Implementing Software Development Guidelines in a Medical Informatics Research Project

L. Ißler[1], C. Spreckelsen[2], C. Weßel[2]
[1]Institute for Medical Informatics, Statistics and Epidemiology (IMISE), University of Leipzig, Leipzig, Germany
[2]Department of Medical Informatics, RWTH Aachen University, Aachen, Germany

Summary

Objectives: Due to the non-commercial, research-oriented context, software in medical informatics research projects is often developed by researchers as a proof-of-concept without applying structured software development process models. A guideline for software development can bring sufficient structure to the development process while avoiding the complexity of industry-standard methods.

Methods: We adapted the common evidence-based guideline development process from medicine to build a guideline for software development in our medical informatics teaching and research project.

Results: Our guideline development used the six steps of problem identification, first proposal, review, revision, gaining consensus and periodic guideline review. Since the developers had taken part in guideline development, our guideline clearly states the consensus of the development team over critical topics. The guideline improved the quality of our source code in structure and understandability.

Conclusions: A software development guideline that is developed following a consensus panel approach is a good instrument for basic software quality assurance in domains where complex, industry-standard software development methods cannot be applied. This is especially the case in non-commercial, research-oriented medical informatics projects where mainly non-software engineers like students do the development work.

Keywords

Software engineering, guidelines, medical informatics

Methods Inf Med 2007; 46: 641–645

doi:10.3414/ME0428

1. Introduction

The main focus of the CERES project at the Department for Medical Informatics at the RWTH Aachen University is the development of a web-based information portal on hospitals [1]. The core of the CERES project is an object-oriented database containing structural data about hospitals like the number of departments and the special areas of surgery. The information is presented through different front-ends over the world wide web, including a textual and a graphical presentation of data about hospitals. The team behind CERES is multidisciplinary. Members are both students and researchers from computer science, medicine, economics and public health. CERES as a teaching project implements project-based learning in computer science, enriched by a multidisciplinary and continued setting [2].

In the year 2005, after two years of project work the CERES project resulted in several student research projects and diploma theses on the one hand and a set of software modules on the other. Almost every software module was developed in the context of a thesis preparation. The scientific part of these theses was done according to the common scientific practice, but the software was developed according to the personal skill level of the respective student. This lead to source code maintenance problems, documentation at the wrong place, missing, or bad, and unstable software resulting from erroneous code.

From these experiences the need for a clearly structured and understandable development procedure arose. We wanted this procedure to cover implementation, integration, testing and evaluation of the developed software; but foremost, it should be helpful, understandable and easy to use [3]. By this it should pay special attention to the work practices in the project [4], namely to the fact that the development team consists mainly of students which stay relatively short in the project. Common industrial-standard process models for software development like the Rational Unified Process (RUP) [5] are too heavyweight for application under these circumstances.

Instead, we claim that a software development guideline is an ideal quality management tool for use in a medical informatics teaching and research project. Such tools are needed especially in medical informatics [6]. Several questions have to be answered:

1) Which steps lead to the implementation of a guideline for software development?
2) What could be the basis for such a guideline?
3) What are the building blocks for such a guideline?
4) Is the guideline approach useful for improving the software quality?

In this paper, we answer these questions by presenting how to apply our guideline-based approach for software development in research projects to the domain of medical informatics.

2. Guidelines in Medical Informatics

Guidelines are systematically developed statements to inform their user about a process in specific circumstances [8]. In medicine, guideline development follows a clear and comprehensible process and is scientifically founded. A group of experts assesses evidence, defines the guidelines in a cyclic process, and eventually gains consen-

Received: May 16, 2006; accepted: October 5, 2006

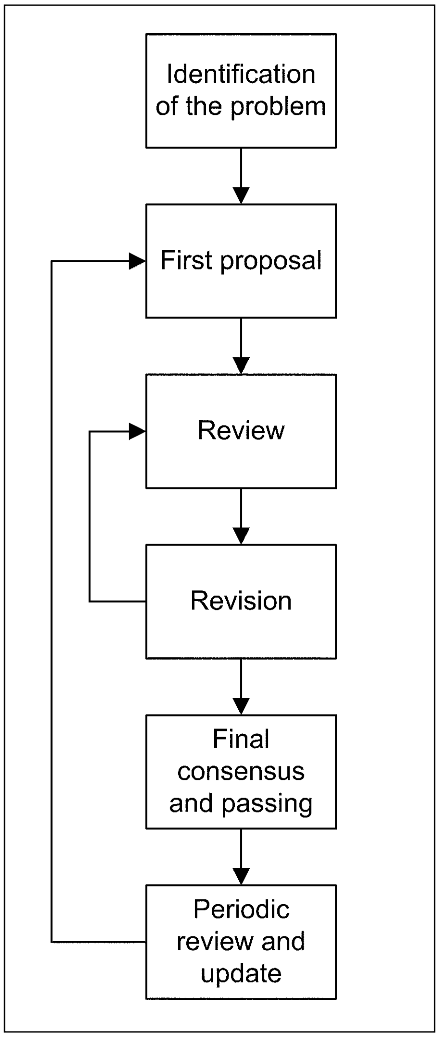
Identification of the problem
First proposal
Review
Revision
Final consensus and passing
Periodic review and update

Fig. 1 The six steps to develop a software development guideline

sus [9]. The guidelines are then implemented in a specific setting by adapting them to the setting's constraints [10, 11]. Most crucial, guidelines must be reviewed and updated after a defined period, to keep the consensus alive and the underlying evidence up-to-date [10, 12].

In software engineering and medical informatics there are recommendations for, for instance, source code quality [13], source code layout [14] and other special aspects of the developed software [15]. It is also common to adapt a recommendation to a specific setting. Using a process model for software development like the Rational Unified Process (RUP) [5] or the V-Model [16] as a template is an example, and the adaptation of general reference models to a specific domain [17] is another.

However, this adaptation of standards is usually not accompanied by neither a systematic process of gaining consensus nor periodic reviews of this consensus. But the consensus of a project team about the applicated rules is crucial, especially in domains with high fluctuation of developers and therefore the permanent need to explain and defend established policies against new team members.

3. A Process Model for Implementation and Maintenance of Software Development Guidelines

Our proposed process for implementing a guideline consists of moderated discussions and the work of a distinct author who is responsible for the guidelines. For each topic or criterion mentioned in a chosen basic standard, goals have to be identified. From the goals, rules which state how to achieve these goals have to be derived.

The six steps of the process are as follows (refer to Fig. 1 for a graphical representation):

1) Identification of the problem, the needs and possible solutions and assignment of the responsible author. The research team decides to implement software development guidelines during a moderated focus group session.
2) First proposal of the guidelines. The responsible author writes a first draft after literature research on existing standards and guidelines, and derives goals and rules which can serve as a basis and pattern.
3) Review. Every team member edits the proposal and the team identifies the necessary adaptation in a consensus panel session. The consensus based upon professional experience is crucial for the progress and success of the implementation process.
4) Revision. The author adapts the guidelines according to the comments of the team members, which could involve revision, dropping and invention of both goals and rules.
5) Final consensus and passing of the guidelines. Iteration of steps 3 and 4 until a final consensus is gained.
6) Periodic review and update. The guidelines must be reviewed after certain periods to assure that they are still clear and applicable to the project. We chose six months as a review period, with an option to review the guidelines on demand if the team identifies relevant topics. This corresponds to a formative evaluation of the feasibility of the guidelines.

We recommend step 1 to be performed as focus group to identify problems, needs and possible solutions [18, 19]. For the review and gaining consensus on the guidelines in steps 3 and 5 we recommend the approach of a consensus panel. Both methods were developed in social sciences as qualitative research methods and have served well for guideline development [21].

4. The CERES Software Quality Guidelines

The guideline development in CERES took three months. The team uses the guidelines since August, 2004 and performed three regular reviews after six, 12 and 18 months of use, respectively. Following the IEEE 610.12-1990 standard [18] as a basis, we developed guidelines with the main target of software quality. These software quality guidelines cover the topics from this standard, namely *correctness*, *reliability*, *efficiency*, *integrity*, *usability*, *maintainability*, *flexibility*, *portability*, *testability*, *reusability* and *interoperability* of software [18]. Part 1 of the ISO/IEC 9126 standard defines another set of similar and also useful criteria [7].

To the original topics we added the two topics *documentedness* and *copyright and licenses* during the consensus process (see Section 3) to fit our individual needs. An overview over the guidelines is given in Table 1. In this section we describe as examples three rules of the guidelines[a].

[a] For space reasons, we cannot discuss all guideline rules in detail. The full text of the guidelines can be obtained from [26].

4.1 Example: Correctness

The IEEE 610.12-1990 standard defines a software as correct if and only if it fulfills a written specification [18]. From this criterion we derived the goal to have a written specification for a software module to be able to test it. The following rule obliges the developers to write a requirements specification with standardized contents:

- The author MUST create a requirements specification document for the software. The requirements specification document ("specification" for short) MUST list the 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.

If a developer follows this rule, every piece of software has a corresponding requirements specification. The software can be tested against the specification and then declared as correct or not.

This rule is a typical example for a guideline rule[b]. It gives a clear instruction to the developer, and it specifies what to do in an exceptional case (indicated by the MAY keyword). Nevertheless, the presented rule leaves out methodical aspects of requirements engineering, which should be added especially for the development of clinical systems [23].

4.2 Example: Documentedness

New members of the CERES project team often had problems getting familiar with the existing bunch of software source code in the project. With well-written, sound, up-to-date, understandable, and easily accessible documentation, getting familiar is easier.

[b] There are two types of rules: mandatory ones and truly optional ones. We indicate the type of the rule by the keywords MUST and SHOULD (and sometimes MAY) (in capitals), which are defined in the internet standards document RFC 2119 [22].

Table 1 Overview over the CERES software quality guidelines. For each objective/goal, there is a rule in the guidelines. The goals for rules presented in the article are emphasized by italics.

Topic	Objectives/Goals
Correctness (4 rules)	*requirements specification*; discussion of the requirements with the team; rational for specification changes; software acceptance test
Reliability and efficiency (3 rules)	defined list of non-functional requirements; provision of a mechanism to measure the non-functional requirements; transfer only needed data
Integrity (2 rules)	one main class for all servlets; using the project-internal database API for database access
Usability (1 rule)	application of the CERES web style guide
Copyright and Licenses (4 rules)	documentation which third-party software with licenses is used; use only free or licensed software
Documentedness (6 rules)	*Javadoc for identifiers*; method explanations in Javadoc; *short tutorial for every module*; list of required third-party libraries for every module; list of resources in access for every module
Maintainability (8 rules)	separation of program logic and input/output; test cases for every module; package naming convention; separation of class files/data files; application of the Sun's Java Coding Conventions; modular programming; use of Java's mechanisms for maintainability and reusability
Flexibility (1 rule)	documentation of the software's intended purpose
Portability (5 rules)	Java version to use; servlet API version to use; browser requirements; avoidance of absolute pathnames; avoidance of hard-coded URLs
Testability (2 rules)	avoidance of a 'do everything' method
Interoperability (3 rules)	documentation of the consumed and produced data and the used protocols; specification of interfaces and usage of XML formats for data exchange; documentation of newly invented protocols

We call this feature 'documentedness', and we think it is a good idea to achieve documentedness in every project.

The easiest way to achieve documentedness in Java (the programming language used in CERES) is to use Java's built-in tool 'Javadoc'. The following is how we formulated this as a rule:

- 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.

But Javadoc has to be accompanied with some documentation outside the source code, as it turned out during a periodic guideline revision. This is stated by the following rule:

- 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.

5. Experiences from Working with the Guidelines

The CERES project started in April, 2002. We introduced the guidelines to the project in August, 2004. Before the guideline introduction, software was developed in seven student research projects. With the guidelines in effect, software was developed in four diploma theses and two student research projects. The students stayed in the project from six to about twelve months.

Until April, 2006, the guidelines were revised three times according to the defined interval of six months. Our original guidelines consisted of 33 rules. In the three periodic revisions 16 rules were revised (which usually meant that they were re-formulated to be more precise), eight new rules were added, and three rules were removed. During this period, the main author of the guidelines (which is the first author of this article) left the project, and another team member took over the responsibility for the guidelines without problems.

As an overall result we observed that the way the student developers deal with problems occurring during the software development changed. Instead of asking on every problem, they checked problems against the guidelines first and brought them to discussion only if the guidelines could not unambiguously solve them. In several cases, developers new to the project suggested new rules during the next periodic revision to clarify the ambiguities they had initially discovered.

It is worth noting that the simple existence of the guidelines significantly reduced the amount of lengthy discussions between the developers about which architectural paradigm to use, whether to reengineer old code, and where to place the software documentation. These topics were now discussed once and then fixed in the guidelines.

An interesting effect of the guidelines was that whenever a developer encountered a piece of old code that violates a guideline rule he voluntarily tried to fix this. The developers were highly motivated not to combine old code of poor quality with the new, high-quality code.

6. Discussion

Summarizing our results of the development and implementation of a software development guideline (which focuses mainly on aspects of software quality), we can answer the questions from Section 1 as follows:

1) Guidelines for software development should reflect the consensus of the development team, and keep this consensus alive. We developed a process model for implementing and reviewing guidelines.
2) Standards from software development should be used as a basis for guideline development.
3) Following our proposed process model leads to a software development guideline that is helpful, understandable and easy to use, and gives advice on how to keep the guideline up to date.
4) From our observations with the guidelines in effect we found that a software development guideline improves the quality of the source code in structure and understandability.

Compared to mature, industry-standard software-development process models like the Rational Unified Process (RUP) [5] and the V-Model [16], a guideline is lightweight and easy to understand but nonetheless effective. This is a great advantage in environments with high fluctuation (like a teaching project) of developers from multiple disciplines (like often in medical informatics).

Our six steps to develop and implement a software development guideline together with their periodic revision ensure that the guidelines reflect the consensus of all developers. This (admittedly) considerable discussion effort leads to a high acceptance of the guidelines [10].

Gaining consensus and formulating it explicitely is also helpful when the development is completed. Software development guidelines state a number of assumptions about the development and implementation process and the software's future use. An explicit formulation of these assumptions can be helpful when applying software to a specific work environment [24] and when moving the software from one context to another [25].

7. Conclusion and Outlook

We regard a guideline for software development as an appropriate and recommendable instrument to ensure high software quality both encountering scientific standards and respecting the special environment of a teaching and research project. We will proceed in the continuous maintenance of the guidelines and evaluate formatively whether the consensus panel approach remains feasible.

We expect that our guideline approach is also applicable to other domains in medical informatics, one of which is modeling. Especially as a quality assurance instrument, it should be helpful to gain consensus about the quality criteria that apply to a developed model, to derive a modeling guideline from these criteria, and to carefully review both criteria and guidelines in defined intervals.

In future, guideline development could be supported by a set of 'rule templates' or 'general recommendations' derived from existing standards and guidelines as a consensus for specific areas in medical informatics. Such general recommendations could help to achieve comparable quality of development work in these areas. Our current guideline could serve as a starting point for this.

Acknowledgments
The authors thank the CERES project members. Their cooperation was and is crucial for the development, maintenance and use of the guidelines. The authors also thank Prof. Dr. Alfred Winter, Leipzig, and the anonymous reviewers for their helpful comments on this paper.
The work presented in this article was done at the Institute for Medical Informatics at the RWTH Aachen University. During this time, the main author moved to the Institute for Medical Informatics, Statistics and Epidemiology of the University of Leipzig.

References

1. Weßel C, Spreckelsen C, Ißler L, Karakas G, Möller W, Palm S, et al. Die Qualitätsberichte der deutschen Krankenhäuser im Internet ab 2005: Erstellung mit Hilfe des objektorientierten Metamodells für Krankenhäuser MINERVA. In: Ammenwerth E, Gaus W, Haux R, Lovis C, Pfeiffer KP, Tilg B, et al., editors. Abstracts der 49. Jahrestagung der GMDS, 26.-30. September 2004, Innsbruck. Niebüll: videel; 2004. pp 144-146. (In German.)
2. Pape B, Bleek WG, Jackewitz I, Janneck M. Requirements for Project-Based Learning – CommSy as an Exemplary Approach. In: Sprague RH, editor. Proceedings of the 35th Annual Hawaii International Conference on System Sciences. Los Alamitos; 2002.
3. Kuhn KA, Lenz R, Elstner T, Siegele H, Moll R. Experiences with a generator tool for building clinical application modules. Methods Inf Med 2003; 42 (1): 37-44.
4. Nykänen P, Karimaa E. Success and Failure Factors in the Regional Health Information System Design Process – Results from a Constructive Evaluation Study. Methods Inf Med 2006; 45: 85-89.
5. Jacobson I, Booch G, Rumbaugh J. The Unified Software Development Process. Addison-Wesley; 1999.
6. Ammenwerth E, Shaw NT. Bad Health Informatics Can Kill – Is Evaluation the Answer? Methods Inf Med 2005; 44: 1-3.
7. International Organization for Standardization. ISO/IEC 9126 Software engineering – Product quality – Part 1: Quality model; 2001.

8. Field MJ, Lohr KN, editors. Clinical practice guidelines: directions for a new program. Washington: National Academy Press; 1990.
9. Shekelle PG, Woolf SH, Eccles M, Grimshaw J. Clinical guidelines: Developing guidelines. British Medical Journal. 1999;318:593–6.
10. Feder G, Eccles M, Grol R, Griffiths C, Grimshaw J. Clinical guidelines: Using clinical guidelines. British Medical Journal 1999; 318: 728-730.
11. Michie S, Johnston M. Changing clinical behaviour by making guidelines specific. British Medical Journal 2004; 328: 343-345.
12. Council of Europe. Developing a methodology for drawing up guidelines on best medical practices (Recommendation (2001) and explanatory memorandum). Council of Europe; 2002.
13. Greif N, Schrepf H. Information Technology Guidelines for Software Development : Guideline for Programming in C. Physikalisch-Technische Bundesanstalt. Bonn; 2002.
14. Sun Microsystems. Code Conventions for the Java Programming Language : Revised April 20, 1999. Sun Microsystems; 1999.
15. Chisholm W, Vanderheiden G, Jacobs I, editors. Web Content Accessibility Guidelines 1.0. World Wide Web Consortium; 1999.
16. Federal Republic of Germany. V-Model 97, Lifecycle Process Model – Developing Standard for IT Systems of the Federal Republic of Germany: General Directive No. 250. Federal Republic of Germany; 1997.
17. Averill E. Reference models and standards. StandardView 1994; 2 (2): 96-109.
18. Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology. ANSI/IEEE Standard 610.12-1990; 1990.
19. McAlearney AS, Schweikhart SB, Medow MA. Doctors' experience with handheld computers in clinical practice: qualitative study. British Medical Journal 2004; 328 (7449): 1162.
20. Smith AC. Design and Conduct of Subjectivist Studies. In: Friedman C, Wyatt J, editors. Evaluation Methods in Medical Informatics. Springer; 1997. pp 223-253.
21. Coreil J. Group interview methods in community health research. Med Anthropol 1995; 16: 193-210.
22. Bradner S. Request for Comments: 2119. Key words for use in RFCs to Indicate Requirement Levels. Network Working Group; 1997.
23. Reddy M, Pratt W, Dourish P, Shabot MM. Sociotechnical Requirements Analysis for Clinical Systems. Methods Inf Med 2003; 42: 437-444.
24. Hartswood MJ, Procter RN, Rouchy P, Rouncefield M, Slack R, Voss A. Working IT Out in Medical Practice: IT Systems Design and Development as Co-Realisation. Methods Inf Med 2003; 42: 392-397.
25. Kaplan B, Shaw NT. Future Directions in Evaluation Research: People, Organizational, and Social Issues. Methods Inf Med 2004; 43: 215-231.
26. Ißler L, Becker N, Spreckelsen C, Weßel C. The CERES Project – CERES Software Development Guidelines. Aachener Schriften zur Medizinischen Informatik 2007; 1. Obtainable via http://publikationen.med-informatik.ukaachen.de/.

Correspondence to:
Lutz Ißler
University of Leipzig
Institute for Medical Informatics, Statistics and Epidemiology (IMISE)
Haertelstr. 16-18
04107 Leipzig
Germany
E-mail: lutz.issler@imise.uni-leipzig.de