Software Engineering

Software Quality
Software Quality

• Software Quality Factors

B. Meyer; Object-oriented software construction; Prentice Hall, 1997
We distinguish between **internal** and **external software quality factors**.

- The internal quality factors can only be perceived by computer professionals.
- The external quality factors are ultimately the relevant ones, as they are perceived by the user.

However, the external quality factors depend on the internal quality factors.
We distinguish between internal and external software quality factors.

- Correctness
- Robustness
- Extendibility
- Reusability
- Compatibility
- Efficiency
- Portability
- Ease of use
- Functionality

The user encompasses all stakeholders:
- the owner,
- the “end user”,
- the administrator,
- ...
Correctness is the ability of software products to perform their tasks as defined by their specification.

• To achieve correctness a precise requirements definition is needed

• Correctness is usually only conditional - we guarantee the correctness of our program on the assumption that the lower layers - upon which our product is built - are correct (E.g. we assume that a processor calculates correctly, that the compiler compiles our program correctly, …)
Robustness is the ability of software systems to react appropriately to abnormal conditions.

• Robustness characterizes what happens “outside of the specification”
• Robustness complements correctness
Extendibility characterizes the ease of adapting software products to changes of the specification.

- Important principles to achieve extendibility:
  - Design simplicity
    A simple architecture is easier to adapt.
  - Decentralization
    Autonomous modules (modules which have minimal coupling to other modules → Software Engineering Design & Construction) are easier to change.

Change is pervasive in software development.
• **Reusability** is the ability of software elements to serve for the construction of many different applications

• **Compatibility** is the ease of combining software elements with others

• **Portability** characterizes the ease of transferring software products to various hardware and software environments (i.e., porting it from Android to iOS; porting it from Windows to Linux,...)
Efficiency is the ability of a software system to place as few demands as possible on hardware resources.

- Resources are the processor time, the space occupied in internal and external memories, the bandwidth used in communication devices, ...
- Always try to use “good” algorithms over “bad” ones, because a computer that is twice as fast as a previous model can handle problem sizes near 2*N if the algorithm’s complexity is O(n). Do ask yourself: If the complexity is O(2^n) a computer that is twice as fast can handle problems of size?

Do not worry how fast it is unless it is also right! Efficiency nearly always have to be balanced with other goals.
Functionality characterizes the extent of possibilities provided by a system.

• Avoid featurism; remain consistent with existing features if you add new ones
Ease of Use is the ease with which people of various backgrounds and qualifications can learn to use software products and apply them to solve problems.
Software Quality

• ... or the lack thereof.
Software failures can be disastrous.

• **Therac-25**
  People died due to an overdose of radiation (1985)

• **Ariane 5**
  A system from Ariane 4 was reused but the specification was ignored (1996)

• **Mars Climate Orbiter**
  There was some confusion about the units (i.e. metric system or English system) that are used (1999).

• ...
Software failures can be disastrous.

• hessische Schulsoftware LUSD
  "just" unusable (2007)

• ...
Lack of software quality.

- CampusNet error message shown to the end user (2010)

**Magic uniPaaS Partitioning Message**

*Error: "The Requester could not connect to the Enterprise Server" (-109)*

<table>
<thead>
<tr>
<th>Enterprise Server</th>
<th>cmapp1.cn.pvww.tu-darmstadt.de/3300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>CampusNet</td>
</tr>
<tr>
<td>Program</td>
<td>ACTION</td>
</tr>
</tbody>
</table>
Error creating task - java.io.IOException: StoreElement release error -
de.espirit.firstspirit.server.storemanagement.ReleaseFailedException: page 'index_17'
(ID=722542) of pageref 'tatjana_korbmaecher' (ID=722547) is never released

... hatte in der Nacht ein Software-Update ausgeführt. Im Gefolge davon kam es zu einem Systemabsturz, weil eine Speicherdatei vollgelaufen war....
Missing software quality in commercial software.

- Lufthansa Buchungssystem “Totalausfall” (2009)

...
Missing software quality in commercial software.
Missing software quality in commercial software.

a·bout [əˈbaut] l. prp. 1. um, um... herum;
2. umher in (dat.): wander about the streets;
3. bei, auf (dat.), an (dat.), um, in (dat.): (somewhere) about the house irgendwo im Haus; have you any money about you? haben Sie Geld bei sich?; look about you! sieh dich um!; there is nothing special about him an ihm ist nichts Besonderes;
4. wegen, über (acc.), um (acc.), von: talk about business über Geschäfte sprechen; I’ll see about it ich werde danach sehen od. mich darum kümmern; what is it about? worum handelt es sich?;
5. im Begriff, da’bei: he was about to go out;
6. beschäftigt mit: what is he about? was macht er (da)?; he knows what he is about or weiß, was er tut.
We distinguish between **internal** and **external software quality factors**.

- **Internal quality factors**
  - modular
  - readable
  - ...

This lecture series’ main subject
An example of missing internal quality.

```csharp
/// <summary>
/// Turns true into false and false into true
/// <param name="_booInpt">True or false</param>
/// <returns>False or true</returns>
private bool trueandorfalso(bool _booInpt)
{
    // I'm quite sure though there is a very
clever C# standard command doing this,
// I just can't find it right now ...
    if (_booInpt == true)
        return false;
    return true;
}
```
An example of missing internal quality.

```javascript
/**
 * Checks to see if Australia is typed into the other country box
 */
function checkContactCountry(inputBox)
{
            |(N|n)(E|e)(W|w) (Z|z)(E|e)(A|a)(L|l)(A|a)(N|n)(D|d)$/);

    if(validator.test(inputBox.value))
    {
        alert("Your Residential Address must be outside Australia. 
        + "Enter your residential address outside this country,"
        + "or visit redacted-travel.com.au to make a booking if 
        + "you live in Australia.");
        inputBox.focus();
        inputBox.select();
    }
}
```
Internal quality.

```scala
def isAnnotatedWith(
  classFile: ClassFile,
  annotationTypes: Iterable[ObjectType]): Boolean = {

  var bufferOutput: Iterable[Object] = Iterable.empty
  val runtimeVisibleAnnotations = classFile.runtimeVisibleAnnotations
  val runtimeInvisibleAnnotations = classFile.runtimeInvisibleAnnotations
  for (annotationType ← annotationTypes) {
    bufferOutput = bufferOutput ++ runtimeVisibleAnnotations.filter {
      case Annotation(`annotationType`, _) ⇒ true
      case _ ⇒ false
    }
    bufferOutput = bufferOutput ++ runtimeInvisibleAnnotations.filter {
      case Annotation(`annotationType`, _) ⇒ true
      case _ ⇒ false
    }
  }

  annotationTypes.nonEmpty &&
  !classFile.isAnnotationDeclaration &&
  bufferOutput.nonEmpty
}
```

Where is the issue/are the issues?
Have you ever noticed that **when someone checks in some complex and, oftentimes, horrific piece of code, the check-in is greeted with an almost deafening silence?** [...] 

The explanation for why this occurs was first given by C. Northcote Parkinson [...] He stated that if you were building something complex, then few people would argue with you because few people could understand what you were doing. If you were building something simple [...] which most anyone could build, then everyone would have an opinion.

---

George V. Neville-Neil

*Painting the Bike Shed - A sure-fire technique for ending pointless coding debates; ACM Queue, ACM 2009 1542-7730/09/0600*
If you want to study code with missing quality...

But, reading other people’s code - in particular if the code is good - is one of the best ways to learn to program.
If you want to study code...
Part of the source code for Comanche, build 055. It is part of the source code for the Command Module's (CM) Apollo Guidance Computer (AGC), Apollo 11.

It is often not possible to improve all software quality attributes.

*Sometimes they are at odds.*
Software Quality

• Good Software

Ian Sommerville; Software Engineering - Eighth Edition; Addison Wesley, 2007
Attributes of “good Software”

• Maintainability
  Software should be written in such a way that it may evolve to meet changing needs of customers.

• Efficiency
  Software should not waste system resources; it includes: responsiveness, processing time, memory utilisation, etc.

• Usability
  Software must be usable by the intended users.

• Dependability (dt. Verlässlichkeit)
  Dependable software does not cause physical or economic damage in the event of system failure. Further properties: Repairability, Survivability, Error Tolerance...
Some Aspects of Dependable Systems

Availability = dt. Verfügbarkeit
Reliability = dt. Zuverlässigkeit
Safety = dt. Betriebssicherheit
Security = dt. Systemsicherheit

- **Availability**: The ability of the system to deliver services when required
- **Reliability**: The ability of the system to deliver services as specified
- **Safety**: The ability of the system to operate without catastrophic failure
- **Security**: The ability to protect itself against accidental or deliberate intrusion

**this includes**
- correctness
- precision
- timeliness

Software Engineering; Ian Sommerville; 2007
Software Quality Assurance (SQA)

• Constructive vs. Analytical
Fostering Software Quality by Means of...

Constructive SQA and Analytical SQA

- Programming Languages
- Scalable Static Analyses
- Lightweight Formal Methods
- Software Development Processes
- Software Architecture
- Domain Specific Languages
- Metrics
- Machine Learning
- Type Systems (Language Based Security)

...
Null Values...

- found in `java.nio.file.FileTreeWalker next()`
  
  ```java
  if (ioe != null) {
    ioe = e;
  } else {
    // here, ioe is null
    ioe.addSuppressed(e);
  }
  ```
A Few (Well Known) Static Analysis Tools

• FindBugs
  Lightweight static analyses on top of Java Bytecode.

• PMD
  Lightweight static analyses on top of the AST using Java Visitors or XPath-based rules.

• CheckStyle
  Lightweight static analyses on top of the AST using Java Visitors.

• CheckerFramework
  Static analyses using pluggable types.

• ConQAT
  Code Clone Detection.
FindBugs™ - Find Bugs in Java Programs

This is the web page for FindBugs, a program which uses static analysis to look for bugs in Java code. It is free software, distributed under the terms of the Lesser GNU Public License. The name FindBugs™ and the FindBugs logo are trademarked by The University of Maryland. As of July, 2008, FindBugs has been downloaded more than 700,000 times.

FindBugs requires JRE (or JDK) 1.5.0 or later to run. However, it can analyze programs compiled for any version of Java. The current version of FindBugs is 1.3.9, released on 20:11:47 EDT, 21 August, 2009. We are very interested in getting feedback on how to improve FindBugs.

Changes | Talks | Papers | Sponsors | Support

New

- **JavaOne talk:** [Slides](#) from my JavaOne talk, Mistakes That Matter.

- **FindBugs community review:** We are previewing FindBugs community review, in which anyone can review issues in open source projects (i.e., mark issues as "must fix" or "mostly harmless"), and those reviews are automatically shared with other reviewers.

  This is a pre-beta release, not ready for deployment. The implementation will be undergoing significant changes before general availability.

Initially, we are posting results for:
Software Inspections - Lightweight Static Software Analysis

```java
class PathExtractor {
    public List<String> extractPaths(String path) {
        List<String> paths = new ArrayList<>();
        // Code to extract paths...
        return paths;
    }
}
```
PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, XML, XSL.

Additionally, it includes CPD, the copy-paste-detector. CPD finds duplicated code in Java, C, C++, C#, PHP, Ruby, Fortran, JavaScript.

5.2.1 (3rd November 2014)

- Release Notes
- Download (Sourcecode, Documentation)
- Online Documentation
Overview

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration files are supplied supporting the Sun Code Conventions, Google Java Style.

A good example of a report that can be produced using Checkstyle and Maven can be seen here.

Important Development Changes

As of September 2013, the Checkstyle project is using GitHub for hosting the following:

- **Source code repository** - replacing the Mercurial repository on SourceForge.
- **Issue management** - replacing the Bugs/Feature/Patches on SourceForge. All new issues should be raised at GitHub, and pull requests are now the preferred way to submit patches.

SourceForge will still be used for website hosting and binary hosting for downloads.
A Few (Well Known) Static Analysis Tools

• JDepend
  Structural analysis on top of Java Bytecode.

• DependencyFinder
  Structural analysis on top of Java Bytecode.

• Stan4J
  Structural analysis on top of Java Bytecode.

• Sonargraph (SonarJ)
  Analyzes the structure of applications.
A Few (Well Known) Static Analysis Tools

- ESC/Java2
  Formal verification using JML Annotations.
- Key
  Formal verification.
- ...
<table>
<thead>
<tr>
<th></th>
<th>Reusability</th>
<th>Maintainability</th>
<th>Correctness</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lightweight static analyses</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>↓-○</td>
</tr>
<tr>
<td>Semi formal methods</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>↓</td>
</tr>
<tr>
<td>Formal methods</td>
<td></td>
<td>✓</td>
<td></td>
<td>↑</td>
</tr>
<tr>
<td>Structure analyses</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>↓</td>
</tr>
<tr>
<td>Style conformance checking</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>↓</td>
</tr>
<tr>
<td>Architecture conformance checking</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>○-↑</td>
</tr>
</tbody>
</table>

- ✓ indicates the method provides the quality attribute.
- ○ indicates the method does not provide the quality attribute.
- ↓ indicates a decrease in effort.
- ↑ indicates an increase in effort.
- ○-↑ indicates a mixed effect on effort.
Classifying Found Issues

- **True** and **False** Positives
- **True** and **False** Negatives
- **Irrelevant True** Positives
- **Perceived False** Positives
**True** and **False** Positives

- A **True** Positive is the correct finding *(of something relevant)*

  This is what static analyses should detect.

- A **False** Positive is a finding that is just incorrect

  False positives are typically caused by the weaknesses of the analysis.
Example of a True Positive

Let’s assume that we have a “basic” analysis to detect object accesses (o.xyz) that appear in a guarded context (if (o != null)) and also outside a guarded context.

```java
void printIt(String args[]) {
    if (args != null) {
        System.out.println("number of elements: " + args.length);
    }
    for (String arg : args) {
        System.out.println(arg);
    }
}
```
Example of a **False Positive**

Let’s assume that we have a “basic” analysis to detect object accesses \(o.xyz\) that appear in a guarded context \((if (o \neq null))\) and also outside a guarded context.

```java
guard
void printReverse(String args[]) {
    int argscount = 0;
    if (args != null) {
        argscount = args.length;
    }
    for (int i = argscount - 1; i >= 0; argscount--) {
        System.out.println(args[i]);
    }
}
guarded & unguarded access
implicit guard
```
True and False Negatives

- A **True** Negative is the correct finding of no issue.
- A **False** Negative is an issue that is not reported.

Generally, only relevant in the context of formal approaches.
Irrelevant True Positives

• Irrelevancy is context-dependent…
  • Issues related to Serialization are irrelevant when your application doesn’t use Serialization at all.
  • A violation of the `hashCode-equals` contract may be completely irrelevant for an (inner) class that is never put in a collection that uses hashes.
• …
boolean handleIt(int i) {
    if (i < 0 || i > 2)
        throw new IllegalArgumentException();

    switch (i) {
    case 0:
    case 1:
        return true;
    case 2:
        return false;
    default:
        throw new UnknownError();
    }
}
Perceived False Positives

GeneralPath result = new GeneralPath(GeneralPath.WIND_NON_ZERO);
...
if (dx != 0 || dy != 0) {
    AffineTransform tx = AffineTransform.getTranslateInstance(dx, dy);
    result = (GeneralPath)tx.createTransformedShape(result);
}

This cast will always fail!
GeneralPath result = new GeneralPath(GeneralPath.WIND_NON_ZERO);

... if (dx != 0 || dy != 0) {
    AffineTransform tx = AffineTransform.getTranslateInstance(dx, dy);
    result = (GeneralPath)tx.createTransformedShape(result);
}

public Shape createTransformedShape(Shape pSrc) {
    if (pSrc == null) {
        return null;
    }
    return new Path2D.Double(pSrc, this);
}

interface Shape

class Path2D implements Shape, Cloneable

/*inner*/ class Double extends Path2D implements Serializable
Perceived False Positives

GeneralPath result = new GeneralPath(GeneralPath.WIND_NON_ZERO);
...
if (dx != 0 || dy != 0) {
    AffineTransform tx = AffineTransform.getTranslateInstance(dx, dy);
    result = (GeneralPath)tx.createTransformedShape(result);
}

public Shape createTransformedShape(Shape pSrc) {
    if (pSrc == null) {
        return null;
    }
    return new Path2D.Double(pSrc, this);
}

interface Shape

class Path2D implements Shape, Cloneable

/*inner*/ class Double extends Path2D implements Serializable

Perceived false positives are the result of issue reports related to complex issues and/or related to reports that are not easy to comprehend.
Perceived False Positives

```java
protected Icon getIconForType(int messageType) {
    if (messageType < 0 || messageType > 3) return null;
    String propertyName = null;
    switch (messageType) {
    case 0:
        propertyName = "OptionPane.errorIcon"; break;
    case 1:
        propertyName = "OptionPane.informationIcon"; break;
    case 2:
        propertyName = "OptionPane.warningIcon"; break;
    case 3:
        propertyName = "OptionPane.questionIcon"; break;
    }
    if (propertyName != null) {
        return (Icon)DefaultLookup.get(optionPane, this, propertyName);
    }
    return null;
}
```

javafx.swing.plaf.basic.BasicOptionPaneUI

Dead Code!
protected Icon getIconForType(int messageType) {
    String propertyName = null;
    switch(messageType) {
        case 0:
            propertyName = "OptionPane.errorIcon"; break;
        case 1:
            propertyName = "OptionPane.informationIcon"; break;
        case 2:
            propertyName = "OptionPane.warningIcon"; break;
        case 3:
            propertyName = "OptionPane.questionIcon"; break;
        default:
            return null;
    }
    return (Icon)DefaultLookup.get(optionPane, this, propertyName);
}
Cryptic True Positives

```java
boolean process() throws Exception {
    boolean done = false;
    do {
        Thread.sleep(500);
        done = (System.currentTimeMillis() % 100l == 0l);
    } while (!done);
    return !done;
}
```

Refactored:
```
return false;
```

if (done)
    done = false
else
    done = true
return done
A Holistic View is required.

Software Process Model

... Development Environment
Traceability Tools
Static/Dynamic Analysis Tools
(High-Level) Documentation
Modeling Languages
Domain Specific Languages
Programming Language
Software Quality

• Summary
Does Distributed Development Affect Software Quality?  
An Empirical Case Study of Windows Vista

Christian Bird\textsuperscript{1}, Nachiappan Nagappan\textsuperscript{2}, Premkumar Devanbu\textsuperscript{1}, Harald Gall\textsuperscript{3}, Brendan Murphy\textsuperscript{2}

\textsuperscript{1}University of California, Davis, USA  
\textsuperscript{2}Microsoft Research  
\textsuperscript{3}University of Zurich, Switzerland

\{cabird, ptdevanbu\}@ucdavis.edu \{nachin, bmurphy\}@microsoft.com gall@ifi.uzh.ch
Browser Security: Lessons from Google Chrome

Google Chrome developers focused on three key problems to shield the browser from attacks.

Charles Reis, Google; Adam Barth, UC Berkeley; Carlos Pizano, Google

The Web has become one of the primary ways people interact with their computers, connecting people with a diverse landscape of content, services, and applications. Users can find new and interesting content on the Web easily, but this presents a security challenge: malicious Web-site operators can attack users through their Web browsers. Browsers face the challenge of keeping their users safe while providing a rich platform for Web applications.

Browsers are an appealing target for attackers because they have a large and complex trusted computing base with a wide network-visible interface. Historically, every browser at some point has contained a bug that lets a malicious Web-site operator circumvent the browser’s security policy and compromise the user’s computer. Even after these vulnerabilities are patched, many users continue to run older, vulnerable versions. When these users visit malicious Web sites, they run the risk of having their computers compromised.

Generally speaking, the danger posed to users comes from three factors, and browser vendors can help keep their users safe by addressing each of these factors:

- **The severity of vulnerabilities.** By sandboxing their rendering engine, browsers can reduce the severity of vulnerabilities. Sandboxes limit the damage that can be caused by an attacker who exploits a vulnerability in the rendering engine.
- **The window of vulnerability.** Browsers can reduce this window by improving the user experience.
Take responsibility!

*There are no excuses. If you develop a system, it is your responsibility to do it right. Take that responsibility. Do it right, or don’t do it at all.*

Recall the “fifteen principles of Software Engineering”.
The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

• Software quality is not just about the (internal) quality of the source code.
• Software quality means different things to different stakeholders.
• To produce quality software a holistic view on a software project is required.
• The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.