# Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Systems

Jörg Rech

**A part of software quality assurance is concerned with the diagnosis of defects, which decrease quality aspects of software systems, such as maintainability, reusability, portability, or performance. Concurrent approaches for defect diagnosis focus on post-development manual inspections of the software's source code. The context-sensitive diagnosis of quality defects is a new approach that is based on a quality model and integrates information on the context of a diagnosed element during software development (in-situ). This thesis proposes an ontology for quality defect representation, a method for quality-oriented, in-situ, and context-sensitive quality defect handling, and a reference architecture for the development of extensible in-situ quality defect diagnosis tools.**

## 1 Introduction

Today, software systems are getting larger and more complex. More systems are integrated into larger systems, more technologies are used to build interoperable systems, and more features are requested by the customers.

To improve and control the quality of their products, software organizations often apply quality improvement activities like refactoring between development iterations. In the refactoring phase systems are reworked at problematic locations to improve the system's quality. But the effort for the manual detection of where to apply individual refactorings result in either short or costly refactoring phases – if it is done at all.

This thesis [5] addresses the problem of diagnosing and handling quality defects (i.e., opportunities for refactoring such as antipatterns, smells, flaws, pitfalls, anomalies, illnesses, etc.) in object-oriented source code following the guiding picture of medicine. It focuses on the techniques for the context-sensitive diagnosis of quality defects and their iterative, semi-automatic, and quality-oriented handling during development. The main contribution of this research is a comprehensive method for the context-sensitive diagnosis and continuous handling of quality defects during development, which consists of three main elements. First, a formal model in form of an *ontology (SQuaD),* that helps to understand the fundamental concepts behind quality defects and the information available for their diagnosis. It describes their structure, symptoms, affected qualities, and associated refactorings as well as their interrelations and dependencies. Second, a *context-sensitive quality defect handling method (SQuaH)* for quality-oriented quality defect diagnosis and handling that inherently supports the recurring handling and refactoring of quality defects. Third, an *extensible reference architecture (SQuaRE)* for in-situ quality defect diagnosis and handling systems.

All these contributions have been successfully applied and their effectivity and efficiency is supported with empirical evidence from several studies as well as experiments. The empirical data indicates that the approach is *accepted and useful* to software engineers, has a positive *effect on the perceived quality* of a software system, and increases *the performance* during quality defect diagnosis and handling.

## 2 Research Challenges

Current approaches to QDD include rule-based detection strategies [8], statistics-based diagnosis [4], and visualization-based diagnosis [9]. However, support for context-sensitive quality defect diagnosis is still missing. More specifically, the following challenges are targeted in this thesis:

**Missing overview of quality defects:** There is no comprehensive, structured, and systematic overview of quality defects and related concepts such as causes, treatments, effects, or symptoms.

**Different quality foci of domains**: It is unclear how application domains prioritize the different quality aspects, and quality defects are not associated with the quality aspects they are decreasing.

**Information flood in QDD**: Quality defect diagnosis tools find many quality defects in a software system but flood the user with too much unnecessary information.

**Contextual quality defects**: Several quality defects are not applicable in particular locations. For example, the antipattern "God Class" is not necessarily applicable in a façade class (ref. the design pattern "Façade").

**Refactoring uncertainty in QDD**: The optimal process for treating (e.g., refactoring) quality defects is unknown if multiple quality defects exist at one location.

**No continuous QDD process**: Current QDD approaches are not continuous and comprehensive enough to handle the recurring diagnosis of quality defects.

**No support for QDD during development**: Existing quality defect diagnosis techniques are not designed to support the developer during programming activities (i.e., in-situ) and in recurring applications of treatments (e.g., refactoring).

## 3 Defect Representation – SQuaD

Researchers and practitioners have developed partial and informal classifications and templates for quality defects (and especially for code smells). However, these cover only one type of defect and do not integrate other concepts such as cause, effects, or treatments.

In order to represent quality defects, the ontology SQuaD was developed using OWL. SQuaD is used to characterize software quality defects and their interrelationships. It includes definitions, descriptions, and explanations of the template and relationships used for quality defects. This ontology is used for the definition of the quality defect diagnosis process SQuaH (section 4) and the reference architecture SQuaRE (section 5). SQuaD consists of three models:

A **quality defect model** that describes and characterizes quality defects, the concepts associated with quality defects, and relationships between them using OWL-DL.

An **information model** used to describe information available in object-oriented software systems that can be used in the diagnosis of quality defects.

A **context model** that describes contextual factors in object-oriented software systems and that can be used in the diagnosis as well as the prioritization of quality defects.

The quality defect model, as presented in Figure 1, models the interrelation between defect-related concepts, structures this knowledge for humans, and formalizes it for machine usage such as prioritization of quality defects.

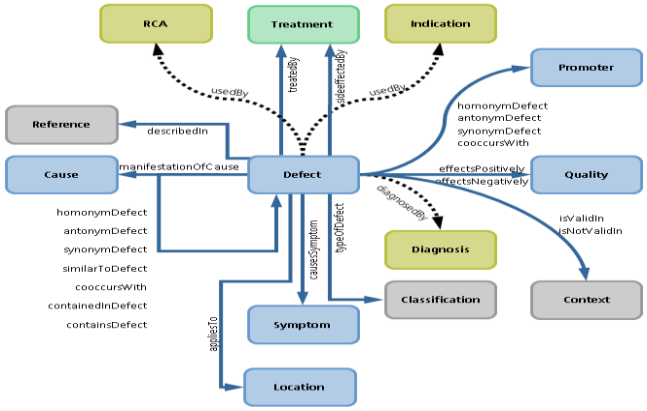


Figure 1 SquaD ontology – conceptual view of the defect model

For a quality *defect* such as “Large Class” we identify several *symptoms* such as “Too many methods” or “Too many Lines Of Code”. These symptoms are based on *characteristics* of a *location* (e.g., a class) such as “lines of code”. Based on these symptoms (and optionally, characteristics) a *diagnosis* rule, such as “(Too many methods AND attributes) OR Too many LOC”, is used to identify defects. A *defect* has a negative effect on a quality aspect (e.g., maintainability) while quality *promoters* such as design patterns have a positive effect. Furthermore, several *treatments* such as “Extract Class” are associated with a defect. *Causes* of the “Large Class” defect are, e.g., that classes in the software system represent (large) tables in a database. To prevent this cause a *prevention* technique such as “Extract Table” might be used. However, some properties of a software system represent fundamental *predispositions* that cannot be changed easily and need comprehensive and often costly *reengineering* activities.

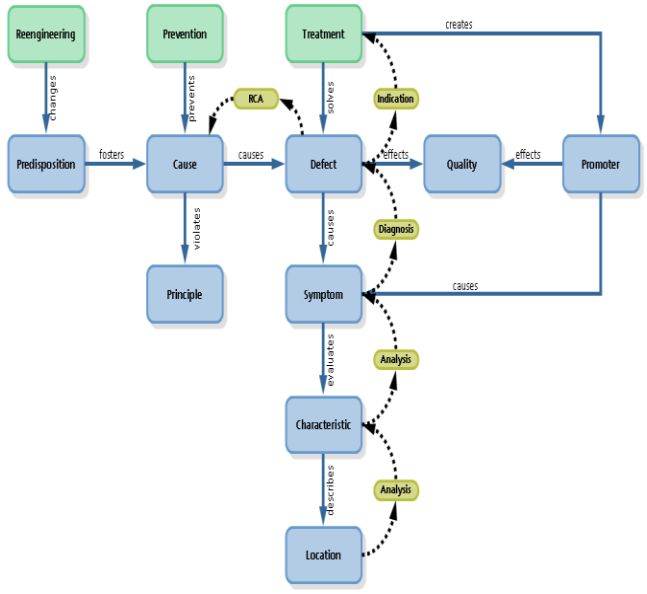


Figure 2 The concept Defect with outgoing relationships

While the conceptual view in Figure 1 describe the main concepts and relationships of the ontology SQuaD, further relationships in the detailed Ontology are required for the handling of quality defects. For the concept defect (see Figure 2), relations such as *synonym defect* describe almost identical defects and is used to reduce the amount of diagnosis activities, *antonym defect* describe opposing defects that do not have to be checked if the other is found, and *treated by* is used to identify the most promising treatment to remove defects (especially, when multiple quality defects are found at one location).

## 4 Diagnosis Process – SQuaH

One current research trend is to increase automation of the mentioned processes in order to support the developers with automated refactoring or defect diagnosis systems. However, current methods for diagnosis are focused on identifying defects in software systems without considering the context of the element or the recurrent application of the process.

In order to systematize the diagnosis and handling of quality defects, the method SQuaH was developed. It addresses the problem of the quality-oriented and context-sensitive diagnosis, treatment, prioritization, and handling of quality defects. SQuaH is based upon SQuaD (section 3) and is used as a basis for the specification of the quality defect diagnosis reference architecture SQuaRE (section 5).

SQuaH consists of eight sub-processes, which focus on different aspects of the handling method. While it is based upon the ADAIR process [8], which consists of the phases Activate, Detect, Advise, Improve, and Record, it extends it in many different aspects. In the process execution, the following sub-processes are performed:

**Analyze System**: The analysis of the system (and other information sources such as a versioning system) triggered by a time or change event. This analysis results in the identification of symptoms that are used in the diagnosis.

**Diagnose Defects**: Manual or automatic quality defect diagnosis techniques are used to diagnose potential quality defects based on the symptoms and the context information.

**Prioritize Defects**: The diagnosed quality defects are prioritized based upon the predefined quality model, the defect context, and other preferences of the user, project, etc.

**Indicate Treatment**: Based on the diagnosed and prioritized quality defects, the quality model, and the defect contexts, a plan for the treatment of the system is constructed.

**Examine Defects**: The engineer processes the potential quality defects based on their priority, analyzes the affected system, and decides if the quality defect is truly problematic and if the system can be modified without side-effects.

**Treat Defect**: If the quality defect is to be removed from the system, the engineer is briefed about the existing quality defects and their rationales as well as about available treatments and their impact on the software’s quality.

**Describe Decision**: If a potential quality defect is unavoidable or its treatment would have a negative impact on an important quality aspect, this decision is formulated and recorded in order to prevent future presentation of this defect.

**Document Change**: The software system is annotated regarding the change (or the rationale why it was not changed) using specific tags.

## 5 Software Architecture – SQuaRE

Current studies have shown a very high interest in intelligent assistance systems, and in the future, we will see more such

tools with a higher coverage of defects (or other assistance topics). Contemporary tools for quality defect diagnosis are used post-development, post-iteration, or during the build-phase, but the event system of the IDE also allows direct and immediate diagnosis during programming.

In order to support the development of quality defect diagnosis tools with in-situ and context sensitive diagnosis techniques, the reference architecture SQuaRE was developed. It describes, defines, and explains relevant interfaces and plugin mechanisms. SQuaRE is based upon SQuaD (section 3) and SQuaH (section 4) and was implemented in a programming IDE focused on the programming language Java (DoctorQ) as well as a modeling IME focused on the modeling language UML (VIDE-DD).

## 6 Evaluation

Previous studies about design guidelines [2] or pattern-based refinement [3] have shown a positive effect on the maintainability of object-oriented designs. However, studies regarding approaches that inform the software engineer during development are rare. The explorative and evaluative studies conducted in this thesis evaluate the situation, motivate the research, and systematically construct the contributions in this thesis. They show that a quality-oriented, context-sensitive, and in-situ quality defect diagnosis approach is required and that the contributions represent a practical and feasible approach to this challenge.

Furthermore, a controlled experiment was conducted to evaluate three main hypotheses. These hypotheses are concerned with the questions if the context-sensitive in-situ diagnosis of quality defect (SQuaH-based technology) is accepted by, and are useful to, software engineers ($H_1^A$), have a positive effect on the software quality ($H_1^Q$), and have an effect on the software engineer's performance ($H_1^T$).

The performance of the subjects was analyzed with regard to the hypotheses. This was done by comparing the test results of the subjects who used the tool DoctorQ with SQuaH features against a control group that used the tool DoctorQ without SQuaH features. In summary, the following effects were extracted from the hypothesis evaluations:

**SQuaH-based technology is accepted among software engineers ($H_1^A$)**: The technology is perceived to help at programming, to have a low effort to use, to be a good idea, subjects have the required environment, it be usable without help, and is not intimidating. However, no significant difference regarding social influence was found.

**SQuaH-based technology improves the quality of the software system ($H_1^Q$)**: The subjects perceived an improvement of Maintainability of 25.3 % (comparison of means; using a Maintainability-oriented quality model).

**SQuaH-based technology increases the performance of developers ($H_1^T$)**: The subjects investigate and treat 33% more quality defects than with standard QDD technology.

## Conclusion

The thesis provides a systematic literature review about the software defect diagnosis field and describes the context-sensitive approach for quality defect diagnosis. It contributes the *ontology SQuaD,* the *handling method SQuaH, and the reference architecture SQuaRE* to the research field.

While the ontology currently contains many quality defects, it is still not complete. In the future, we plan to build a more comprehensive version of the (instance) ontology. Furthermore, the integration of Experience Management via participatory discourse systems [6] or Wiki-based documentation platforms [7] might be used to support the reuse of knowledge and exchange between engineers. Similarly, results from AI research such as software agents might be applied in order to intelligently assist the software engineer as a knowledge worker [1].

## Referenzen


[1] Althoff, K.-D., Hanft, A., Mänz, J., Newo, R., Schaaf, M., Decker, B., Nick, M., and Rech, J., "Intelligent information systems for knowledge work(ers)," presented at the 6th Industrial Conference on Data Mining (ICDM'06), 2006.

[2] Briand, L. C., Bunse, C., and Daly, J. W., "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs," IEEE Transactions on Software Engineering, vol. 27, no. 6, pp. 513-30, 2001.

[3] Bunse, C., Pattern-based Refinement and Translation of Object-Oriented Models to Code, PhD Thesis. Kaiserslautern: University of Kaiserslautern, Department of Computer Science, 2000.

[4] Kreimer, J., Adaptive Erkennung von Software-Entwurfsmängeln, PhD Thesis. Universität Paderborn, Fakultät Elektrotechnik, Mathematik und Informatik, 2005.

[5] Rech, J., Context-sensitive Diagnosis of Quality Defects in Object-Oriented Software Systems, Ph. D. Thesis. Hildesheim: University of Hildesheim, Department IV, 2009.

[6] Rech, J., Decker, B., Althoff, K.-D., Voss, A., Klotz, A., and Leopold, E., "Distributed Participative Knowledge Management: The indiGo System," in The Global Enterprise: Entrepreneurship and Value Creation. Binghamton (New York): Haworth Press, 2007.

[7] Rech, J., Ras, E., and Decker, B., "RIKI: A System for Knowledge Transfer and Reuse in Software Engineering Projects," in Open Source for Knowledge and Learning Management, M. D. Lytras and A. Naeve, Eds.: IDEA Group Publishing, 2007.

[8] Robbins, J. E., Cognitive Support Features for Software Development Tools, Ph.D. Thesis. Irvine: University of California1999.

[9] Simon, F., Meßwertbasierte Qualitätssicherung: Ein generisches Distanzmaß zur Erweiterung bisheriger Softwareproduktmaße (in German), Ph.D. Thesis. Brandenburgische TU Cottbus, Fakultät für Mathematik, Naturwissenschaften und Informatik, 2001.


## Kontakt


**Dr. Jörg Rech**
Semantic Technologies
Kurt-Schumacher Str. 72
67663 Kaiserslautern, Germany
Joerg.Rech@gmail.com


Jörg Rech is an entrepreneur and founder of the company Semantic Technologies, which is working on systems in the area of Web 2.0, Web 3.0 and the Semantic Web. After studying Computer Science at the University of Kaiserslautern, he was a scientist and project manager at the Fraunhofer IESE. His research concerns defect diagnosis, software analysis, semantic technologies, and knowledge management.