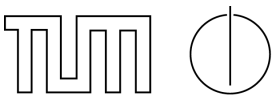
Tutorial for **Software Quality Management**
Summer Term 2010
Dr. S. Wagner
L. Heinemann, S. Islam
27.4.2010



Technische Universität München
Fakultät für Informatik

Tutorial Sheet 1: Introduction

A computer with internet access is required for this tutorial sheet.

Assignment 1: Assessing Maintainability

Download the source code of Jabref, an open source bibliography reference manager written in Java, from the following URL:

http://downloads.sourceforge.net/project/jabref/jabref/2.6/JabRef-2.6-src.zip?use_mirror=kent.

Open the class `net.sf.jabref.EntrySorter` in an editor of your choice and perform a review of the source code. Make a list of issues that in your opinion negatively affect the *maintainability* of the software.

[**Antwort:**

- Line 30: .* import should be avoided
- Line 32: Class should be documented
- Line 34: Commented code should be avoided
- Line 34-38: Fields should be private
- Line 35: Unintuitive identifier name. Better: bibtexEntries
- Line 36: Identifier names should not be abbreviated. Better: comparator, bibtexEntryComparator
- Line 39: Commented code should be avoided
- Line 43: Constructor should be documented
- Line 44: Commented code should be avoided
- Line 53: Commented code should be avoided
- Line 59: Method should be documented
- Line 68: Commented code should be avoided
- Line 87: Unintuitive identifier name
- Line 89: Commented code should be avoided
- Line 98: Method should be documented

- Line 99: No indentation
- Line 102: Method should be documented
- Line 106: Commented code should be avoided
- Line 109: Method should be documented
- Line 115: Method should be documented
- Line 117: If-statement should use curly brackets
- Line 124: Method should be documented
- Line 129-132: Commented code should be avoided
- Line 140: Commented code should be avoided
- Line 147: Commented code should be avoided

]

Assignment 2: Assessing Usability

Jakob Nielsen developed the following ten principles for user interface design:

- *Visibility of system status:* The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- *Match between system and the real world:* The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
- *User control and freedom:* Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
- *Consistency and standards:* Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
- *Error prevention:* Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
- *Recognition rather than recall:* Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- *Flexibility and efficiency of use:* Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
- *Aesthetic and minimalist design:* Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

- *Help users recognize, diagnose, and recover from errors:* Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- *Help and documentation:* Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

(Source: http://www.useit.com/papers/heuristic/heuristic_list.html)

1. Pick two of these heuristics and try to concretize them by giving positive and negative examples of aspects in user interfaces.
2. Name an example a software you use frequently that violates at least one of the principles and explain the violation.
3. Discuss advantages and disadvantages of design heuristics.

[**Antwort:**

1.)
Visibility of system status
An upload button on a web page is enabled, until clicked. Then it is replaced with a progress indicator until the file has finished uploading.

Consistency and standards
Word, Excel, and PowerPoint all use the same style toolbar with the same primary menu options: Home, Insert, Page Layout, etc. Consistency results in efficiency and perceived intuitiveness.

2.)
Flexibility and efficiency of use
In Windows media player, it can require 6 clicks to sort the list of songs descending by rating, which can be considered a rather common operation. In iTunes this can be achieved with at most 2 clicks. (Source: http://www.usabilitybugs.de/)

3.)
Pro: Expert knowledge is captured
Con: High-level guidelines often need interpretation/concretization to context

]