Research Ideas and Outcomes 1: e7576
doi: 10.3897/rio.1.e7576

OPEN ACCESS

DFG Grant Proposal

Continuous and Focused Developer Feedback on Software Quality (CoFoDeF)

Stefan Wagner ‡

‡ University of Stuttgart, Stuttgart, Germany

Corresponding author: Stefan Wagner (stefan.wagner@informatik.uni-stuttgart.de)

Reviewable v1

Received: 21 Dec 2015 | Published: 21 Dec 2015

Citation: Wagner S (2015) Continuous and Focused Developer Feedback on Software Quality (CoFoDeF). Research Ideas and Outcomes 1: e7576. doi: 10.3897/rio.1.e7576

Executive summary

Software is intangible and, therefore, difficult to understand and control. Quality problems often creep in unnoticed because the developers do not have a full view of the history and all the consequences of their changes to the system. Therefore, we observe quality decay in many software systems over time. A countermeasure would be frequent quality analyses starting early in the development, e.g. in nightly builds. Frequent quality analyses reduce the problem of quality decay because quality defects are detected soon after they were introduced, but they still allow defects to be introduced. This means that builds might fail, other developers might have already worked with that code revision and changes will result in rework.

I propose to apply the detection of quality problems directly before or while they are created. For this, we need continuous and focused feedback about a software's quality to the developers. To give that feedback to the developers without overwhelming them, a main goal of this project is to understand the quality information needs of developers. We will build an information needs theory by empirical analysis and use this theory to investigate diverse quality checks integrated directly into development environments. We will build on advances in test-case generation, static analysis and repository mining to provide this feedback. Finally, we will extensively investigate the impact of the created continuous and focused feedback on the developers in controlled experiments and industrial case studies. We see the main challenges in providing the right feedback to the developer at the right time and in the right context as well as supplying the developers with

the necessary explanations to understand and make use of the shown information. We will address these challenges by building focused and learning analyses which understand the context of the change and the developer. The analyses will learn manually and automatically the developer's preferences. Furthermore, we mine and include rationales from detailed quality models, web resources and slicing into the developer feedback. If we succeed, the results will increase the usage of quality analyses in practice, make the developers aware of quality problems, and reduce quality problems and the expenditure they cause. Furthermore, our project will open up possibilities for research to create and integrate new kinds of immediate feedback to the developers.

Keywords

Software Quality, Information Needs, Fast Feedback

Third parties involved in the project

As case study partners:

- AEB GmbH, Stuttgart
- CQSE GmbH, Garching b. München

State of the art and preliminary work

Continuous quality control and just-in time quality assurance

Quality control is a process which emerged from the manufacturing area where it means inspecting products during and after production to detect problems and ensure a high level of quality. Feigenbaum (2002) extended it to *total quality management*, which includes the idea of continuously improving the quality of processes and products. This broader view has been applied to software development (Kan et al. 1994) with some success.

The more narrow view of statistical quality control has been attempted for software development as well. For example, Okumoto (1985) uses a logarithmic Poisson model to predict failure intensity and control if the progress lies within specific limits. Weller (2000) describes statistical process control for software development and gives examples of establishing upper and lower control limits for problem arrival rates or inspection rates. Because of the inherent differences between manufacturing and development processes, these approaches are not accurate in software development and, therefore, have not been widely adopted in practice.

Kamei et al. (2013) proposed just-in-time quality assurance for software development. They predict defect-prone changes at check-in time to reduce effort inspecting changes. They can reduce the effort to inspect changes by 80 %. While this brings feedback closer to the developer and the corresponding task, it focuses on the narrow aspect of defect proneness and only gives feedback at check-in time.

While total quality control and management, especially continuous improvement, fit well to software development projects, statistical quality control has not gained broad acceptance. Just-in-time quality assurance has focused on a very narrow area of feedback so far.

Preliminary work We formulated the ideas of quality control and continuous improvement in software as continuous software quality control (Deissenboeck et al. 2008a, Wagner 2013). We see this to be the direct countermeasure against quality decay because it describes the process to measure and evaluate the quality of the software product frequently. This includes manual analysis such as reviews and inspections as well as tool-based analyses. The latter are especially interesting because they can be applied together with an hourly or nightly build. Such tools, for example for bug pattern identification or clone detection, are often integrated in dashboard tools to present an overview of the quality data of a software system. We have gathered experience in this area with our own tool called ConQAT, which can measure properties such as cloning directly but also easily integrates other measurements.

Developer information needs

The concept of information needs was coined in the area of book libraries and documentation. Wilson (1994) shows in his review that there is an extensive theory of information needs of an information seeker as well as the information seeking process. Other research domains, such as organisational decision making and marketing, have developed similar concepts and contain further theories and experimental results.

The only mentioning of information needs theory in software engineering is by Jedlitschka et al. (2013). Yet, they apply it to the software manager's information need for technology selection. Fritz and Murphy (2010) do not explicitly discuss the information needs concept but investigate questions developers ask. They include partly quality-related questions such as “What code is related to a change?” A full quality information needs theory is missing, however.

There is a rich theory on information needs in other disciplines. In software engineering, there is little work yet on developers' information needs. There is no sufficient theory on needs for quality-related information.

Dashboards, IDEs and interaction history

Dashboards are tools that integrate and aggregate the data from various sources to present them in a useful way to developers or quality engineers. Well known examples include Bauhaus, SonarQube or Hackystack (Johnson 2007). The primary presentation interface for dashboards is a web browser. Newer dashboards now also have integrations into integrated development environments (IDEs). There is an Eclipse plugin for SonarQube, and Teamscale (Heinemann et al. 2014) provides plugins for Eclipse and

Visual Studio. Both mostly present the lists and diagrams directly in the IDE. The interaction only allows to see details of the provided findings.

Recently, there have been advances in using the interaction histories of developers to adapt IDEs to their information needs. Murphy et al. (2006) did an initial study on how Java developers use Eclipse based on their interaction history. They employ the Mylar Monitor (http://www.eclipse.org/mylar) to record traces of usage in the IDE and derive the usage of different views and features. Ying and Robillard (2011) could also relate when developers were editing code to what kind of task they were working on. Kersten and Murphy (2006) proposed to capture and model tasks in a task context model. They are able to show only the task-related parts of the IDE to the developer. This resulted in the successful open-source project Mylyn.

A variety of dashboards provide quality feedback to developers, mostly in a web browser but also with first IDE integrations. Task models of IDE users help in focusing what to show developers according to their task. Yet, task modelling and IDE adaptation has not been used for broader quality feedback.

Preliminary work We have gathered experience in the area of dashboards with our own tool called ConQAT (Deissenboeck et al. 2008a) which can measure properties such as cloning directly but also easily integrates other measurements. It is also the basis for the incremental dashboard tool Teamscale.

Static analysis

Static analysis is the assessment of code without executing it. Several static analyses are already integrated into common IDEs. For example, checking coding guidelines or parsing and compilation are all static analyses. More sophisticated analyses, such as bug pattern detection (Hovemeyer and Pugh 2004), clone detection (Li et al. 2006) or architecture conformance analysis (Murphy et al. 1995), are more or less deeply integrated into IDEs as well. The problem is that often the information is not found easily, for example hidden in some window with warnings (Ostberg et al. 2013). Furthermore, as static analysis is easy to execute but knows little about the semantics of the software, it tends to produce high numbers of false positives (Wagner et al. 2005, Zheng et al. 2006). The danger is that developers discard the warnings altogether and valuable information is ignored.

Gode and Koschke (2009) were the first to propose incremental clone detection to give developers feedback on cloning more quickly. Kawaguchi et al. (2009) even brought incremental clone detection directly to the IDE but do not provide a clear solution to updating the clone information. Hummel et al. (2010) make clone detection instantaneous and scalable using an index-based approach.

Static analysis has not only been used to analyse the program statements but also the comments contained in the source code. Khamis et al. (2010) propose the tool *JavadocMiner* for analysing the quality of JavaDoc comments using readability indices and word-type count heuristics. Steidl et al. (2013) go beyond that by automatically classifying different types of comments and provide heuristics specific for the types. They could show the agreement of their analysis with expert opinion.

Many static analyses are able to provide interesting results but are not readily executable during changes; they also provide little guidance on how to act on the results.

Preliminary work We started working on the effectiveness and efficiency of bug pattern tools in 2004 with a comparison of such tools for Java with reviews and tests (Wagner et al. 2005). We found that the tools can detect a subset of the defects detected by reviews but a set of defects disjoint from the one found by tests. We extended this with a further study at a different company and found, among other results, that we can reduce the rate of false positives significantly by spending time on carefully configuring the analysis tools (Wagner et al. 2008). More recently, we investigated the possibilities for using static analysis at SMEs (Gleirscher et al. 2013) and proposed an experiment setup to discover further barriers in using static analysis (Ostberg et al. 2013).

A second thread of investigation in the area of static analysis has been clone detection. We performed a widely cited study on the effects of cloning, especially in the context of inconsistent changes (Juergens et al. 2009). We found that every other inconsistent change during which the developer was not aware of the clones, led to a defect in software systems in the field. We were also the first to study cloning in Matlab Simulink models in an industrial environment (Deissenboeck et al. 2008b). We could show that cloning is a problem in modelling as well. Similarly, we were able to analyse a set of industrial requirements specifications for cloning where we also found substantial problems with these duplications (Juergens et al. 2010). More recently, we investigated techniques to detect semantic clones by extracting code chunks, generating tests and comparing the I/O behaviour (F. Deissenboeck et al. 2012).

Test generation

There have been considerable advances in automatically generating test cases in recent years. They have enabled guided test generation to achieve a high coverage of code structure. For example, Randoop (Pacheco et al. 2007) incrementally generates method sequences based on random selection and previously constructed sequences. Similarly, DART (Godefroid et al. 2005) aims at exercising new execution paths by a dynamic analysis of executions under random testing. They see the predicates in a path as path constraints and combine concrete executions with symbolic executions.

Also search-based software engineering has been used to generate test cases. McMinn (2004) applies metaheuristic search techniques to cover program structures or exercise specific program features. Dallmeier et al. (2012) describe TAUTOKO, a test-case generation approach specifically for mining specifications by systematically generating test cases to extend the execution space.

Modern techniques for automatic test generation can systematically cover the execution space. These techniques have not been employed for generating immediate and interactive feedback for developers.

Preliminary work We have worked on test case generation based on behavioural models (Pretschner et al. 2005). This is often called model-based testing. We can employ the experience from generating test cases for real-life systems in this project.

Mining and prediction

Over the last years, researchers have made use of the increasing number of publicly available software repositories from open-source projects to (semi-)automatically mine interesting insights into relationships in the product, process and team. One particular area on which we will concentrate is the prediction of likely changes to other files after a file has been changed (co-changes). Gall et al. (2003) analysed the data from the CVS of a picture archiving and communication system. They identified change couplings to find weaknesses in the architecture. Ying et al. (2004) extended the idea by giving recommendations for likely co-changes. They mined the repositories of Eclipse and Mozilla and were able to find interesting recommendations for co-changes. Similarly, Zimmermann et al. (2005) predicted how changes will propagate over a system using static attributes and change histories. Using eight open-source systems, they could show that their results support developers in navigating the source code and preventing errors. There are implementations of Eclipse plug-ins (http://www.st.cs.uni-saarland.de/Eclipse/) bringing specific analysis results into the developers' working environment using local analyses. They have not been integrated into a comprehensive quality feedback system which prioritises the hints.

Kim and Ernst (2007) combine repository mining with static analysis to prioritise the warnings generated by static analysis according to whether defects of the same category have been fixed in the history of the system.

Repository mining has become a useful means of predicting likely co-changes; the recommended co-changes have not been prioritised and incorporated into other feedback systems.

Preliminary work We have worked on using Bayesian nets as a basis for predicting software quality (Wagner 2010), sensitivity analysis to simplify prediction models (Wagner 2007) and a comparison of expert-based and machine-learning-based models for software quality prediction (Lochmann et al. 2013). Recently, we conducted an industrial case study on finding coupled changes in the Git history of a Java system and qualitatively analysed how interesting the results are using interviews with the developers. We found that recommendations on co- changes were considered very interesting by both novice and senior developers. The study is not yet published.

Quality models

Researchers have worked on quality models for several decades to better capture what software quality is. This resulted in a large number of available quality models (Kläs et al. 2009). Quality models could, in principle, provide a solid foundation for quality measurement and understanding how a measurement should be interpreted. We found in a survey of practitioners, however, that the existing standard quality models contain a gap between the high-level quality attributes and concrete quality measurement (Wagner et al. 2009, Wagner et al. 2012a). Therefore, modern quality models aim to bridge that gap. The French Squale project (Mordal-Manet et al. 2009 and others (Marinescu et al. 2005, Schackmann et al. 2009) proposed new quality models by aggregating quality measures.

Contemporary quality models provide links between measurement and high-level quality goals; therefore, they can provide rationales in developer feedback which have not been used directly in IDEs.

Preliminary work To address the issues with standard quality models, we developed an operationalised quality model (Wagner et al. 2012b, Wagner et al. 2015) in the German research project Quamoco (http://www.quamoco.de). We have been able to build a base quality model that covers a broad range of quality attributes but is concrete enough to be measured for the programming languages Java and C# but explains for each measurement the relationship to quality attributes. We also developed an editor for the Quamoco quality models which integrates with ConQAT for analyses and aggregations (Deissenboeck et al. 2011).

API Documentation and online discussions

Today's programming consists of the frequent usage of existing APIs. The multitude of APIs one needs to understand can be problematic. Therefore, various researchers have worked on ways to support a developer in API usage. In this proposal, we will concentrate on support via the extraction of information from existing textual sources.

The primary source for such information is the API documentation. Yet, this documentation is also large and information might be difficult to be found. Dekel and Herbsleb (2009) proposed a tool to display API documentation directly in the IDE while developers are using the API. They found that with this documentation, developers were more successful in fixing bugs.

Furthermore, researchers increasingly make use of other online resources, in particular online discussion forums. Hou and Li (2011) manually analysed newsgroup discussions on frameworks and APIs to learn about obstacles. During their analysis, they found that it can be a challenge to find relevant discussions on an API manually. Nasehi et al. (2012) analysed the code examples in StackOverflow and found that they are most useful if they are accompanied by corresponding explanations. They also found that 65 % of the questions in StackOverflow are related to some API. Ponzanelli et al. (2013) already developed a tool called Seahawk which tries to fetch interesting discussions from StackOverflow depending on the context the developer is working on.

Focused information on APIs can increase the effectiveness of developers; online discussions contain valuable information about API usage and obstacles. This has not been used to give interactive feedback on software quality.

Related projects

The DFG SPP 1593 is most closely related to this project proposal. The stated problems of legacy systems and continuous evolution in the SPP also are reasons why focused developer feedback makes sense. In the SPP, the projects ADVERT and URES are most closely related. ADVERT aims at keeping artefacts consistent during the evolution which we partially support by fast feedback about clone detection and coupled change analysis. URES captures links between artefacts automatically and gives rationales. Both projects do not aim at presenting the analysis results directly and continuously to the developers.

The SPECMATE project, mostly funded by an ERC Advanced Grant, is related because it has a strong emphasis on test case generation. Their focus is not on giving continuous feedback to the developers.

Summary

Continuous quality control is an approach to address quality decay in software regularly during a software project. Advances in static analysis, test generation and repository mining allow us to give further feedback to developers, potentially just-in-time while performing changes. These analyses have not been incorporated into a joint feedback system that gives focused hints. Contemporary quality models, dynamic slicing and online discussions could even provide rationales for the feedback to support its acceptance and understandability.

Project-related publications

- S. Wagner, A. Goeb, L. Heinemann, M. Kläs, C. Lampasona, K. Lochmann, A. Mayr, R. Plösch, A. Seidl, J. Streit, and A. Trendowicz. Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology*, 62:101–123, 2015.
- S. Wagner, F. Deissenboeck, M. Aichner, J. Wimmer, M. Schwalb. An Evaluation of Two Bug Pattern Tools for Java. In: *Proc. 1st IEEE International Conference on Software Testing, Verification and Validation (ICST 2008)*, pages 248-257. IEEE Computer Society Press, 2008.
- F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. Mas y Parareda, M. Pizka. Tool Support for Continuous Quality Control. *IEEE Software*, vol. 25, no. 5, pp. 60-67, 2008.
- S. Wagner. A Bayesian Network Approach to Assess and Predict Software Quality Using Activity-Based Quality Models. *Information and Software Technology*, vol. 52, no. 11, pp. 1230-1241, 2010.
- S. Wagner, J. Jürjens, C. Koller, P. Trischberger. Comparing Bug Finding Tools with Reviews and Tests. In: *Proc. 17th IFIP International Conference on Testing of Communicating Systems (TestCom '05)*. Volume 3502 of LNCS, pages 40-55. Springer-Verlag, 2005.

- E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner. Do Code Clones Matter? In: *Proc. 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, 2009.
- M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner. Introduction of static analysis in small- and medium-sized software enterprises: Experiences from technology transfer. *Software Quality Journal*, 22(3):499–542, 2015.
- J.-P. Ostberg, J. Ramadani, and S. Wagner. A novel approach for discovering barriers in using automatic static analysis. In *Proc. International Conference on Evaluation and Assessment in Software Engineering (EASE'13)*. ACM, 2013.
- F. Deissenboeck, L. Heinemann, B. Hummel, and S. Wagner. Challenges of the dynamic detection of functionally similar fragments. In *Proc. 16th European Conference on Software Maintenance and Reengineering (CSMR'12)*. IEEE, 2012.
- A. Abdulkhaleq and S. Wagner. A Controlled Experiment for the Empirical Evaluation of Safety Analysis Techniques for Safety-Critical Software. In: *Proc. 19th International Conference on Evaluation and Assessment in Software Engineering (EASE 2015)*. ACM, 2015.

Objectives, concept and approach

Objectives

Motivation Software is a unique engineering product: It is intangible, it can be copied without cost and there is no need for production but only development. Nonetheless, we have learned that it needs to be built with the same rigour as any other engineering product or the quality, schedule and cost can be disappointing. The intangible nature of software is a factor which complicates working with it. Developers have difficulties overlooking all consequences of changes made to a software system. Often problems do not arise directly but only complicate later comprehensibility or introducing further changes which can be many years in the future. Humans are particularly bad in handling actions with effects with such long feedback loops (Kahneman 2012).

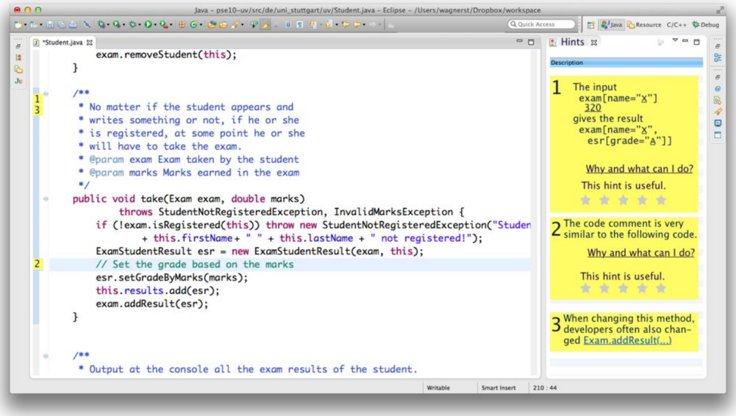
A lot of work in software engineering research and practice aims at shortening these feedback cycles. For example, iterative and agile development processes run in iterations with only a few weeks length to get feedback from users and customers early. Continuous integration (Beck 1999) and deployment provides a developer an immediate information if they broke the build or the software could not be deployed. We have worked on *continuous software quality control* (Deissenboeck et al. 2008a, Wagner 2013) which incorporates various quality analyses into build and integration processes which are performed continuously. Our intention has been to avoid quality decay or *software ageing* (Parnas 1994) occurring unnoticed and causing problems later in a software's evolution. A very successful example of such a continuous quality control technique is clone detection: the discovery of code that has been copied. Code clones cause additional effort in comprehending and maintaining a system as well as cause defects to stay in the code (Juergens et al. 2009). We and others have developed tool support that gives the developers feedback on the state of cloning in their system after each analysis execution. It helps to make the intangible software more comprehensible and to make quality decay visible.

At the same time, ever more powerful IDEs, for example Eclipse, help us to automate all kinds of tasks in programming and design. We do not need to wait until we get feedback from the compiler: several IDEs now offer immediate parsing and background compilation, i.e. instant feedback on what you type. **In contrast, continuous quality control and current just-in-time quality assurance still requires that the code be written, compiled and submitted to the version control system to get feedback. To some degree, the damage has already been done by then.** The developer has probably started working on another task by then and needs to remember what the change was about. Hence, *continuous quality control* as promoted today does not really prevent quality decay but helps to detect and remove it early.

As hardware becomes ever faster and more useful software analyses become available, we need to provide useful feedback to the developers immediately. I propose **continuous and focused developer feedback** (CoFoDeF) on quality-related issues in the code (and potentially other artefacts) that is being changed or written. Binkley (2007) sees an emerging trend to enable "source-code analysis performed at edit time, compile time, link time, and run time." The main challenges lie in (1) **understanding when a developer needs which information**, (2) **making interesting analyses immediate and continuous,** (3) **focusing on the most important feedback** so that the developers are not overwhelmed and (4) **provide rationales for the hints** so that the developers know how to act in response.

A User's Perspective In the long term, the users of the project's results will be software developers. Continuous feedback will happen right where changes to the code are made: the IDE. Fig. 1 shows a mock-up of what the integration might look like. On the left we have a window with the source code of a Java method. On the right, there is a *Hints* window presenting the most important hints for the current work of the developer.

In this example, the developer scrolled to this method and brought the cursor to the comment marked with the yellow box containing a "2". The CoFoDeF tool creates a test case for the method in the background and shows it as hint number "1". The hint is also interactive in the sense that the underlined values of the test can be changed and a new feedback is immediately calculated. As the comment is unnecessary because the code statement directly after it contains the same information, the tool presents this as hint number "2". There are also direct links to rationales ("Why and what can I do?") and a possibility to rate the hint. A third hint provides a link to a method that probably has to be changed when the present method is changed. When the developer actually changes the method, this hint would move higher in priority to reflect the task performed by the developer.


Java – pse10-uv/src/de/uni_stuttgart/uv/Student.java – Eclipse – /Users/wagnerst/Dropbox/workspace
Quick Access
Java
Resource
C/C++
Debug
*Student.java
Hints
Description
Writable
Smart Insert
210 : 44

```
        exam.removeStudent(this);
    }

    /**
     * No matter if the student appears and
     * writes something or not, if he or she
     * is registered, at some point he or she
     * will have to take the exam.
     * @param exam Exam taken by the student
     * @param marks Marks earned in the exam
     */
    public void take(Exam exam, double marks)
            throws StudentNotRegisteredException, InvalidMarksException {
        if (!exam.isRegistered(this)) throw new StudentNotRegisteredException("Studen
                + this.firstName+ " " + this.lastName + " not registered!");
        ExamStudentResult esr = new ExamStudentResult(exam, this);
        // Set the grade based on the marks
        esr.setGradeByMarks(marks);
        this.results.add(esr);
        exam.addResult(esr);
    }

    /**
     * Output at the console all the exam results of the student.
```

1 The input exam[name="X"] 320 gives the result exam[name="X", esr[grade="A"]]

Why and what can I do?

This hint is useful.

2 The code comment is very similar to the following code.

Why and what can I do?

This hint is useful.

3 When changing this method, developers often also changed Exam.addResult(...)

Figure 1.

Mockup of continuous and focused feedback in an IDE

Objectives The underlying aim of this project is to reduce quality problems in the development and usage of software systems. The concrete objective is to **understand the quality information needs of developers** and to **provide the results of modern analysis techniques** at the **right moment and context** in a **focused and understandable way** and **empirically evaluate the usefulness of such feedback**.

Continuous Feedback from Analyses [C]

Objective C. Create a concept for providing continuous feedback from quality analyses for developers.

Quality information needs theory [C1] We need a solid theoretical basis about what information about quality developers need during their work. We build on the existing work on information needs in other disciplines. We will conduct a developer study with students to investigate the small-scale information needs during typical development tasks. Investigating students will allow us to reach a larger sample size while the differences between students and practitioners are often small (e.g. Salman et al. 2015). We will develop a theory for quality information needs based on existing information needs theories, results from existing studies and our developer study.

Analyses before changes [C2] We want to be able to run analyses before a change to the system showing hints to the developers before they start the change. These hints will contain information about what the developers should take care of. The initial analysis used will be clone detection because of existing work on presenting clone information inside an IDE, for example in ConQAT. It gives the developer an indication that there are copies of this code that might have to be changed as well. We will complement clone detection by coupled change analysis based on repository mining to show other code that probably needs to be changed as well. The techniques for recommending coupled changes has been well investigated, has shown to provide useful recommendations and is different in analysis object and goal.

Analyses while changing [C3] Different analyses will be interesting while a change is conducted. We will also start with clone detection as an example using incremental, index-based clone detection [20]. Therefore, we will be able to give a hint to developers directly after they cloned code to refactor it. The second analysis will be a comment quality analysis to show that we can provide hints going beyond code statements. We will focus on giving hints to the developers directly after they added or changed a comment about its quality relying on the method by Steidl, Hummel and Juergens [51]. The third analysis we investigate will generate hints by contemporary automated test generation. This analyses allows us to explore dynamic analyses and higher degrees of interactivity between the developer and the analysis.

Focusing Feedback and Rationales [F]

Objective F. Create a concept to focus the provided feedback and provide rationales so that the feedback is useful for the individual developers in their specific contexts.

Context-specific prioritisation [F1] To avoid overwhelming the developers with long lists of hints, we will continuously determine the context in which the developer is and adapt the shown hints and their order accordingly. One component in the context will be the tasks and activities performed by the developer. A result from a generated test might be more interesting while a developer adds a new feature than during a code refactoring. We will derive the current activity by observational studies of the developer's actions. The second component will be the time span since the last change by this developer. For example, a probable co-change is more interesting if the last change was made long ago.

Learning prioritisation [F2] The second goal for providing useful hints is to enable developers to give feedback manually. It will be possible to black-list each hint ("Do not show this hint again."). In addition, each hint can be rated with a simple five-star rating (up to "This hint is very useful to me."). In combination with the context detection [F1], we will record these ratings and learn under which contexts what hints are more important. Beyond these manual ratings, we want to explore automatic learning for improving the presentation of hints to the developers. It will observe the actions of the developers and if they react to the hints shown. For example, if a probable co-change was presented in a hint, does the developer go there and also change something? This will indicate if the hint was useful and should be presented again under similar circumstances.

Rationales [F3] Create a concept for rationales for feedback so that developers understand and get help to act on it. Several modern quality models provide a link from measurements to quality attributes, sometimes with clear explanations (Wagner et al. 2015). Both can be helpful for developers for understanding the quality impact of a hint. Therefore, we will include such quality models and extract their contents for giving rationales, for example why introducing a clone is problematic. In addition, web discussion sites, such as StackOverflow, contain a lot of information on programming, programming languages, libraries and all sorts of problems with these. Building on existing extraction techniques for such web resources, we will look for rationales for the provided hints just-in-time and provide them to the developer. For example, after generating a test case for a new piece of code, we will look for related information on the called libraries in this code which might be interesting to interpret the test result. Finally, dynamic slicing (Agrawal and Horgan 1990, Korel and Laski 1990) is a method to reduce a program to those statements that influence the output in a given run. This helps a developer understand how a certain result is created. Therefore, this fits well for explaining the results of an automatically generated test case. We will include dynamic slicing in test-case generation and execution [C3] and provide the slice as a rationale for the test results to the developers.

Empirical Evaluation of the Concepts [E]

Objective E. Empirically evaluate the developed concepts and theory.

Experimental evaluation [E1] Accompanying the theoretical, conceptual and constructive work, we will run a series of experiments with students to investigate the hypothesis “students more likely work and successfully resolve hints if they are supported by the focusing and rationale concept.” Students will be asked to perform maintenance tasks on existing software systems. One group of students will get all the hints generated by all implemented analyses from Objective C. A second group will receive a filtered and ordered subset based on the theories and concepts developed in Objectives C and F.

Evaluation by case studies [E2] We will complement the student experiments with case studies on pilot applications at industrial partners and large student projects. As we do not expect to get a large sample size here, we will investigate another hypothesis more qualitatively: “Developers see benefits in continuous feedback and accept it as part of their daily work.” We will provide the developers with the necessary tooling developed for the other objectives and regularly interview them and observe their work.

Expected Benefits: Advancing from punch cards and receiving the results of compiling and test runs only a day after to online compilation achieved huge productivity gains. The programmers got much quicker feedback on what they had done. Continuous and focused quality feedback can be a similar step up in productivity. Certain parts, such as instant parsing and background compilation, are already implemented in modern IDEs and help the developers to avoid quality problems. We can, however, give much more feedback, e.g. on potential quality problems instantly when the product is created or changed. This will also educate the developers to avoid reoccurring of these problems.

The continuous and focused feedback approach will open up a multitude of research opportunities. New kinds of analyses can be conceived and integrated into the approach. This includes other existing techniques, such as further dynamic or static analyses or mining of other repositories, as well as completely new techniques which might yield interesting information as immediate feedback.

Work programme including proposed research methods

The project will contain *theoretical/conceptual/constructive* and *empirical* research. In detail, we will develop the quality information needs theory, create the concepts for prioritisation, rationales and learning, implement prototypes and evaluate all that in experiments and case studies. We will first discuss the work-package strategy, then the time line with milestones and, finally, descriptions of each work package.

Work-package strategy

Ten work packages structure the project into manageable units as shown in Fig. 2. In square brackets, we give the objectives each work package contributes to. The empirical evaluation (WP 9) of the results of the other work packages and project management (WP 10) are the foundation of the project. The centre are the analyses we will exemplarily employ to generate hints in interactive quality control. In WP 5, we adapt and extend existing clone detection analyses; in WP 6, we investigate our existing work on coupled changes [48] for our interactive analysis concept. Both can be applied before a change. After a change, we will employ comment quality analysis (WP 7) building on (Steidl et al. 2013) and automated test generation. For all analyses, we have crosscutting issues investigated in the final four work packages. The information needs theory will be provided by WP 1. WP 2 investigates how to prioritise hints from single and all analyses. WP 3 extracts rationales from different sources for the hints. At last, we will experiment with different distributions of the analyses over servers and clients in WP 4.

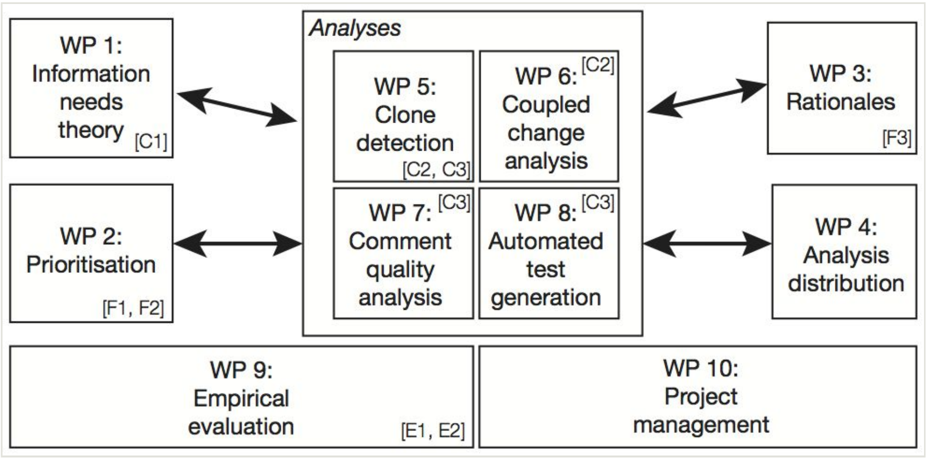

WP 1: Information needs theory [C1]
WP 2: Prioritisation [F1, F2]
Analyses
WP 5: Clone detection [C2, C3]
WP 6: Coupled change analysis [C2]
WP 7: Comment quality analysis [C3]
WP 8: Automated test generation [C3]
WP 3: Rationales [F3]
WP 4: Analysis distribution
WP 9: Empirical evaluation [E1, E2]
WP 10: Project management

Figure 2.
Work packages strategy

Work packages

WP 1: Information needs theory

Objective. The aim of this WP is to create a quality information needs theory as basis and guide for the continuous and focused feedback. The theory describes the information seeking process, shows how a developer interacts with analyses in general and what factors are relevant for the satisfaction of the developer with provided information. Furthermore, it will contain how an analysis can collect and learn from the feedback of the developer.

Content

- Developer study with students in the software engineering programme during a practical course with about 120 participants. The study will contain questionnaires, interviews and direct observations of students developing software.
- Review of existing theories and empirical results on information needs
- Building a quality information needs theory for software developers
- Development of a learning concept based on Mylyn (detailed description of sources for learning and effects)
- Initial part of the learning concept: incorporation of manual feedback on incorrect hints
- Conceptual integration of concepts for context and task understanding and activity observation (see also WP 2)
- Extending the analyses from WP 5 to 8 with the information needs and learning concepts
- Collaboration with Prof. Thomas Fritz and Prof. Albrecht Schmidt on developing the information needs theory
- Research contests at ISSTA, MSR and SAS for integration of further analyses in the interaction and learning concept (see Sec. 4.2)

Expected Results

- Information needs theory
- Learning concept
- Extensions of the analyses

Success Criteria: > 30 % higher probability that students work on and resolve hints

Planned Effort: 10 person-months

WP 2: Prioritisation

Objective. The objective of this work package is to provide a mechanism for pro- viding the developers with a prioritised list with only few selected hints most appropriate for the developer's task at that time.*Content*

- Investigation of current prioritisation mechanisms (e.g. severity classes in static analysis tools)
- Determining the developer's tasks and work context by observing the actions taken in the IDE and other development tools
- Adapting the prioritisation to the current context and task
- Offering a direct, manual feedback mechanism for developers and adapting the prioritisation accordingly
- Putting the manual feedback and context into relation to learning hint and context links
- Observing the actions taken by the developer after a hint was presented and learning if the developer acted on it
- Putting developer actions into relation to manual feedback and context to adapt prioritisation
- Running a developer contest at EclipseCon for visualising hints/warnings (see Sec. 4.2)
- Collaboration with Prof. Daniel Weiskopf and Dr. Fabian Beck on hint visualisation

Expected Results

- Prioritisation and corresponding learning concept
- Prototypical implementation

Success Criteria: > 50 % higher probability that students work on and resolve hints (in combination with WP 1)

Planned Effort: 11 person-months

WP 3: Rationales

Objective. The aim of this work package is to provide useful rationales for the hints presented to the developer mined from various sources.*Content*

- Investigation of existing rationales of warnings and hints in IDEs and static analysis tools
- Investigation of existing quality models that contain descriptions and justifications; matching to hints
- Extraction of rationales from quality models (initial candidate: Quamoco [58]) and inclusion in corresponding hints
- Investigation of means to extract discussion results from web resources
- Matching hints to web discussions and extracting discussion results to include in hints

- Investigation of existing slicing techniques

Build dynamic slicing mechanism to create rationale for dynamic results (in particular test results)

Expected Results

- Concept to include rationales extracted from quality models, web resources and slicing
- Prototypical implementation

Success Criteria: > 80 % higher probability that students work on and resolve hints (in combination with WP 1 and 2)

Planned Effort: 8 person-months

WP 4: Analysis distribution

Objective. Although it is not an explicit project objective, we will need to investigate the appropriate distribution of the analyses over the existing servers and the client running the IDE. The aim is to find a distribution concept that requires minimal dependency of the developer on server connections but a performance on the client so that the developer does not notice any disturbances by the analysis runs.

Content

- Investigation of existing distribution concepts (purely client-based, purely server-based, mix)
- Development of three distribution concepts and prototypical implementations
- Experimental comparison of different prototypes

Expected Results

- Analysis distribution concept
- Prototypical implementation
- Comparison results

Success Criteria: Students and developers do not notice performance degradation in experiments/case studies.

Planned Effort: 5 person-months

WP 5: Clone detection

Objective. Clone detection is our primary example analysis as it can be applied before and after a change and there are existing results on incremental and immediate feedback. The aim of this work package is to adapt an existing clone detection approach and tool for the application in continuous feedback.

Content

- Investigation of existing clone detection techniques and tools (in particular incremental and scaleable approaches)
- Selection of a clone detection approach and tool to adapt (initial candidate: ConQAT)
- Incorporation of the information needs and learning concept of WP 1
- Incorporation of rationales from WP 3
- Collaboration with Dr. Florian Deissenboeck, Dr. Benjamin Hummel and Dr. Elmar Juergens

Expected Results

- Concept for interactive and learning clone detection
- Prototypical implementation

Success Criteria: Functional clone detection approach and implementation

Planned Effort: 6 person-months

WP 6: Coupled change analysis

Objective. In this work package, we will add a second analysis that can be executed before a change is performed. We will build a coupled change analysis to be applied in continuous feedback.

Content

- Investigation of existing coupled change analysis approaches and tool support
- Selection of an approach and tooling for inclusion in continuous feedback
- Building sufficient tool support to include in IDE (Eclipse)
- Incorporation of the information needs and learning concept of WP 1
- Incorporation of rationales from WP 3
- Collaboration with Prof. Harald Gall and Dr. Burak Turhan on making such analyses more interactive.

Expected Results

- Concept for interactive and learning coupled-change analysis
- Prototypical implementation

Success Criteria: Functional coupled-change analysis and implementation

Planned Effort: 4 person-months

WP 7: Comment quality analysis

Objective. The aim of this work package is to extend the CoFoDeF approach by a representative analysis which does not only examines the executable code. We will investigate comment analysis. This is an analysis that can be applied after a change (i.e. after changing a comment). Therefore, we will build a comment quality analysis to be applied in continuous feedback.

Content

- Investigation of existing comment quality analysis approaches and tools
- Selection of an approach and tooling for inclusion in continuous feedback (initial candidate: Steidl et al. (2013))
- Extending and adapting the approach and tool to work in an IDE (Eclipse)
- Incorporation of the information needs and learning concept of WP 1
- Incorporation of rationales from WP 3
- Collaboration with Dr. Benjamin Hummel and Dr. Elmar Juergens

Expected Results

- Concept for interactive and learning comment quality analysis
- Prototypical implementation

Success Criteria: Functional comment quality analysis and implementation

Planned Effort: 4 person-months

WP 8: Automated test generation

Objective. The goal of this work package is to establish and validate the integration of test results as an immediate feedback for developers. In particular, we aim at automatically generating and executing test cases on the fly. Ideally, we are able to generate just the test case giving the developer the most information in his or her current context.

Content

- Investigation of existing test-case generation approaches (Randoop, DART or TAUTOKO)
- Selection of an approach and tooling for inclusion in continuous feedback (initial candidate: Randoop)
- Extending and adapting the approach and tool to work in IDE (Eclipse)
- Generation and execution of tests on the fly
- Presentation of test cases and results
- Incorporation of the information needs and learning concept of WP 1
- Incorporation of rationales from WP 3

- Collaboration with Prof. Alexander Pretschner on automated test generation

Expected Results

- Concept for interactive and learning automated test generation
- Prototypical implementation

Success Criteria: Functional test-case generation implementation

Planned Effort: 5 person-months

WP 9: Empirical evaluation

Objective. This work package concentrates the efforts on empirically evaluating the effects of continuous feedback. We aim to investigate detailed effects in student experiments as well as the overall effect in case studies. (See also the hypotheses defined in Objective E above.)

Content

- Student experiment on the effect of the information needs and learning concepts to focus on important hints. It consists of a comparison of two groups: one only with the results from clone detection and coupled-change analysis, the other with focused, interactive and learning analyses. There will be a quantitative analysis of the probability to work on a hint and the probability of successful resolution.
- Student experiment on the effect of rationales. This has the same design as the one before but one group only gas the analysis results, while the other has hints including rationales.
- Case study in student project on effect of complete approach (one-year, 10-student projects in software engineering BSc programme of the University of Stuttgart, monthly interviews and observations of each team member during development, qualitative analysis of interview transcripts and observation videos using qualitative (Glaser et al. 1968) and quantitative content analysis)
- Case study in industrial project on effect of complete approach (six month, five team members each, internal tool development at CQSE GmbH and logistics system development at AEB GmbH, monthly interviews and observations of each team member during development, qualitative analysis of interview transcripts and observation videos using qualitative and quantitative content analysis); if industrial partners are not available, this could also be conducted as part of a further student projects.

Collaboration with Dr. Florian Deissenboeck, Dr. Martin Feilkas, Dr. Benjamin Hummel and Dr. Elmar Juergens in the industrial case study at their company

Expected Results

- Quantified effect on probability to work on hints and on successful resolution of hint of information needs and learning concept as well as rationales

- Qualitative theory on acceptance and further effects of continuous feedback

Success Criteria

- Student experiments with significant results for $\alpha < 0.05$ and statistical power > 0.8.
- Qualitative case study results with an inter-rater agreement for codes $\alpha > 0.8$

Planned Effort: 16 person-months

WP 10: Project management

This work package contains the management of the whole project. This includes the coordination of the other work packages, organisation of regular workshops of the team members and with collaboration partners, assessing the project progress, updating the project plan and all necessary reporting. Furthermore, this work package also contains the organisation and execution of the programming and research contests, i.e. negotiating with partners, creating marketing materials, distributing calls, organising prizes and rooms and publicising the results.

Planned Effort: 3 person months

Data handling

We will publish all data not classified confidential from the collaborating companies or personal data as technical reports to make them available to the community. Where legally possible, we will make the data and source code available on ZENODO, a repository for open research data.

Other information

We will deeply integrate this research into our teaching and further collaborations with industry. We plan to provide a series of supervised student projects (BSc theses, MSc theses and guided research projects) as well as to integrate results as part of our lectures on software engineering in particular the master-level course "Software Quality Assurance and Maintenance". Furthermore, we plan to publish the main results of the project in an open-access journal article.

Descriptions of proposed investigations involving experiments on humans, human materials or animals

The research project contains several experiments which involve humans to investigate information needs. All research activities will comply with the Declaration of Helsinki of ethical principles regarding human experimentation.

Information on scientific and financial involvement of international cooperation partners

There are no cooperations planned where agreements with other research foundations would be concerned.

Budget

Basic module

Funding for staff

Non-doctoral research staff: We apply for the funding of two research assistants with the opportunity to prepare for a doctoral degree over the whole three years. In computer science, it is only possible to attract PhD students by offering full-time positions. The PhD students will contribute 72 person-month in total (with 12 person-month per year). One of the students will focus on information needs (WP 1), coupled change analysis (WP 6), prioritisation (WP 2) and comment quality analysis (WP 7), the other on analysis distribution (WP 4), clone detection (WP 5), rationales (WP 3) and automated test generation (WP 8) while both work together on the empirical studies (WP 9) and project management (WP 10).

The project needs technical infrastructure to develop tool support. Hence, we need a dedicated test server and a development server (see below). The build-up, development and administration will be covered by the full-time technical administrator of my group.

Student assistants: We apply for funding of two student assistants working 8.5h/week at the local standard rates at Uni Stuttgart (€ 500/month). These students will support the research assistants in planning and executing the experiments as well as the case studies (WP 9). For example, they will prepare example software the study subjects have to modify, implement online questionnaires for data collection and take care of videotaping developer observations. The student assistants will also help in the implementation of the various tool prototypes.

Direct project costs

Equipment up to € 10,000, software and consumables

Standard workstations/laptops for the staff will be supplied using the funds of the research group.

We will need a dedicated development server to facilitate modern software development. It will hold the source code repository, an issue tracker, a mailing list and a development Wiki for documentation. We will also use it to serve as public web server for information to the public. This allows us to make the development artefacts and source code public easily when we choose to open source the software. We plan to use a MacPro server for that. With the education price, this will cost us € 3,999.

Code analysis and data mining are often very expensive computing tasks. Hence, for an efficient and usable computation of the quality control knowledge and fast queries, we request a dedicated test server as quality control server. We plan to use a powerful Mac Pro which costs € 9,648.

Travel expenses For the case study with CQSE, both research assistants will have to travel to Munich seven times and stay for three days to conduct the observations and interviews. This gives a cost of € 5,600 in total. The case study with AEB does only cost local transport fees. We plan with € 100.

The assistant working on information needs and repository mining will visit Harald Gall and Thomas Fritz twice for four weeks each to collaborate on these tasks. For train tickets and hotel, we expect € 4200 in total.

We plan to publish the results primarily in international conferences. The most relevant ones for us are the International Conference on Software Engineering (ICSE), the Automated Software Engineering Conference (ASE), the International Conference on Software Maintenance and Evolution (ICSME) and the International Conference on Software Analysis, Evolution, and Reengineering (SANER) and their co-located workshops.

Therefore, we calculate costs of € 1,800 for participating in SANER 2017, € 1,800 for participating in ICSE 2017, € 1,800 for participating in ASE 2017, € 1,800 for participating in SANER 2018, € 3,600 for participating in ICSE 2018 (both research assistants), € 1,800 for participating in ASE 2018, € 3,600, € 1,800 for participating in ICSME 2018 and € 1,800 for participating in SANER 2019.

Other: We request specific funds for compensating experiment subjects. We plan to perform two experiments with our students (WP 9). To not interfere with the grading of regular subjects, we will conduct the experiments outside of normal courses. To motivate the students to participate, we will pay them a small compensation based on the salary for student assistants. In particular, we plan to have about 30 participants with 8 hours of work each in each experiment. In total, these are costs of € 7,500.

Project-related publication expenses: Finally, we plan to publish a journal article about the overall resulting approach and its empirical evaluation at a journal with open access option which will result in publication costs of € 2,000.

Module: Project-Specific Workshops

We will seek the help of the Eclipse community in WP 2 and run a programming contest at the next Eclipse conference for the easiest-to-understand graphical interface for displaying hints to developers. We want to stimulate discussion on that topic in the community and benefit from creative ideas. Furthermore, it will make our project already more widely known. As an incentive for developers to submit a visualisation proposal, we plan to award the best proposal (as evaluated by a small committee) with a prize of € 500. We aim to hold the contest at the next international EclipseCon 2017. We plan travel costs for two organisers of the programming contest at € 1,800 each. So, we expect a total cost of € 4,100.

In WP 1, we plan a second series of contests targeted at three research communities. As the four analyses we have chosen to investigate are only exemplary, we want other researchers to include their analyses into our continuous feedback concept. This disseminates our results into the community and allows us to investigate further analyses in the case studies in year 3. This will contain three workshops/events at corresponding scientific conferences. We plan to run a challenge at the International Conference on Mining Software Repositories in 2017, a workshop together with the International Symposium on Software Testing and Analysis 2017 and a workshop at the Static Analysis Symposium 2017. The incentive for the researchers to participate will be the publication of the companying papers in the conference/workshop proceedings. To intensify this incentive, I also plan a subsequent journal special issue on continuous feedback in, for example, the Automated Software Engineering Journal. I plan to have two organisers at each challenge or workshop. The first organiser will be the applicant. The second one will be one of the project staff for whom we expect travel costs of € 1,800 per workshop. In total this gives € 7,200.

Timeline

We show the distribution of the work packages over the three years of the project in Fig. 3. In year 1, we will start with the information needs theory (WP 1) with the initial example of clone detection before the change (WP 5) and add the coupled change analysis (WP 6). In parallel, we will work on how to distribute the analyses over servers and development clients. We will also start first empirical evaluations.

In year 2, we will refine, rework and empirically test the information needs theory based on the experience from year 1. We will continue to work on clone detection (after the change) and add two new analyses: comment quality analysis (WP 7) and automated test generation (WP 8). Also for those analyses, we will investigate how to distribute the analysis. Finally, we will start working on the prioritisation of the hints generated by the analyses. All this work will be accompanied by experiments to validate the feedback approach.

In year 3, we will finalise the prioritisation and extract rationales (WP 3) from various sources to be included in the hints. The main focus, however, will lay on the empirical evaluation in larger case studies.

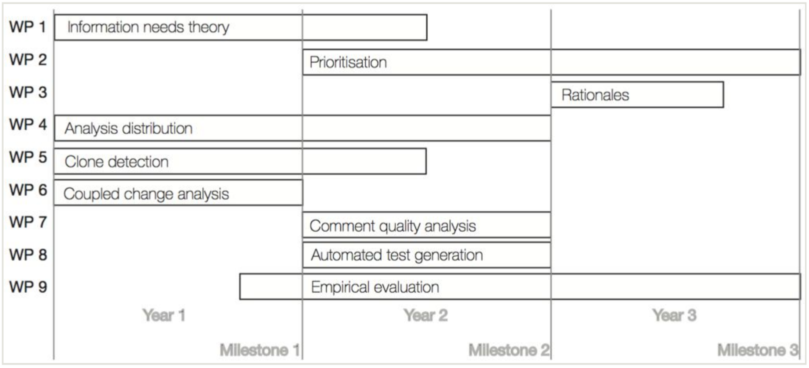


Figure 3.

Distribution of the work packages over time, with milestones

The milestones and success criteria for all three years are summarised in Table 1.

Table 1.

Milestones and success criteria (with corresponding objectives in square brackets)

Milestone 1 (Year 1)	unctional clone detection before analysis and coupled change analysis [I1], initial in- teraction and learning concept [I3], initial analysis distribution solution, initial empirical evaluation in student experiments with acceptable performance impact on the IDE and > 30 % higher probability of students to work on and successfully resolve a hint with the interaction and learning than without
Milestone 2 (Year 2)	Complete interaction and learning concept [I3], functional clone detection after analy- sis, comment quality analysis and automated test generation [I2], analysis distribution solution, context-specific prioritisation [P1], empirical evaluation in student experiments with no noticeable performance impact on the IDE and a > 50 % higher probability of students to work on and successfully resolve a hint with interaction, learning and prioritisation than without
Milestone 3 (Year 3)	Final prioritisation solution [P2, P3], rationales [R1, R2, R3], student experiments with a > 80 % higher probability of students to work on and successfully resolve a hint with interaction, learning, prioritisation and rationales than without, case studies with a large student project and an industrial project showing acceptance and benefits of continuous feedback in a realistic environment over six month

Project requirements

Employment status information

Wagner, Stefan: Full, permanent professorship at the University of Stuttgart

Composition of the project group

I have two senior research assistants financed by the university who work on topics closely related to the content of this project. Therefore, they will work with the project team and dedicate substantial parts of their working time to it.

1. Wagner, Stefan, Prof. Dr., full professor, financed by the University of Stuttgart: Project leader, supports project management (WP 10), research contests/ workshops (WP 1), information needs theory (WP 1) and extracting rationales from quality models (WP 3)
2. Ostberg, Jan-Peter, research assistant, financed by the University of Stuttgart: works on barriers on the usage of static analysis in his PhD project; he will lead the tool development parts of the project as he is also an experienced developer; he will support prioritisation (WP 2) and clone detection (WP 5).
3. Ramadani, Jasmin, research assistant, financed by the University of Stuttgart: works on developer support by repository mining in his PhD project; he will support the coupled change analysis (WP 6) and the empirical evaluations (WP 9).

Project-relevant cooperation with commercial enterprises

The cooperation with CQSE and AEB in case studies is described in above. The cooperation is solely for conducting empirical studies.

Project-relevant participation in commercial enterprises

I am working as a freelancer in software development. There is no direct relationship to the project proposal.

Acknowledgements

I would like to thank Daniel Graziotin who reviewed the proposal and gave valuable feedback.

Funding program

This proposal has been submitted to the DFG Research Grant programme.

Hosting institution

Institute of Software Technology, University of Stuttgart

References

- Agrawal H, Horgan J (1990) Dynamic program slicing. ACM SIGPLAN Notices 25 (6): 246-256. DOI: 10.1145/93548.93576
- Beck K (1999) Embracing change with extreme programming. Computer 32 (10): 70-77. DOI: 10.1109/2.796139
- Binkley D (2007) Source code analysis: A road map. Proc. Future of Software Engineering (FOSE'07). IEEE.
- Dallmeier V, Knopp N, Mallon C, Fraser G, Hack S, Zeller A (2012) Automatically Generating Test Cases for Specification Mining. IEEE Transactions on Software Engineering 38 (2): 243-257. DOI: 10.1109/tse.2011.105
- Deissenboeck F, Heinemann L, Herrmannsdoerfer M, Lochmann K, Wagner S (2011) The Quamoco tool chain for quality modeling and assessment. Proc. 33rd International Conference on Software engineering (ICSE'11). ACM DOI: 10.1145/1985793.1985977
- Deissenboeck F, Juergens E, Hummel B, Wagner S, Mas y Parareda B, Pizka M (2008a) Tool Support for Continuous Quality Control. IEEE Software 25 (5): 60-67. DOI: 10.1109/ms.2008.129
- Deissenboeck F, Hummel B, Juergens E, Schaetz B, Wagner S, Girard J-, Teuchert S (2008b) Clone detection in automotive model-based development. Proc. 30th Int. Conf. on Software Engineering (ICSE'08). DOI: 10.1145/1368088.1368172
- Dekel U, Herbsleb JD (2009) Reading the documentation of invoked API functions in program comprehension. 17th International Conference on Program Comprehension (ICPC '09). IEEE
- F. Deissenboeck LH, Hummel B, Wagner S (2012) Challenges of the dynamic detection of functionally similar fragments. Proc. 16th European Conference on Software Maintenance and Reengineering. IEEE
- Feigenbaum A (2002) Total Quality Management. Encyclopedia of Software Engineering. URL: http://dx.doi.org/10.1002/0471028959.sof359 DOI: 10.1002/0471028959.sof359
- Fritz T, Murphy G (2010) Using information fragments to answer the questions developers ask. Proc. 32nd International Conference on Software Engineering (ICSE '10). ACM DOI: 10.1145/1806799.1806828
- Gall H, Jazayeri M, Krajewski J (2003) CVS release history data for detecting logical couplings. Proc. 6th International Workshop on Principles of Software Evolution. IEEE DOI: 10.1109/iwpse.2003.1231205
- Glaser B, Strauss A, Strutzel E (1968) The Discovery of Grounded Theory; Strategies for Qualitative Research. Nursing Research 17 (4): 364. DOI: 10.1097/00006199-196807000-00014
- Gleirscher M, Golubitskiy D, Irlbeck M, Wagner S (2013) Introduction of static quality analysis in small- and medium-sized software enterprises: experiences from technology transfer. Software Quality Journal 22 (3): 499-542. DOI: 10.1007/s11219-013-9217-z
- Godefroid P, Klarlund N, Sen K (2005) DART: directed automated random testing. Proc. Programming Language Design and Implementation (PLDI '05). ACM DOI: 10.1145/1065010.1065036

- Gode N, Koschke R (2009) Incremental clone detection. Proc. 13th European Conference on Software Maintenance and Reengineering (CSMR'09). DOI: 10.1109/csmr.2009.20
- Heinemann L, Hummel B, Steidl D (2014) Teamscale: Software quality control in real-time. Proc. International Conference on Software Engineering (ICSE'14). DOI: 10.1145/2591062.2591068
- Hou D, Li L (2011) Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. Proc. 19th International Conference on Program Comprehension (ICPC'11). IEEE DOI: 10.1109/icpc.2011.21
- Hovemeyer D, Pugh W (2004) Finding bugs is easy. ACM SIGPLAN Notices 39 (12): 92. DOI: 10.1145/1052883.1052895
- Hummel B, Juergens E, Heinemann L, Conradt M (2010) Proc. International Conference on Software Maintenance (ICSM'10) . IEEE, 10 pp.
- Jedlitschka A, Juristo N, Rombach D (2013) Reporting experiments to satisfy professionals' information needs. Empirical Software Engineering 19 (6): 1921-1955. DOI: 10.1007/s10664-013-9268-6
- Johnson PM (2007) Requirement and design trade-offs in Hackystat: An in-process software engineering measurement andanalysis system. First International Symposium on Empirical Software Engineering and Measurement (ESEM'07). DOI: 10.1109/ESEM.2007.36
- Juergens E, Deissenboeck F, Hummel B, Wagner S (2009) Do code clones matter? Proc. International Conference on Software Engineering (ICSE'09). DOI: 10.1109/icse.2009.5070547
- Juergens E, Deissenboeck F, Feilkas M, Hummel B, Schaetz B, Wagner S, Domann C, Streit J (2010) Can clone detection support quality assessments of requirements specifications? Proc. 32nd International Conference on Software Engineering. ACM DOI: 10.1145/1810295.1810308
- Kahneman D (2012) Thinking. Fast and Slow. Penguin, 490 pp.
- Kamei Y, Shihab E, Adams B, Hassan A, Mockus A, Sinha A, Ubayashi N (2013) A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering 39 (6): 757-773. DOI: 10.1109/tse.2012.70
- Kan SH, Basili VR, Shapiro LN (1994) Software quality: An overview from the perspective of total quality management. IBM Systems Journal 33 (1): 4-19. DOI: 10.1147/sj.331.0004
- Kawaguchi S, Yamashina T, Uwano H (2009) SHINOBI: A tool for automatic code clone detection in the IDE. Proc. 16th Working Conference on Reverse Engineering (WCRE'09). DOI: 10.1109/wcre.2009.36
- Kersten M, Murphy GC (2006) Using task context to improve programmer productivity. Proc. 14th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-14). DOI: 10.1145/1181775.1181777
- Khamis N, Witte R, Rilling J (2010) Automatic quality assessment of source code comments: the JavadocMiner. Proc. 15th International Conference on Applications of Natural Language to Information Systems. DOI: 10.1007/978-3-642-13881-2_7
- Kim S, Ernst MD (2007) Which warnings should I fix first? Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007). ACM DOI: 10.1145/1287624.1287633

- Kläs M, Heidrich J, Münch J, Trendowicz A (2009) CQML scheme: A classification scheme for comprehensive quality model landscapes. Proc. 35th Euromicro SEAA Conference. DOI: 10.1109/seaa.2009.88
- Korel B, Laski J (1990) Dynamic slicing of computer programs. Journal of Systems and Software 13 (3): 187-195. DOI: 10.1016/0164-1212(90)90094-3
- Li Z, Lu S, Myagmar S, Zhou Y (2006) CP-Miner: finding copy-paste and related bugs in large-scale software code. IEEE Transactions on Software Engineering 32 (3): 176-192. DOI: 10.1109/tse.2006.28
- Lochmann K, Ramadani J, Wagner S (2013) Are comprehensive quality models necessary for evaluating software quality? Proc. 9th International Conference on Predictive Models in Software Engineering (PROMISE 2013). ACM DOI: 10.1145/24993 93.2499404
- Marinescu C, Marinescu R, Mihancea RF, Ratiu D, Wettel R (2005) iPlasma: An integrated platform for quality assessment of object-oriented design. Proc. 21st IEEE International Conference on Software Maintenance (ICSM'05). IEEE DOI: 10.1109/ ICSM.2005.63
- McMinn P (2004) Search-based software test data generation: a survey. Software Testing, Verification and Reliability 14 (2): 105-156. DOI: 10.1002/stvr.294
- Mordal-Manet K, Balmas F, Denier S, Ducasse S, Wertz H, Laval J, Bellingard F, Vaillergues P (2009) The squale model – a practice-based industrial quality model. IEEE International Conference on Software Maintenance (ICSM'09). IEEE DOI: 10.110 9/icsm.2009.5306381
- Murphy GC, Kersten M, Findlater L (2006) How are Java software developers using the Elipse IDE? IEEE Software 23 (4): 76-83. DOI: 10.1109/ms.2006.105
- Murphy GC, Notkin D, Sullivan K (1995) Software reflexion models: bridging the gap between source and high-level models. Proc. 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering. ACM DOI: 10.1145/222124.222136
- Nasehi SM, Sillito J, Maurer F, Burns C (2012) What makes a good code example?: A study of programming. Q&A in StackOverflow. Proc. International Conference on Software Maintenance (ICSM'12). IEEE
- Okumoto K (1985) A Statistical Method for Software Quality Control. IEEE Transactions on Software Engineering 12: 1424-1430. DOI: 10.1109/tse.1985.232178
- Ostberg J-, Ramadani J, Wagner S (2013) A novel approach for discovering barriers in using automatic static analysis. Proc. International Conference on Evaluation and Assessment in Software Engineering (EASE'13). ACM DOI: 10.1145/2460999.2461010
- Pacheco C, Lahiri SK, Ernst MD, Ball T (2007) Feedback-directed random test generation. Proc. 29th International Conference on Software Engineering (ICSE '07). IEEE DOI: 10.1109/ICSE.2007.37
- Parnas DL (1994) Software aging. Proc. International Conference on Software Engineering (ICSE '94). IEEE DOI: 10.1109/ICSE.1994.296790
- Ponzanelli L, Bacchelli A, Lanza M (2013) Seahawk: stack overflow in the IDE. Proc. International Conference on Software Engineering (ICSE'13). IEEE DOI: 10.1109/ icse.2013.6606701
- Pretschner A, Prenninger W, Wagner S, Kühnel C, Baumgartner M, Sostawa B, Zölch R, Stauner T (2005) One evaluation of model-based testing and its automation. Proc. 27th International Confernce on Software Engineering (ICSE'05). ACM

- Salman I, Misirli AT, Juristo N (2015) Are students representatives of professionals in software engineering experiments? Proc. 37th IEEE International Conference on Software Engineering (ICSE). IEEE DOI: 10.1109/icse.2015.82
- Schackmann H, Jansen M, Lichter H (2009) Tool support for user-defined quality assessment models. MetriKon.
- Steidl D, Hummel B, Juergens E (2013) Quality analysis of source code comments. Proc. 21st International Conference on Program Comprehension (ICPC'13). IEEE DOI: 10.1109/icpc.2013.6613836
- Wagner S (2007) An approach to global sensitivity analysis: FAST on COCOMO. Proc. First International Symposium on Empirical Software Engineering and Measurement. DOI: 10.1109/ESEM.2007.44
- Wagner S (2010) A Bayesian network approach to assess and predict software quality using activity-based quality models. Information and Software Technology 52 (11): 1230-1241. DOI: 10.1016/j.infsof.2010.03.016
- Wagner S (2013) Software Product Quality Control. Springer, 210 pp. URL: http://dx.doi.org/10.1007/978-3-642-38571-1 DOI: 10.1007/978-3-642-38571-1
- Wagner S, Jürjens J, Koller C, Trischberger P (2005) Comparing bug finding tools with reviews and tests. Proc. 17th International Conference on Testing of Communicating Systems (TestCom'05). 3502. Springer DOI: 10.1007/11430230_4
- Wagner S, Deissenboeck F, Aichner M, Wimmer J, Schwalb M (2008) An evaluation of two bug pattern tools for Java. Proc. International Conference on Software Testing, Verification and Validation (ICST'08). IEEE DOI: 10.1109/icst.2008.63
- Wagner S, Lochmann K, Winter S, Goeb A, Klaes M (2009) Quality models in practice: A preliminary analysis. Proc. 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09). IEEE DOI: 10.1109/esem.2009.5316003
- Wagner S, Lochmann K, Winter S, Goeb A, Kläs M, Nunnenmacher S (2012a) Software quality models in practice. Technical Report TUM-I129 . Technische Universität München
- Wagner S, Lochmann K, Heinemann L, Kläs M, Trendowicz A, Plösch R, Seidl A, Goeb A, Streit J (2012b) The Quamoco product quality modelling and assessment approach. Proc. 34th International Conference on Software Engineering (ICSE'12). DOI: 10.1109/icse.2012.6227106
- Wagner S, Goeb A, Heinemann L, Kläs M, Lampasona C, Lochmann K, Mayr A, Plösch R, Seidl A, Streit J, Trendowicz A (2015) Operationalised product quality models and assessment: The Quamoco approach. Information and Software Technology 62: 101-123. DOI: 10.1016/j.infsof.2015.02.009
- Weller EP (2000) Practical applications of statistical process control [in software development projects]. IEEE Software 17 (3): 48-55. DOI: 10.1109/52.896249
- Wilson TD (1994) Information needs and uses: fifty years of progress. In: Vicerky BC (Ed.) Fifty Years of Information Progress: A Journal of Documentation Review. Aslib
- Ying AT, Murphy GC, Ng R, Chu-Carroll MC (2004) Predicting source code changes by mining change history. IEEE Transactions on Software Engineering 30 (9): 574-586. DOI: 10.1109/tse.2004.52
- Ying ATT, Robillard MP (2011) The influence of the task on programmer behaviour. Proc. 19th International Conference on Program Comprehension (ICPC'11). DOI: 10.1109/icpc.2011.35

- Zheng J, Williams L, Nagappan N, Snipes W, Hudepohl JP, Vouk MA (2006) On the value of static analysis for fault detection in software. IEEE Transactions on Software Engineering 32 (4): 240-253. DOI: 10.1109/tse.2006.38
- Zimmermann T, Zeller A, Weissgerber P, Diehl S (2005) Mining version histories to guide software changes. IEEE Transactions on Software Engineering 31 (6): 429-445. DOI: 10.1109/tse.2005.72