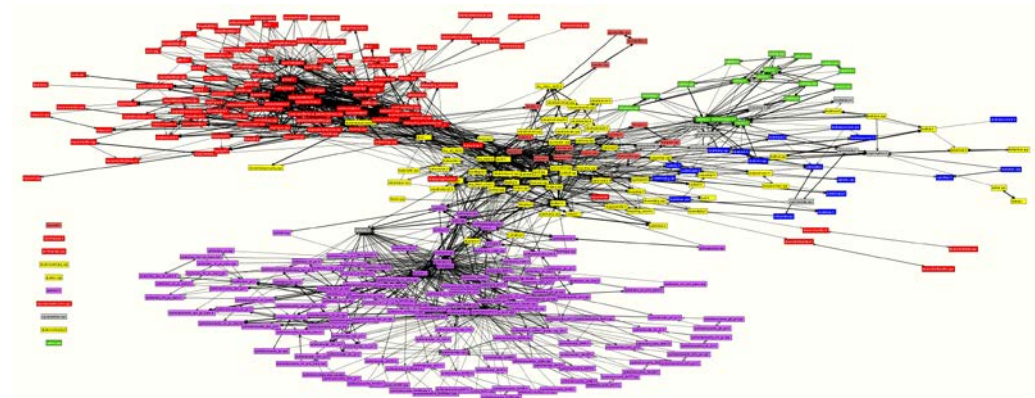
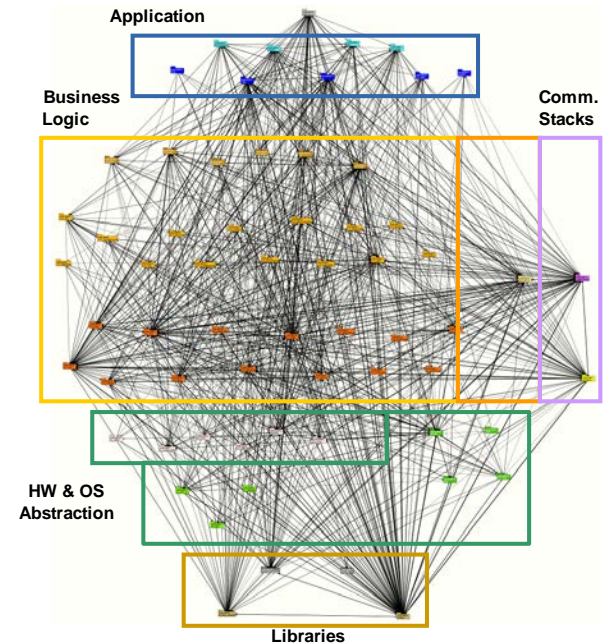
How would you measure the following quality aspects?

Area	Aspects
Architecture	<ul style="list-style-type: none"> ■ Structuredness / Modularity ■ Interfaces / Dependencies ■ Architecture / Design principles ■ Simplicity ⇔ Complexity ■ Extensibility / Variability ■ ...
(Software) Design	<ul style="list-style-type: none"> ■ Code Quality / Code Smells ■ Structure ■ Clarity / Maintainability ■ Conciseness ■ ...
Testing and Quality	<ul style="list-style-type: none"> ■ Risk coverage ■ Feature coverage ■ ...
Documentation	<ul style="list-style-type: none"> ■ Completeness ■ Correctness ■ Consistency ■ ...

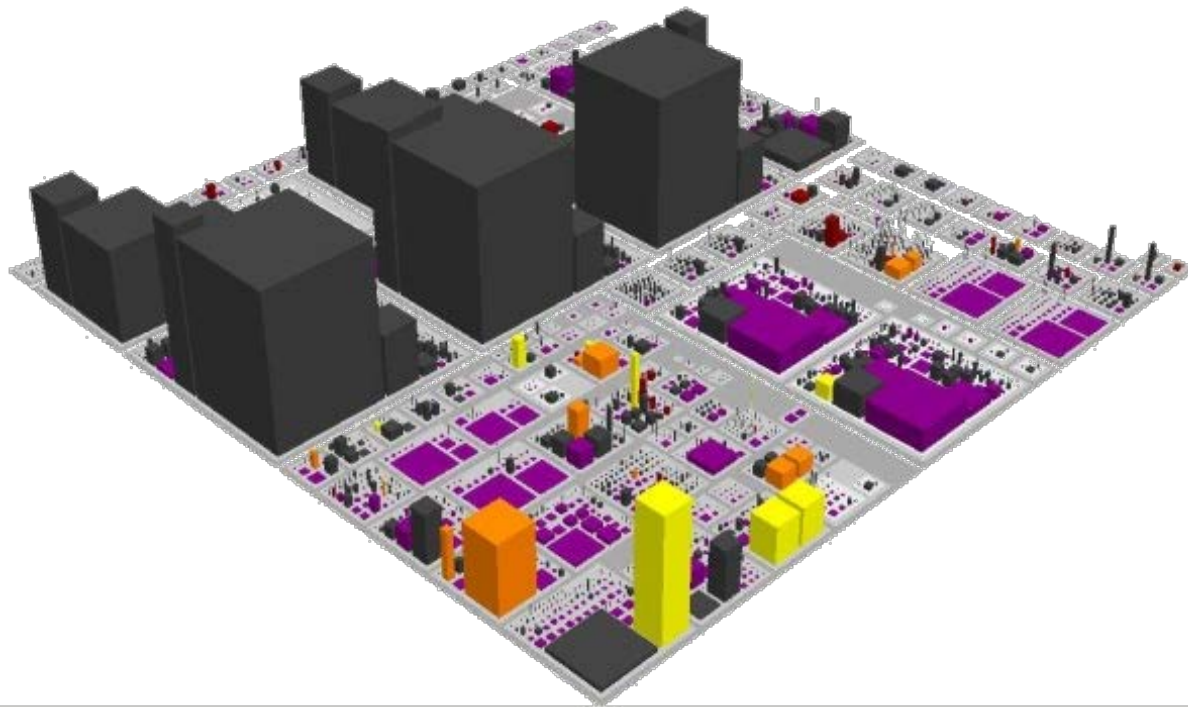


Visualizing internal quality – Architecture Dependency structure matrix (DSM) & dependency graphs

	..1	..2	..3	..4	..5	..6	..7	..8	..9	10	11	12	13	14	15	16
1	.	3		263	125	164		45						142	128	
2	42	.	2597	342	74	335	172	75	69			4		213	1492	
3	452	2172	.	397	75	215	1293	47	18			149		1318	9955	
4	605	1274	437	.	557	2157	1329	1224				63		784	1715	
5	826	3427	062	1812	.	3100	436	1514	13			308		2136	2908	
6	607	1745	630	2881	543	.	7858	4984	1113	1995		7917	6	2008	2501	3
7	228	1085	316	969	81	707	.	900	1	5		1841		156	70	
8	129	918	487	690	592	954	691	.	997	131		250		2279	1453	
9	305	1150	488	665	840	1573	1914	1071	.	156		600	2	73	244	
10	44	53	30	23	63	176	91	6	12	.	9	175			2	
11	33	40	40	75	92		52	66		795	.	210				
12	521	216	678	866	237	443	498	1191	167	680		.		8	48	
13	1317	22	1892	1449	1524	7334	1620	1128	8			68	.	662	760	
14	11	11	28	395		84		89	6			2		.	1521	
15	7845	8428	8085	3339	774	0056	9076	4542	5951	639	108	0110	892	769	.	27
16	388	151	1515	2515	1163	3543	1051	1523	1			69		177	507	.



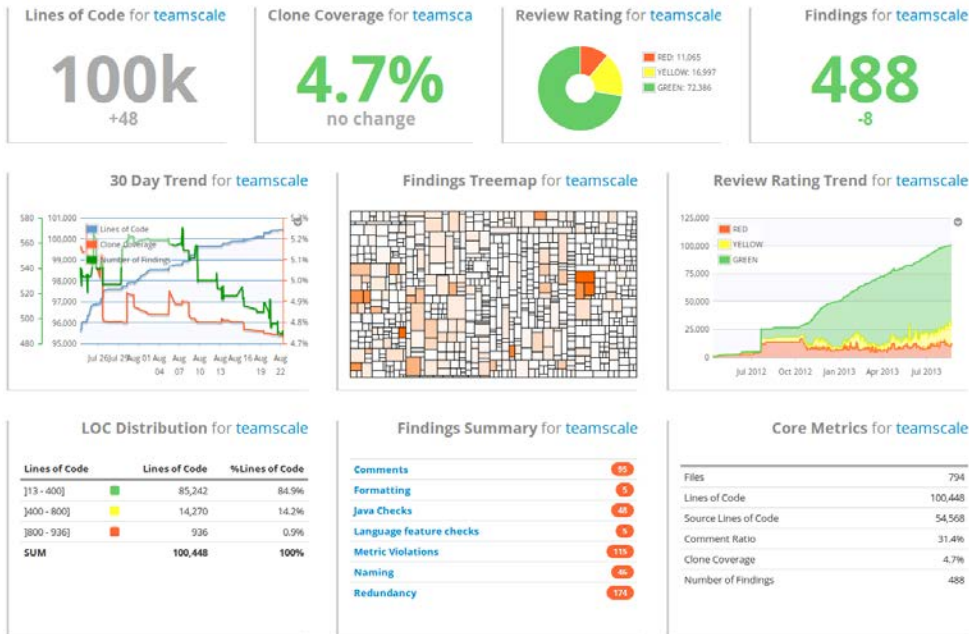
Visualizing internal quality – Code Code City



Visualize internal qualities

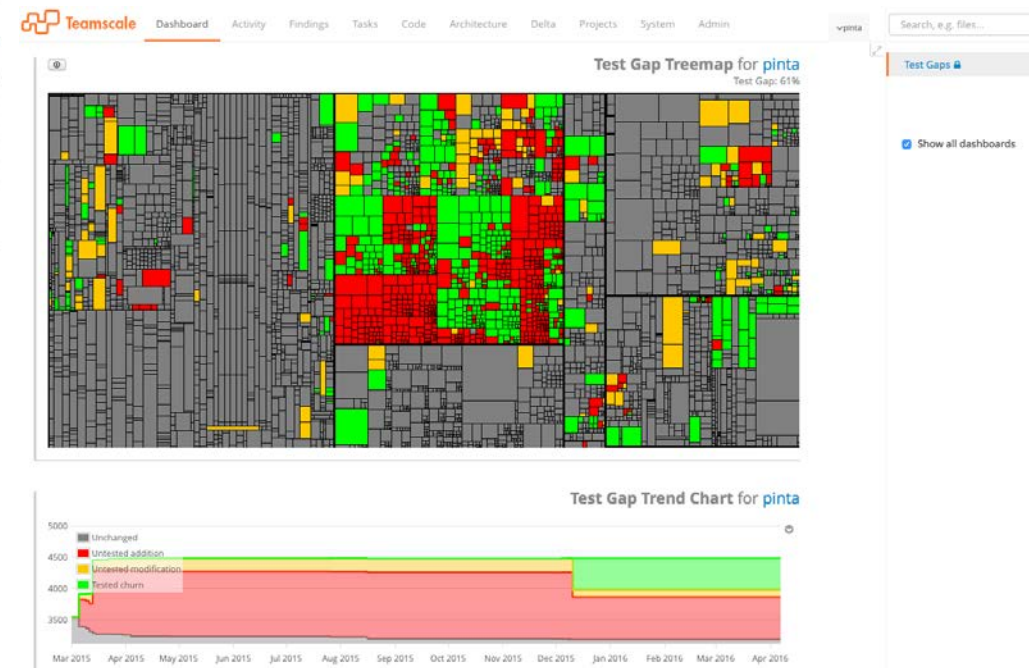
- Size, complexity
- Change frequency, code churn
- Criticality
- Error proneness
- ...

Visualizing internal quality – Code and beyond CQSE Teamscale



Visualize internal qualities

- Architecture conformance
- Clone detection
- Test gap analysis
- Structural analysis
- History analysis



<https://www.cqse.eu/>

Restricted © Siemens AG 2016-2017

Visualizing internal quality – Documentation Findings from automated document analysis

Quality model for documents:

Komponente weitergeleitet werden. Darüber hinaus verringert sich der Störeffekt durch die Instrumentierung auf die Target-Software. Wie die jeweiligen Filtermöglichkeiten im Detail aussehen, wird in der Systemspezifikation beschrieben.

35 Filter können beim Start eingerichtet werden

36 In einer späteren Version können Filter interaktiv definiert, aktiviert und deaktiviert werden

3.1.5.2 Trigger

37 Auf Instrumentierungsseite können Trigger eingerichtet werden.

Für den Test des verteilten Software-Systems muß vom Benutzer bei allen diesen Testszenarien ein geeigneter Testtreiber bereitgestellt werden, der die Applikation "zum Laufen bringt", geeignete Testanreize liefert und die Testdurchläufe steuert (Anmerkung: Ein solcher Testtreiber müßte vom Entwickler für die Durchführung der Tests auch ohne Anwendung erstellt werden, und wird daher vorausgesetzt). In einer späteren Version ist beabsichtigt, daß die Erstellung solcher Testtreiber unterstützt.

Comment [DQM13]: Avoid duplicates in document: Document: ReqSpec_980305.doc has duplicate content: Sentence (Wie die jeweiligen Filtermöglichkeiten im Detail aussehen, wird in der Systemspezifikation beschrieben.) - Sentence 2 (Wie die jeweiligen Filtermöglichkeiten im Detail aussehen, wird in der Systemspezifikation beschrieben.)

Comment [DQM15]: Avoid long sentences: Sentence consists of 52 words (max. 30) - Section: Test-Komponente

Comment [DQM16]: Each figure must be referenced in text: Figure 'Abbildung 9' is not referenced in a sentence

Comment [DQM22]: Each term of a local glossary should be unique, as multiple definitions create confusion (homonym): Local glossary entry (API) found in document 'ReqSpec_980305.doc' has a different definition compared to the same glossary entry found in e.g. document 'D:\dqm_bin_0.2.0\published_docs_adapte d\DD\AnalyseDD.doc'

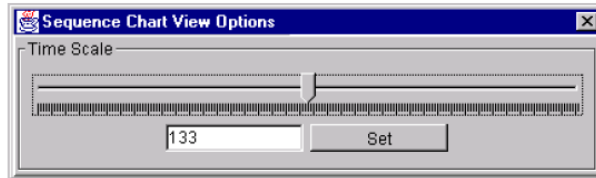


Abbildung 9: Sequence Chart View Option „Time Scale“

9 Glossar

API Application Programming Interface

<https://code.siemens.com/codequality/document-quality-tool>

Restricted © Siemens AG 2016-2017

- Q Documentation Factors
 - Conformity to Documentation Convention
 - Conformity to Documentation Convention @Document
 - DQM DESOR - Define Expected Skills of Readers
 - DQM DMHA - Document must have Author
 - DQM DMHTOC - Document must have TOC
 - DQM DMHV - Document must have Version ID
 - DQM DMHVH - Document must have Version History
 - DQM DMS - Documents must have a State
 - DQM PGLOS - Provide local glossary
 - DQM PIFF - Provide Index for Figures
 - DQM PIFT - Provide Index for Tables
 - Conformity to Documentation Convention @Figure
 - Conformity to Documentation Convention @Project
 - Conformity to Documentation Convention @Table
 - Conformity to Naming Convention - Conformity to Naming Convention
 - Content Integrity
 - Content Integrity @Document
 - DQM AIDOC - Avoid Incomplete Documents
 - Definition and Usage Consistency
 - Reference Validity
 - Reference Validity @CrossReference
 - DQM ABIR - Avoid Broken Internal References
 - DQM ABRIF - Avoid Broken References to Internal figures
 - DQM ABRIT - Avoid Broken References to Internal tables
 - Reference Validity @Hyperlink
 - Spelling Integrity
 - Storage Consistency
 - Structural Complexity
 - Structural Complexity @Section
 - DQM ADNS - Avoid Deeply Nested Sections
 - Structuredness
 - Structuredness @Document
 - DQM AESD - Avoid Extremely Small Documents
 - DQM ATDS - Adhere to Document Structure
 - Uselessness

Internal Quality

Agenda

Internal Quality / External Quality / Quality in Use

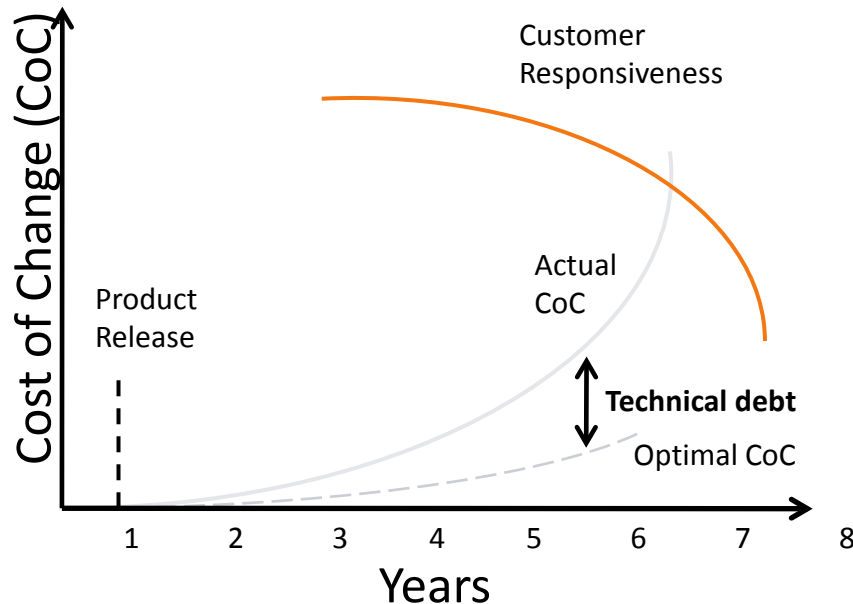
Understand and Drive Internal Quality

Understand and Manage Technical Debt

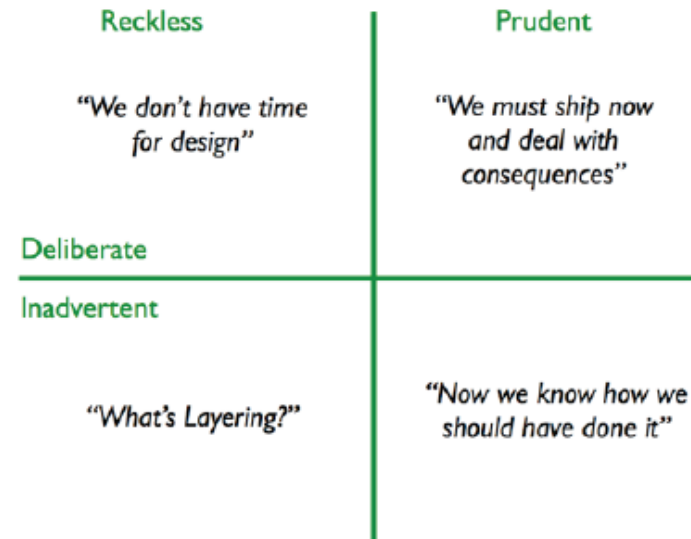
(Test) Code and Architecture Quality Management

Summary

What is technical debt?



Technical Debt (M. Fowler)



Fowler 2009, 2010

Technical debt (TD) accumulates for several reasons:

- Daily tradeoffs between quick value, quality and project constraints (cost, schedule)
- Maintenance postponed in favor of important „business value add“ projects
- Deliberate or inadvertent ignorance of (internal) quality
- Lack of maintenance for additional functionality and complexity
- Aging effects of continuously growing or changing engineering artifacts
- Postponed upgrades of the underlying platform infrastructure

Monetizing technical debt

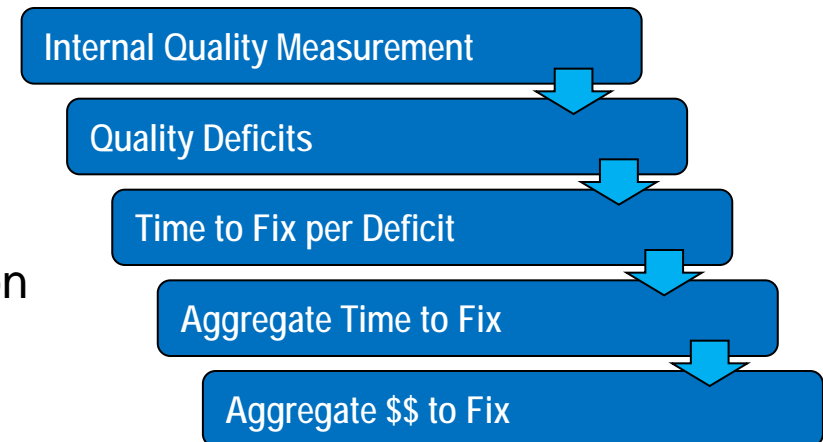
Time is money

- „Think about the amount of money the borrowed time represents – the grand total to eliminate all issues found in the code.“
(Israel Gat, Cutter Consortium)
- Once monetized, the operational, financial, and business implications of TD can be understood by any stakeholder.



Composition of the technical debt metric

- TD is a composed financial metric based on quantitative measures of the structural quality
- The elements of TD pile up from implementation level metrics to aggregated maintenance cost



Original figure ©2010 Israel Gat

System vs. software technical debt

Software technical debt focuses more on:

- Software / code quality

System technical debt includes:

- System definition quality (requirements, architecture, interfaces, trades, analysis)
- System operational/performance issues – both internal and interoperability
- System lifecycle management issues (e.g., adaptability, extensibility, ...)
- System maintainability/sustainment issues

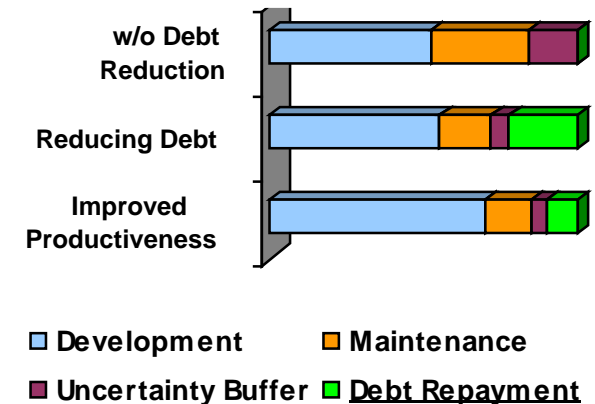
Strategic investment to reduce technical debt

Strategies for technical debt management

- Ignorance is bliss? → It gets worse!
- Replace / Re-implement → High cost / risk
- Debt conversion → Shifts problem to another place
- Intentionally going into debt → Only with a clear payback plan!
- Strategic investment in TD reduction → Investment in TD reduction and prevention

Strategic investment in technical debt reduction

- Incorporate technical debt reduction as a regular activity
- Make technical debt a visible item on the backlog
- Make it visible to all stakeholders, not only to development
- Adapt technical debt investment over time as needed



Managing technical debt

Technical debt metrics

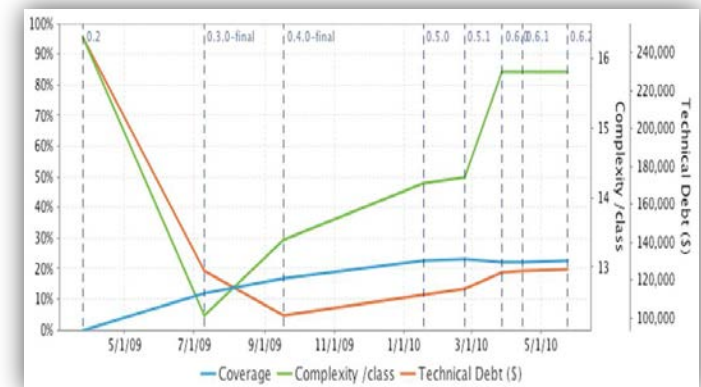
- Operationalize technical debt metrics
- Integrate monitoring of technical debt metrics into your development process
- E.g. check technical debt with every build/integration step

Statistical process control

- Define technical debt characteristics and thresholds
- Set up quality gates and define escalation strategies

“Stop the Line”

- ... when ever technical debt rises above defined limits
- It's the Lean principle of early error detection/correction



Internal Quality

Agenda

Internal Quality / External Quality / Quality in Use

Understand and Drive Internal Quality

Understand and Manage Technical Debt

(Test) Code and Architecture Quality Management

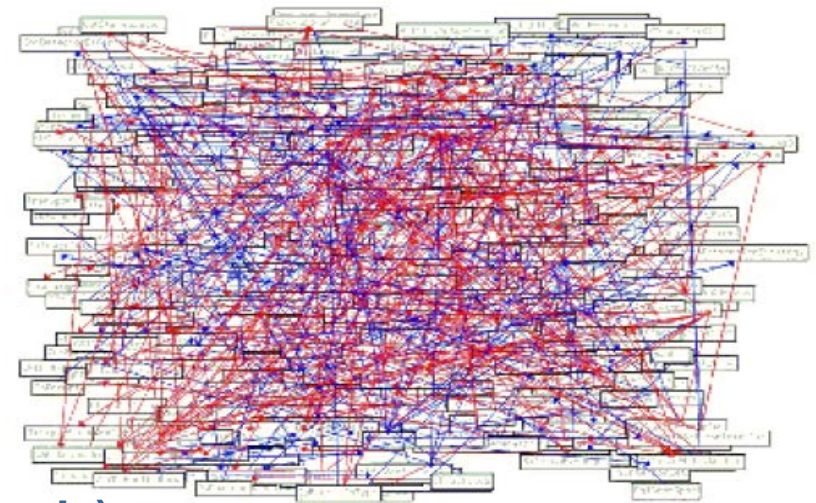
Summary

Why is (test) code quality management an important issue for Test Architects?

In many projects the responsibility for internal code quality is not well defined. The test architect and the software architect have to ensure that (test) CQM is done well.

The Test Architect should be the protector and driver of the (internal) quality

- for the system under test (SUT, production code)
- for the test system (test code)!



Benefits from the Test Architect's point of view:

- Higher overall quality – Speeds up development by facilitating change
- Decreasing extension and maintenance efforts
- Better reusability of components
- Influence on non-functional requirements

(Test) Code quality assessments help to make the internal quality transparent



**Go / no-go decisions
at important milestones**

**Strategic decisions
like
platform selection**

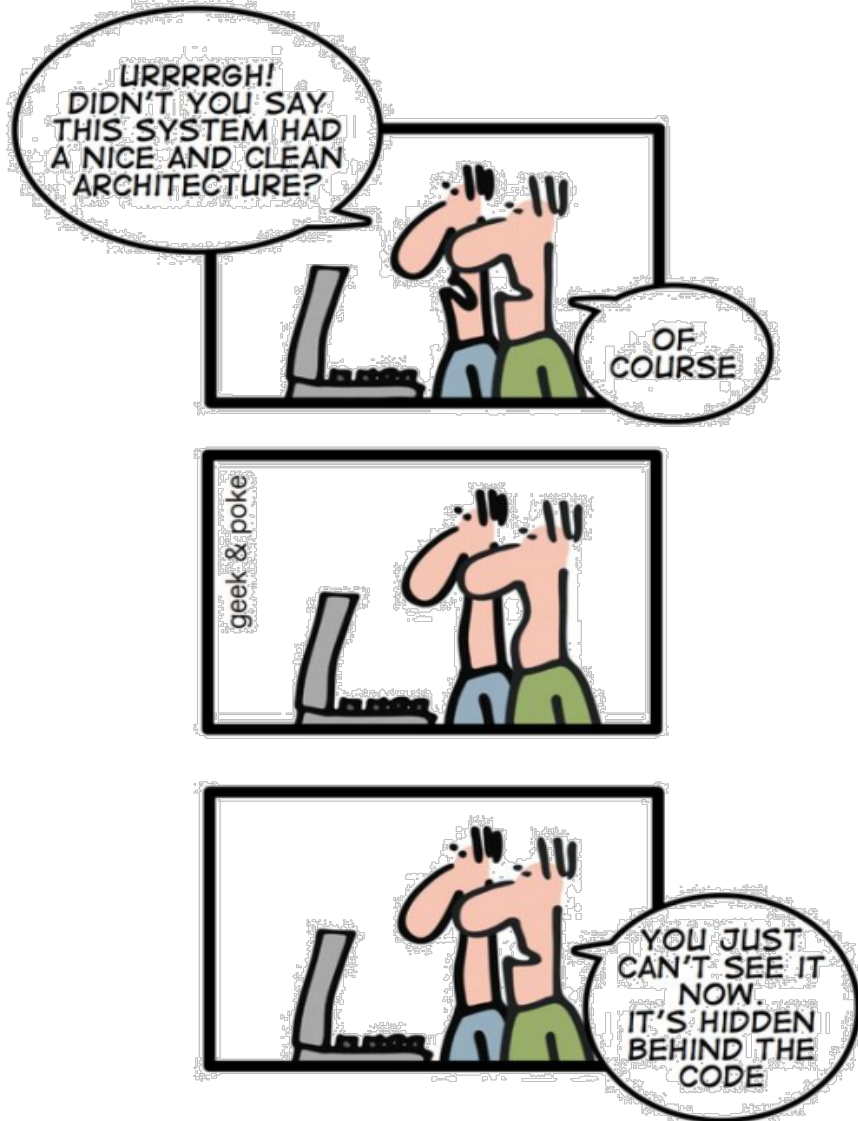


**Making
software aging
visible**

**Identifying
"servicing" needs and
support maintenance**



(Test) Coders love clean architectures and honor the dead



Some Java code – Do you find potential quality issues ...? *Ingenuity for life*

```
/**
 * This class does some nonsense in a
 * dubious way.
 */
public class DubiousStuff {

    ArrayList<String> names = new
        ArrayList<String>();
    enum Color {BLUE, GREEN, RED, YELLOW};

    public DubiousStuff(){
        names.add("Bob");
        names.add("John");
        names.add("Mary");
    }

    public String callMe(String paramName){
        Color myColor = Color.YELLOW;

        if (paramName.length() < 3){
            return null;
        }
        String allNameString = paramName;
```

```
try {
    for (int i=0; i<=names.size(); i++){
        if (paramName != names.get(i)){
            allNameString = allNameString
                + " " + names.get(i);
        }
    }
} catch (Exception e) {}

switch (myColor){
    case BLUE:
        allNameString = allNameString
            + " Blue";
    case GREEN:
        allNameString = allNameString
            + " Green";
        break;
    case RED:
        allNameString = allNameString
            + " Red";
}
return allNameString;
}
```

... And which non-functional requirements may be at risk? *Ingenuity for life*

```
/**
 * This class does some nonsense in a
 * dubious way.
 */
public class DubiousStuff {

    ArrayList<String> names = new
        ArrayList<String>();
    enum Color {BLUE, GREEN, RED, YELLOW};

    public DubiousStuff(){
        names.add("Bob");
        names.add("John");
        names.add("Mary");
    }

    public String callMe(String paramName){
        Color myColor = Color.YELLOW;

        if (paramName.length() < 3){
            return null;
        }
        String allNameString = paramName;
```

Reliability

```
try {
    for (int i=0; i<=names.size(); i++){
        if (paramName != names.get(i)){
            allNameString = allNameString
                + " " + names.get(i);
        }
    }
} catch (Exception e) {}

switch (myColor){
    case BLUE:
        allNameString = allNameString
            + " Blue";
    case GREEN:
        allNameString = allNameString
            + " Green";
        break;
    case RED:
        allNameString = allNameString
            + " Red";
}
return allNameString;
}
```

Security, Reliability

**Reliability,
Maintainability**

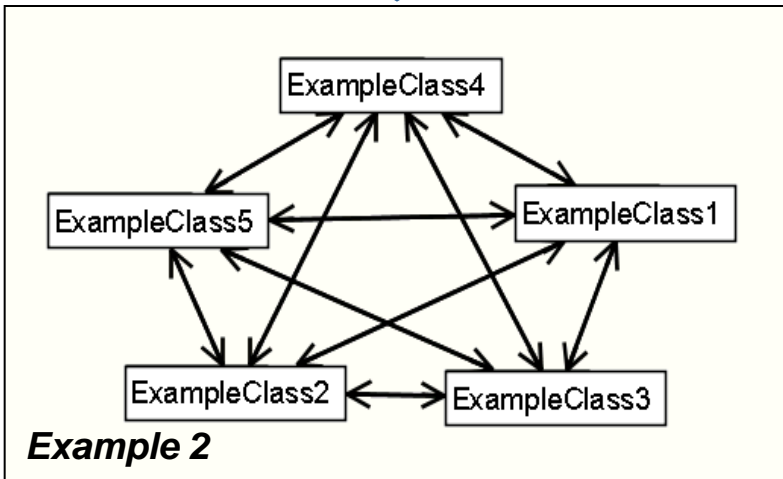
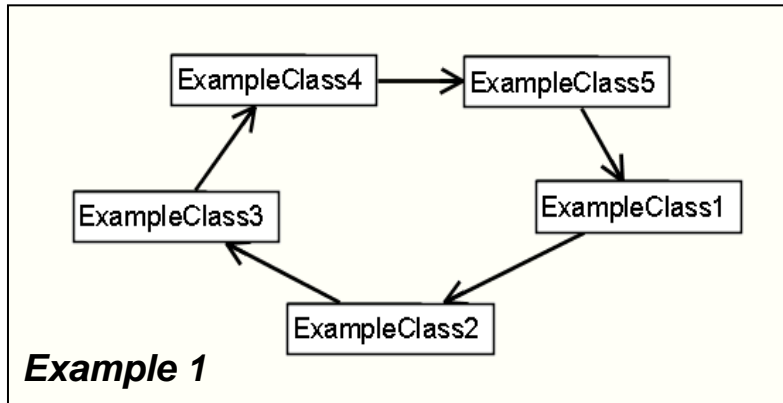
Reliability

Maintainability

Maintainability

Another example

Which issues might be raised by *Example 1*?



- How long does it take until **Example 1** emerges to **Example 2**?
- What is the purpose of the classes and how do they collaborate?
- What are the consequences of changes?
- How can unit tests be implemented without simulating / stubbing all the rest (bad testability)?
- What is the impact on the development process if these classes were developed by distributed teams?

Another example

Which issues might be raised by *Example 1*?

How long does it take until *Example 1* emerges to *Example 2*?

- Typically this goes faster than expected.
- Root cause (class cycle) is the same in both examples. But while *Example 1* can probably be fixed easily, breaking the cycle in *Example 2* is almost impossible.

What is the purpose of the classes and how do they collaborate?

- Understanding the purpose of classes and their interaction can become rather difficult. In some projects there exist class cycles with 50 and more classes, which makes getting into them a nightmare, for example for new team members.

What are the consequences of changes?

- Especially when changing the interface: Imagine adding an additional parameter to one of the class constructors. Which classes might be affected directly and indirectly and must be changed as well?

How can unit tests be implemented without simulating / stubbing all the rest?

- It is difficult to identify "units" if everything is closely related (bad testability).

What is the impact on the development process if these classes were developed by distributed teams?

- Separation of responsibilities for artifacts among distributed teams is difficult.
- If this were subsystems instead of classes: Possible influence on build process (broken build, build time).

(Test) Code quality is relevant on different levels

Micro level

- Code level: Naming conventions, layout, risky statements, ...

Macro level

- For example class design, best practices on higher level, ...

Architecture level

- Structure-oriented: Layers, components, usage of interfaces, coupling, ...

... And which non-functional requirements may be at risk? **SIEMENS** Ingenuity for Life

```
/**
 * This class does some nonsense in a
 * dubious way.
 */
public class DubiousStuff {

    ArrayList<String> names = new
        ArrayList<String>();
    enum Color {BLUE, GREEN, RED, YELLOW};

    public DubiousStuff() {
        names.add("Bob");
        names.add("John");
        names.add("Mary");
    }

    public String callMe(String paramName) {
        Color myColor = Color.YELLOW;

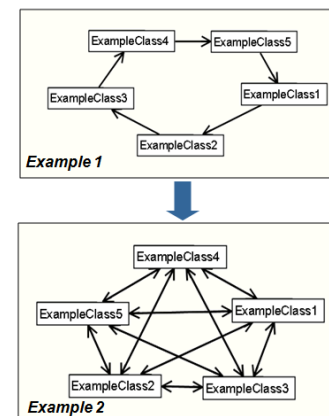
        if (paramName.length() < 3) {
            return null;
        }
        String allNameString = paramName;
    }
}
```

Security, Reliability
Reliability, Maintainability
Reliability
Maintainability
Maintainability

```
try {
    for (int i=0; i<names.size(); i++){
        if (paramName != names.get(i)){
            allNameString = allNameString
                + " " + names.get(i);
        }
    }
} catch (Exception e) {}

switch (myColor){
    case BLUE:
        allNameString = allNameString
            + " Blue";
    case GREEN:
        allNameString = allNameString
            + " Green";
        break;
    case RED:
        allNameString = allNameString
            + " Red";
}
return allNameString;
}
```

Another example
Which issues might be raised by **Example 1**?



- How long does it take until **Example 1** emerges to **Example 2**?
- What is the purpose of the classes and how do they collaborate?
- What are the consequences of changes?
- How can unit tests be implemented without simulating / stubbing all the rest (bad testability)?
- What is the impact on the development process if these classes were developed by distributed teams?

Micro level – What might be the problem?

Problematic code

- Unspecific ("any") exception caught
- Empty catch block

Possible consequences

- No differentiation between different types of exceptions
- Problematic system state is not visible but remains unnoticed (bad testability)
- Reliability endangered

```

▪  /**
▪   * This class does some nonsense in a
▪   * dubious way.
▪   */
▪   public class DubiousStuff {

▪       ArrayList<String> names = new
▪       ArrayList<String>();
▪       ...
▪       public String callMe(String
▪       paramName){
▪           ...
▪           String allNameString = paramName;
▪           try {
▪               for (int i=0; i<=names.size();
▪               i++){
▪                   if (paramName != names.get(i)){
▪                       allNameString = allNameString
▪                       + " " + names.get(i);
▪                   }
▪               }
▪           } catch (Exception e) {}

▪       return allNameString;
▪       }
▪       }

```

Micro level – What can help? (1)

Coding guidelines

Coding guidelines

- Describe the code quality goals in a way that developers can understand
- Show how the quality goals can be implemented in a concrete way
- Communicate best practices in an easy and effective way
- Must be checked to have them followed

Rule (mandatory): Dealing with exceptions

Catch blocks must not be empty. If an exception has been caught, the code must either do something sensible to deal with the detected problem or at least create a logging message.

Tool reference: Checkstyle – EmptyBlock (using LITERAL_CATCH)

Micro level – What can help? (2)

Monitor the current status

Checkstyle Results

The following document contains the results of Checkstyle. [XML](#)

Summary

Files	Infos	Warnings	Errors
283	0	4276	43

Files

Files	I	W	E
org/apache/jmeter/DynamicClassLoader.java	0	12	0
org/apache/jmeter/JMeter.java	0	31	1
org/apache/jmeter/NewDriver.java	0	16	0
org/apache/jmeter/ProxyAuthenticator.java	0	2	0

Checkstyle Results - Microsoft Internet Explorer provided by CIO - INAC V3.195

Adresse: D:\CQM\Projekte\Qualipso\Activity_5\WP_5_5\QualityConcept\tools\Checkstyle-4.4\JMeter\target\site\checkstyle.html

Links: Windows Marketplace

RedundantModifier	111	Warning
AvoidNestedBlocks	0	Warning
EmptyBlock	57	Warning

- tokens: "LITERAL_CATCH"
- option: "text"

Checkstyle Results - Microsoft Internet Explorer provided by CIO - INAC V3.195

Adresse: D:\CQM\Projekte\Qualipso\Activity_5\WP_5_5\QualityConcept\tools\Checkstyle-4.4\JMeter\target\site\checkstyle.html

Links: Windows Marketplace

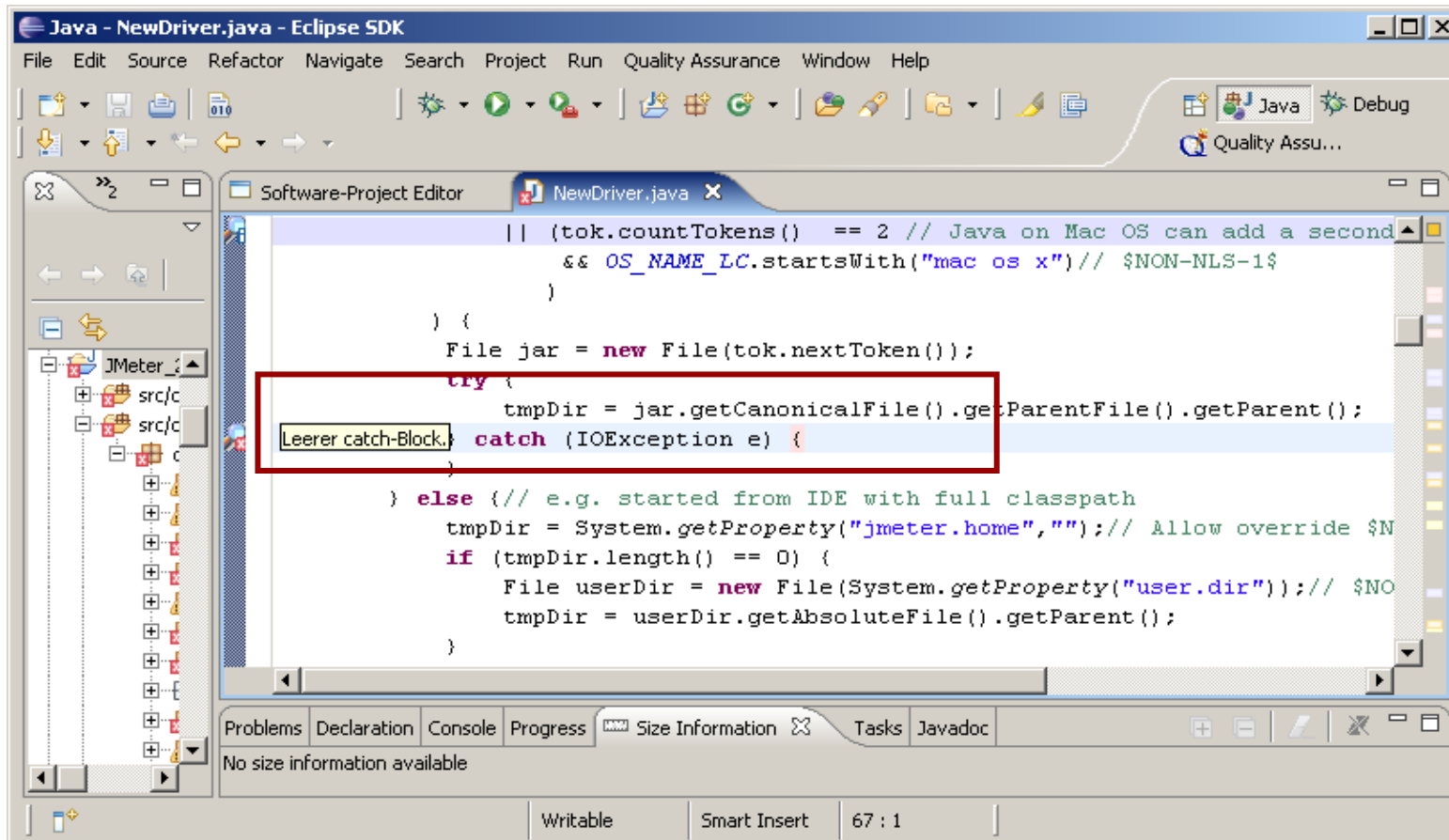
org/apache/jmeter/NewDriver.java

Violation	Message	Line
Warning	Name 'loader' must match pattern '^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*\$'.	47
Warning	Name 'jmDir' must match pattern '^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*\$'.	50
Warning	Line is longer than 122 characters.	60
Warning	Empty <code>catch</code> block.	67
Warning	Line is longer than 122 characters.	87

- Monitoring of coding guideline violations allows early insights about code quality status
- Static code analysis tools (code checkers) help to identify possible problems
- Rule sets can be adapted to project-specific needs
- Improvement activities can be triggered early enough to be effective
- Broken coding guidelines are often an indicator for code of dubious quality in general

Micro level – What can help? (3)

Support developers in complying to the rules



- IDE plug-ins allow the detection of possible problems already *during* coding
- Fixes can be done straightaway and locally

Macro level – What might be the problem? (1)

```
// Horrible abuse of exceptions. Don't ever do this!  
  
try {  
    int i = 0;  
    while (true)  
        a[i++].f();  
} catch (ArrayIndexOutOfBoundsException e) {  
}
```

Source: Joshua Bloch, Effective Java, Item 39

→ Ignores best practice: **Use exceptions only for exceptional conditions**

- Here an exception is used to terminate the loop perhaps in the misguided attempt to improve the performance.

Macro level – What might be the problem? (2)

Macro level problems are usually design problems

- **Class design**
 - Encapsulation: Accessibility of classes and members is not minimized
 - Inheritance is used too often instead of favored composition
- **Usage of patterns**
 - Dangerous implementation of the Singleton pattern
 - Inadequate use of the Model-View-Controller pattern
- **Usage of libraries**
 - Project-specific conventions on how to do tracing and logging are not followed
 - (Parts of) libraries are reimplemented instead of reused

Macro level – What can help?

Macro level issues are difficult to identify solely by the help of tools

→ This area requires **special attention** and activities by the architect

In many cases **knowledge about the semantics** is required

→ (Manual) reviews are absolutely needed

But there is some overlap with the micro level and architecture level;
that means some of the tools mentioned there are helpful, for example:

- Static code analysis tools (code checkers): Low-level best practices ...
- Architecture analysis tools: Size and complexity metrics

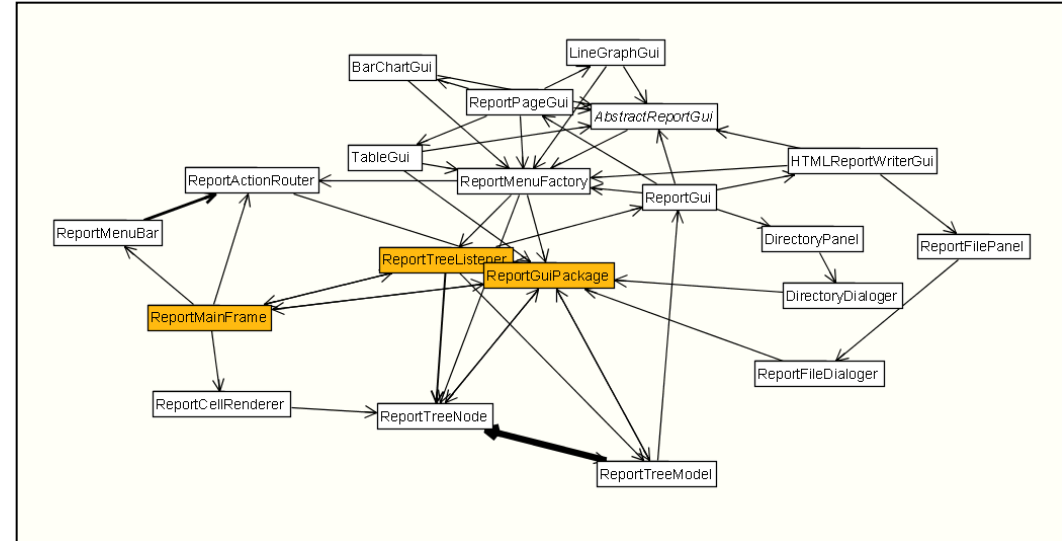
Architecture level – What might be the problem?

Problematic code

- Class cycle with 20 participants
 - 3 classes with more than 25 methods
- Indicators for system degeneration

Possible consequences

- Understanding the purpose of the classes and their interaction
- "Isolating" individual classes for independent unit tests (testability)
- Estimating the effect of changes
- Maintainability endangered



Architecture level – What can help? (1)

Coding guidelines?

Are coding guidelines an adequate means to counteract these issues?

- Is it possible to define rules in a clear and consistent way?
- Does it make sense to define rules anyway or are the underlying best practices too fuzzy?
- Is it possible to identify meaningful and absolute metrics which could be checked by the help of tools?

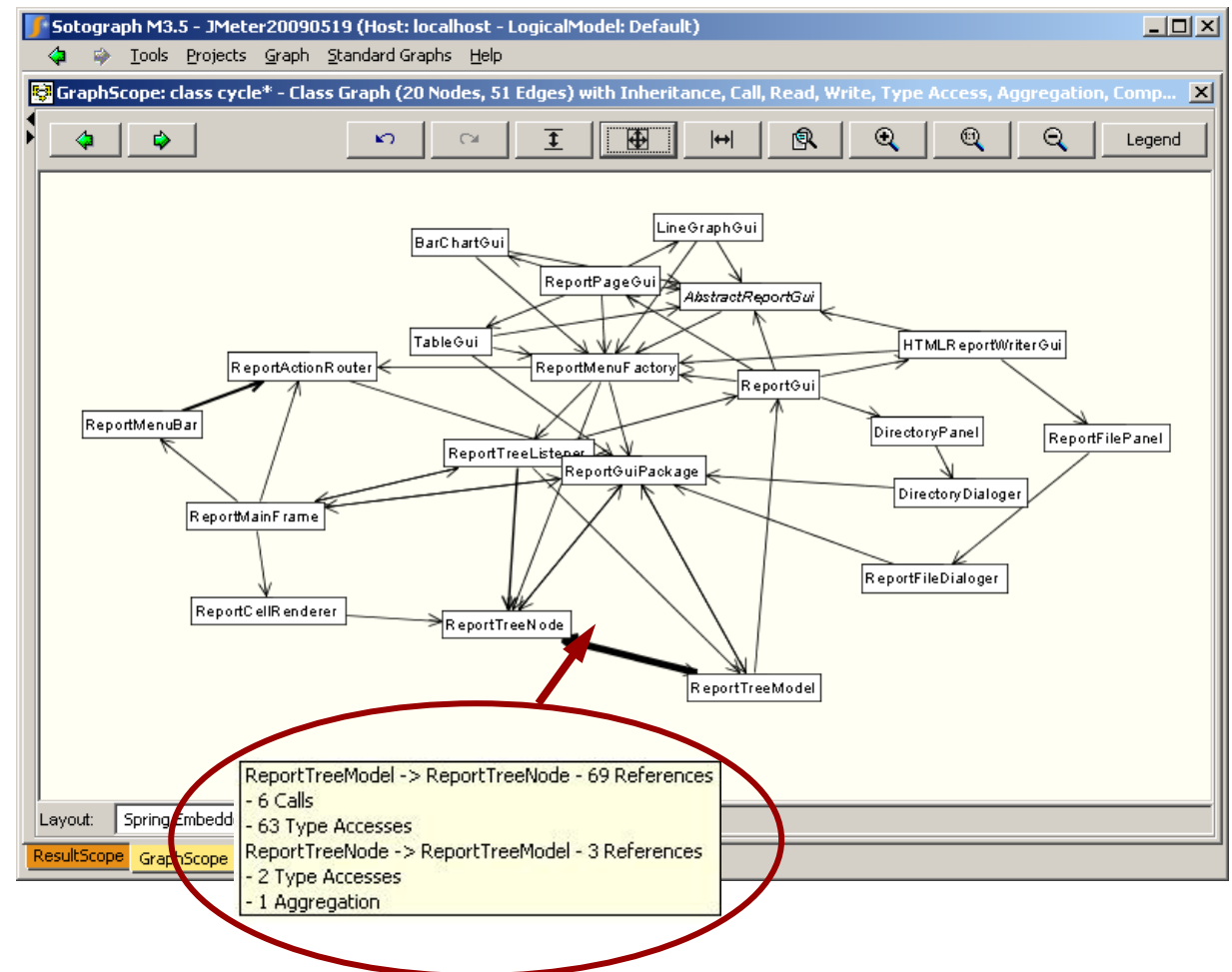
What other ways could be helpful to spread these kinds of best practices in the development team?

- Pair programming
- Code reviews
- Training
- Coding problem of the week
- Posters ("Code Quality Golden Rules")

Architecture level – What can help? (2)

Identifying improvement strategies

- Breaking cycles usually not a local decision but several classes and responsible developers involved
- Architect involvement is needed for finding best locations for breaking the cycle
- Tools can help to identify good candidates for that

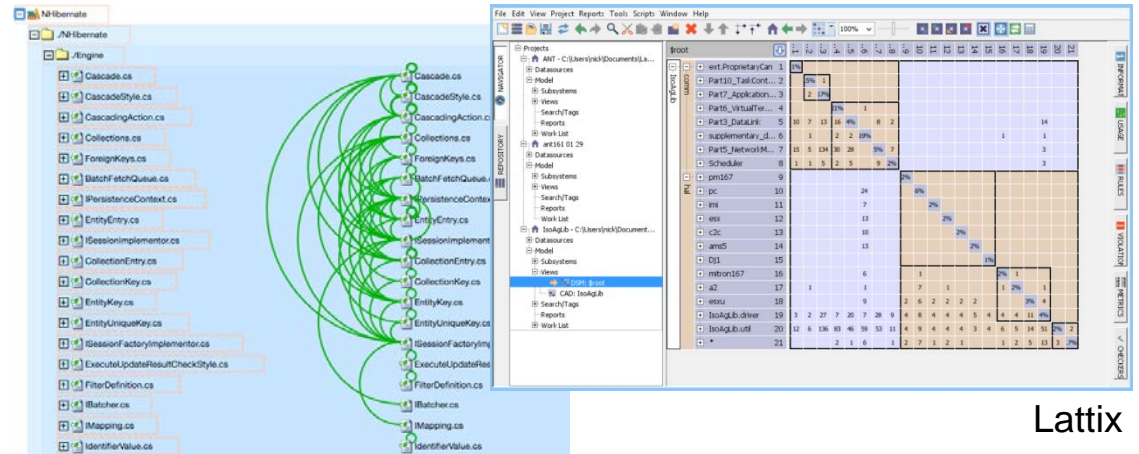


Architecture level – What can help? (3)

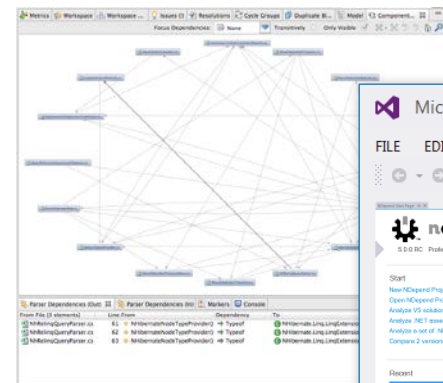
Tool-based architecture analysis with CQM tools

Tools can be used for:

- Checking architectural conformance
- Identification of suspicious components and trends (software aging)
- Analyzing the impact of design changes or design alternatives
- Code comprehension



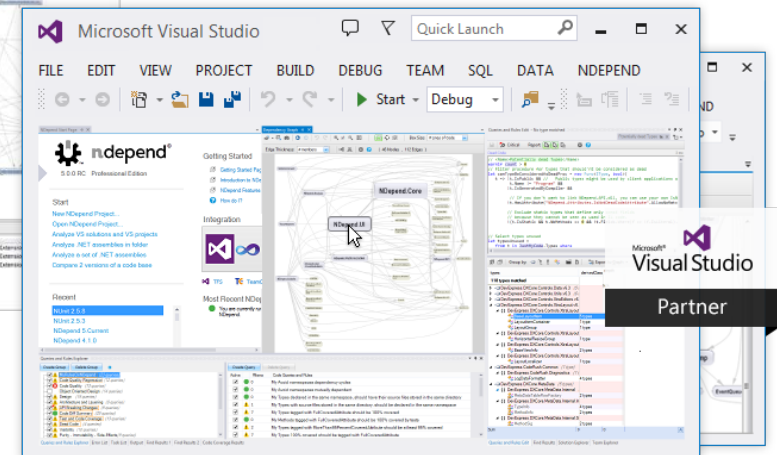
Sonargraph (hello2morrow)



ndepend

Choice of an **adequate tool** depends on **project-specific goals**

Siemens Corporate Technology Code Analysis Wiki
http://manatee.mchp.siemens.de/CA/Main_Page



Fostering *test* code quality culture

Insistence on code quality should be part of every project culture
***Test* Code quality is not only the job of developers and testers, but in fact an important task for the software/system architects and test architects**

Communication is a crucial aspect

- Involve all concerned roles when defining the rule set
- Convince the involved parties of the benefits CQM offers
- Provide everyone with the needed information in an adequate form
- Consider sensitivities when handling assessment results
- Cultivate a culture of trust to increase acceptance

Adapt coding guidelines and tooling to projects' needs

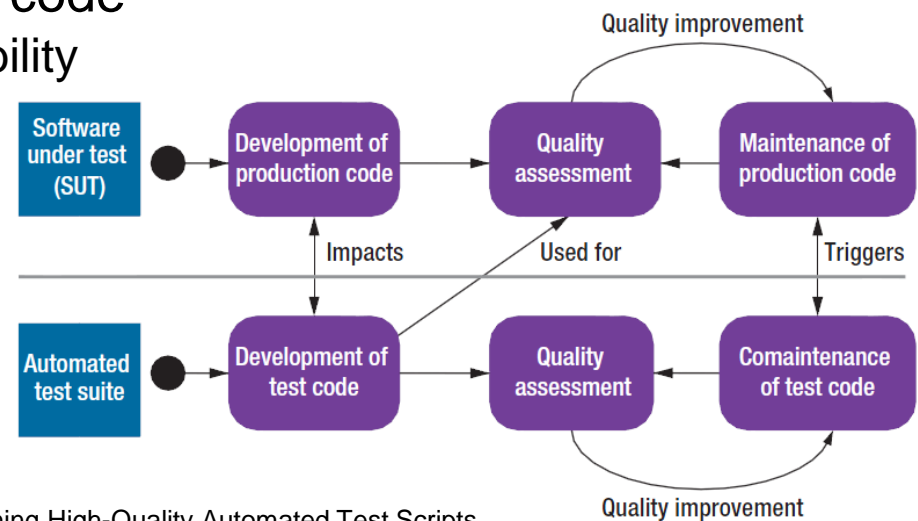
Tools and automation are needed to realize *test* CQM efficiently in practice



Software Test Code Engineering (STCE)

End-to-end test script engineering and test script management

- Use (test) patterns as guidelines to ensure quality
- Functional-quality attributes of test code
 - Correctness in properly testing the SUT
 - Effectiveness in fault detection (→ assess and verify test suite quality)
 - If the test case fails, does the SUT really have a fault?
 - If the SUT has a fault, does the test suite detect it?
- Nonfunctional-quality attributes of test code
 - Maintainability, understandability, readability
 - Reliability
 - Test smells, e.g. test redundancy
- Comaintenance
 - Test antipatterns
 - Determine test case sensitivity
 - Minimize coupling with SUT



Reference: Vahid Garousi, Michael Felderer: Developing, Verifying, and Maintaining High-Quality Automated Test Scripts
IEEE Software, Vol. 33, No. 3, May / Jun 2016: 68-75

Involvement of a test architect in internal software quality and *test* code quality management

Understand

- Get to know the importance of internal software quality and *test* CQM
- Identify coding rules and best practices that support project-specific quality goals

Analyze and assess

- Evaluate compliance with coding rules – Monitor the quality status from the beginning
- Check whether potential problems indicated by analysis tools are real problems that have to be fixed
- Detect indicators for the degeneration of the test system structure

Contribute and drive

- Select adequate tools – aligned with those used for system development
- Foster *test* code quality culture in the development and test teams
- Define and trigger improvement activities

Internal Quality

Agenda

Internal Quality / External Quality / Quality in Use

Understand and Drive Internal Quality

Understand and Manage Technical Debt

(Test) Code and Architecture Quality Management

Summary

What we have learned

Internal quality and technical debt have major impact on the operation and evolution of a system and must be actively managed.

Technical debt is a financial metric that allows to communicate with non-technical stakeholders about systems' internal quality.

Sustaining internal quality needs a lot of discipline and must be part of daily's development work.


The Test Architect is the protector of the quality of the SUT (production code) and of the test system (test code).



Further readings

Use the SSA Wiki :
<https://wiki.ct.siemens.de/x/fReTBQ>

and check the “Reading recommendations”:
<https://wiki.ct.siemens.de/x/-pRgBg>

- 
- **Architect's Resources:**
 - Competence related content
 - Technology related content
 - Design Essays
 - Collection of How-To articles
 - Tools and Templates
 - Reading recommendations
 - Job Profiles for architects
 - External Trainings
 - ... more resources