MEDIZINISCHE FAKULTÄT ♦ RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

INSTITUT FÜR MEDIZINISCHE INFORMATIK

GESCHÄFTSFÜHRENDER DIREKTOR: UNIVERSITÄTSPROFESSOR DR. DR. KLAUS SPITZER

The CERES Project

CERES Software Development Guidelines

Lutz Ißler

Nicolas Becker

Cord Spreckelsen

Christa Weßel

Volume 2, Band 1, Februar 2007

ISSN 1860-8906

ISBN 978-3-9810089-5-1

Aachener Schriften zur Medizinischen Informatik

Aachener Schriften zur Medizinischen Informatik
ISSN 1860-8906
ISBN 978-3-9810089-5-1

Herausgeber: Institut für Medizinische Informatik der RWTH Aachen
Pauwelsstr. 30
D-52074 Aachen

Geschäftsführender Direktor: Universitätsprofessor Dr. Dr. Klaus Spitzer

Contents

1 Introduction **3**

2 Definition of terms **4**

3 Preamble **5**

4 Applicability **5**

5 The guidelines **6**
5.1 Correctness 6
5.2 Reliability and efficiency 6
5.3 Integrity 7
5.4 Usability 7
5.5 Copyright and Licenses 7
5.6 Documentedness 8
5.7 Maintainability 9
5.8 Flexibility 10
5.9 Portability 10
5.10 Testability 10
5.11 Reusability 10
5.12 Interoperability 11

6 Development Course **12**

References **13**

1 Introduction

In a medical informatics research project, there are usually people from different disciplines working together. Whilst the scientific work in such a project is done according to well-known standards, there is no common procedure to help those project members who develop software. The resulting software is often a patchwork consisting of software modules with many architectural styles, modularization concepts, and degrees of maturity, which comes from the typically very different skill levels of the developers.

This led to the introduction of the CERES Software Development Guidelines in the CERES Project. The project works on the research on a web-based information system on hospital related data [WKI+3b], [WWS06]. The goal is to offer ubiquitous, up-to-date and valid information about hospitals, their structure, their services and their results. The users - citizens and staff members of hospitals, health insurance companies, institutions of the government and professional medical associations - shall be able to look on different hospitals at a glance on one platform. The central, object-oriented database is based on a meta-model of "the German hospital". The user is able to access this database on runtime by different web-frontends, which are for example "Search", "Tables", "Texts" and "Map".

The multi-disciplinary project team consists of scientists and students of computer science, medicine, public health and economics. The tools and modules of the web-based information system on hospital related data are mostly the result of student's research projects and diploma theses.

The first author of the guideline designed the guideline based on literature work and iterative feedback of the team members in 2004. The author and the team checked the guidelines on their appropriateness and revised them if necessary six-monthly. The second author took over to proceed with the maintenance of the guidelines from June 2005 on. The two senior scientists of the project guided the development and maintenance of the guidelines.

2 Definition of terms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this section are to be interpreted as described in RFC 2119[1].

The following is cited from RFC 2119:

MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

[1] Request For Comments No. 2119. See http://www.faqs.org/rfcs/rfc2119.html

3 Preamble

This guideline document MUST be revised every six months by the CERES project team. In the revision, the project team should have a closer look on specifications on the used software and discuss wether or not to update these specifications in the guidelines. Some footnotes include information that has to be observed for forthcoming revisions.

The steps of the revision MUST be documented in section 6 of this guideline document (Stage of Affairs).

Criteria regarding the current server environment are documented in the CERES Tomcat Tutorial.

4 Applicability

The guidelines apply to all software developed as a part of the CERES project. Mainly, "all software" are all servlets running on the CERES project server, and the modelling tool ZEUS.

Every developer MUST follow the guidelines. This refers also to the re-engineering of software that was not developed according to the guidelines.

Software developed for throw-away purpose[2] MAY be developed according to the guidelines.

[2] A German circumscription would be "Testballon".

5 The guidelines

The guidelines were developed according to the software quality criteria defined in the IEEE 610.12-1990 standard.

The following is called the "CERES software quality maxime" and forms the foundation of all software quality management in the CERES project:

> A piece of software in CERES achieves minimum "software quality"
> iff
> all MUST criteria from the guidelines are fulfilled.

5.1 Correctness

- The author MUST create a requirements specification document for the software. The requirements specification document ("specification" for short) MUST list the scientific goals, the functional requirements and the nonfunctional requirements. The specification MAY (for applications that are intended to interact with end-users: SHOULD) contain a description of several scenarios in which the software is used in the intended manner to solve problems.
- The author MUST present the specification to the project group. The project group SHOULD give feedback to the author, and the author MUST overwork the specification according to the feedback.
- The author MUST report any changes of the specification to the project group, and the author MUST document the changes along with the rationals for the changes, in the specification.
- The author (supported by at least one project group member) MUST perform an software acceptance test[3]. In this test, it is checked wether the software fulfills all functional and nonfunctional requirements stated in the specification. The author MUST write a test report and present the report to the project group.

5.2 Reliability and efficiency

- The author MUST state the following criteria as nonfunctional requirements: the usual usage environment for the software (hardware, software, internet connection speed, etc.), the usual number of users working with the software at the same time, the usual amount of data processed with the software, and the expected response times for the user interaction.

[3] German: Abnahmetest

- If the software has to fulfill any performance requirements, the author SHOULD verify the conformance to these requirements on an environment that represents the intended target computer. If no such system is available, the author MUST document the characteristics of the used test system (hardware and software resources etc.).

- The author MUST provide a mechanism to measure wether the software fulfills the reliability criteria. The author MUST substantiate the suitability of this mechanism. For example, a mechanism that documents the stable running of a servlet, which processes requests from 5 users for 24 hours, may be appropriate to demonstrate the reliability of this servlet.

- The author MUST ensure that only as much data as needed is requested from or transferred to the database or other data sources. In general, that means to transfer only data that is required to complete the current task.

  Example 1: It is not necessary to retrieve the whole extension of a class from the database when only one object of the class is needed for the further processing.

  Example 2: The decision to store JAVA objects to a file in the direct way by the serialization features of the JAVA SDK will store a huge overhead of data compared to a individually developed XML storage of the objects. The developer has to measure how to achieve a efficient implementation without an inadequate raise of the development time.

5.3 Integrity

- All servlets MUST be derived from the class `CeresServlet`[4].

- Every software accessing the database MUST use the CERES toolkit for this access.[5]

5.4 Usability

- All CERES web applications MUST follow the CERES Web Application Style guide.

5.5 Copyright and Licenses

- For every third-party library or component, the CERES developer inventing the usage of the library MUST document which license applies to the used library in

[4] During the revision process of the CERES Software Development guidelines it must be continuously checked that `CeresServlet` is still a generic class for the CERES Project.

[5] The usage of the CERES toolkit is required because the toolkit implements an access control layer for the database, and because the toolkit abstracts from the used database.

the central "CERES used components: licenses and sources of supply" document.

- If a particular software needs a third-party library, the author MUST ensure this library is available at least as free software or, if no free library for the particular functionality exist, obtain the permission of the project group to use a non–free library.
- If the development of a particular software requires a specific development environment, the author MUST
  - ensure the development environment is available at least as free software, or
  - ensure the institute owns a license for the development environment, or
  - obtain the permission of the project group to use a non–free development environment.
- The usage of free software with proprietary licenses (eg. the GNU GPL[6]) is acceptable, although CERES is not published as free or open source software, because CERES is currently in an experimental state. In the case CERES is deployed and either available by download or purchase, the project group has to reconsider all license questions.

5.6 Documentedness

- Every identifier declared as `public` or `protected` MUST be documented with a Javadoc comment.
- Every complex method, every class, and every package MUST be documented with a Javadoc comment that contains at least an introducing explanation of the complex functionality.
- The author MUST clarify the structure of the source code and the general idea of the developed algorithms in his thesis, a tutorial or the CERES Project Documentation. The author MAY use comments to clarify the structure of the source code and the general idea of the developed algorithms to other developers reading the source code[7].
- Every software MUST be accompanied by a short tutorial that explains the main program logic in order to support programmers that maintain or extend the software in the future. This tutorial MUST contain at least one class diagram (formulated in the Unified Modeling Language) that outlines the software architecture.

[6] GNU General Public License

[7] This does neither mean the obligation to comment every obvious statement nor the right to abandon comments at all.

- Every software MUST be accompanied by a list of required third-party libraries or components. For each library or component, it MUST be stated where to get it from and how to install it.
- Every servlet MUST be accompanied by a list of resources it may access during runtime.

5.7 Maintainability

- Every software SHOULD separate program logic and input/output. That is, every software SHOULD follow the intentional spirit of existing design patterns[8], for example the model-view-controller design pattern[9].
- The author MUST test the software in a systematic and reproducible manner with the objective of verification. The author MUST identify relevant and meaningful test data and present the test design to the project group who supports the author to set it in an adequate form. If the software is intended to run on the project server, the author must first test it on a separate runtime environment and then on the project server. For example, JUnit test case classes[10] or a cognitive walkthrough could be appropriate for testing.
- All classes developed for a special tool within the CERES project MUST be contained in a package named "ceres.(toolname)".
- All extensions developed for ZEUS SHOULD be contained in a package named "ceres.zeus.modules.(toolname)".
- Every developer of a servlet MUST separate between Java class files, static files that are served by the web server[11] and additional files that do not belong to the former. The Tomcat Tutorial (section **??**) specifies how to achieve this.
- Every developer MUST follow the JAVA Code Conventions[12], with one exception: use only tab stops for indention.
- Every developer MUST follow the well-known modular programming paradigm.[13]
- Every programmer SHOULD make excessive use of the mechanisms the JAVA programming language provides to support maintainability and reusability. In particular, these are the language constructs for packages and interfaces, and modifiers for identifier visibility.

[8] See [GHJ95]

[9] See http://www.exciton.cs.rice.edu/JavaResources/DesignPatterns/MVC.htm for an explanation of the model-view-controller pattern.

[10] If you have to invent tests on whole servlets or other special types of classes, consider the use of a specific JUnit extension for that type.

[11] These are static files that must be accessed by the user.

[12] See http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

[13] For a reference on this, see standard literature on object-oriented programming and software design.

5.8 Flexibility

- Every developer MUST document the intended purpose of the developed software. The documentation MUST include the original purpose and SHOULD try to draw an exact outline of the purpose in order to establish a border to other modules.

5.9 Portability

- All software in the CERES project except for servlets MUST be developed using the Java SDK 5.0[14] (Java 1.5.0). In order to avoid compatibility problems, servlets MUST be developed using Java SDK 1.4.2[15]. Every Software MUST be testet for compatibility if the given Java version changes.
- All servlets MUST support the Servlet API, version 2.4[16].
- As all CERES web applications MUST follow the CERES Web Application Style guide, all CERES web applications MUST also be developed respecting the browser requirements specified there.
- All software in the CERES project MUST NOT use absolute pathnames or hard-coded URLs, port numbers etc., if avoidable. Instead, the software SHOULD use relative pathnames or obtain such values from its context[17].
- All software in the CERES project MUST NOT contain any pathnames (absolute or relative), URLs, port numbers etc. hardcoded in the source. Instead, servlets MUST define constants for such values in the class `ceres.Configuration`. All other software except for servlets MUST define constants for such values in a configuration file.

5.10 Testability

- A class SHOULD NOT contain a single method that does all work. In most cases, such a class will be the result of disregarding the object oriented programming paradigm, and therefore leading to a bad testability. However, sometimes such a method `doEverything()` cannot be avoided (eg. in servlet development).

5.11 Reusability

- The reusability of a software is the direct result from its Maintainability and Portability. Confer the respective section for further explanations.

[14]The Java version recommended here might change with the next guideline revision.
[15]The Java version recommended here might change with the next guideline revision.
[16]The API version recommended here might change with the next guideline revision.
[17]For example, to obtain its URL, a servlet could use the field `ceres.HttpServletInfo.baseUrl`.

5.12 Interoperability

- Every developer MUST document which data the developed software consumes and outputs, and which protocols are used for this.
- Every software module exchanging Java objects with another software module MUST provide a Java interface declaration. Every software module exchanging any other kind of data with another software module MUST exchange this data via XMI documents.[18]
- Every developer MUST document wether the software invents a proprietary interface or protocol for data exchange. If such a thing is invented, the developer MUST document it.

[18] HERA may define an additional interface. If the development of HERA continues, this point must be adapted.

6 Development Course

LI20040615	In the meeting on 2004-06-15, the CERES project group specified several guidelines for the software development in CERES. The list is still incomplete and will be finished in the meeting on 2004-06-22.
LI20040625	Completed the guidelines according to the results of the discussion in the meeting on 2004-06-22.
LI040715	Revised the guidelines according to the results of the discussion in the meeting on 2004-07-06.
LI20040731	In the meeting on 2004-07-27, the CERES project group accepted the guidelines. The guidelines affect every CERES software project that is started after 2004-07-27, including projects that are already in the planning phase but not yet in the coding phase.
LI20041027	Replaced all rules that refer to browser compatibility or usability by references to the Web Application Style guide.
LI20050201	Overworked the guidelines according to the first periodic revision of the guidelines on 2005-02-01.
LI20050419	Added a portability criterion concerning absolute paths.
NB20050605	Added a maintainability criterion concerning the separation of files according to their function.
NB20050810	Adapted the guidelines according to the periodic revision of the guidelines on 2005-08-09.
NB20060215	Adapted the guidelines according to the periodic revision of the guidelines on 2006-02-14.
NB20060227	Adapted the guidelines according to the discussion on 2006-02-21.
NB20060819	Adapted the guidelines according to the periodic revision of the guidelines on 2006-08-08.

References

[GHJ95] Erich Gamma, Richard Helm, and Ralph Johnson. *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison-Wesley, Reading (Massachusetts), 1995. 9

[WKI+3b] C Weßel, G Karakas, L Ißler, F Weymann, S Palm, C Spreckelsen, and K Spitzer. Ceres - ein instrument zur webbasierten darstellung, pflege und visualisierung von krankenhausinformationen. In W Köpcke, editor, *Informatik, Biometrie und Epidemiologie in Medizin und Biologie. Abstracts der 48. Jahrestagung der GMDS*, pages 361–362, Jena, 14.–19. Sept. 2003b. Urban und Fischer Verlag. Project CERES. 3

[WWS06] Christa Weßel, Frédéric Weymann, and Cord Spreckelsen. A framework for the web-based multi-method evaluation of a web-based information system on hospitals. In M Löffler and A Winter, editors, *51. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS). Klinische Forschung vernetzen. 10.-14. September 2006*, page 90, Leipzig, 2006. Universität Leipzig. Abstract online: http://www.egms.de/en/meetings/gmds2006/06gmds226.shtml. 3