TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

Abschlussbericht - Q-Effekt: Qualitätssicherungsmaßnahmen effektiv steuern

Sebastian Eder, Benjamin Hummel

TUM-I1871

Schlussbericht

Q-Effekt: Qualitätssicherungsmaßnahmen effektiv steuern

13. November 2018

Dr. Martin Feilkas
Dr. Benjamin Hummel
Rainer Niedermayr

CQSE GmbH
Lichtenbergstr. 8
85748 Garching

Prof. Dr. Dr. h.c. Manfred Broy
Dr. Sebastian Eder
Dr. Benedikt Hauptmann

Lehrstuhl für Software & Systems Engineering
Technische Universität München
Boltzmannstr. 3
85748 Garching

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS15003 (A-B) gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

1 Kurzdarstellung

Software ist heute für nahezu alle Unternehmen ein zentraler Bestandteil ihres Geschäfts, sowohl bei der Unterstützung und Umsetzung von Geschäftsprozessen (z.B. bei Versicherungen und Banken oder zur Realisierung der Wertschöpfungskette in der Industrie) als auch als Teil der entwickelten und vertriebenen Produkte (z.B. eingebettete Software in Automobilen, Flugzeugen etc.). Softwaresysteme, insbesondere die erfolgreichen, werden oftmals über viele Jahrzehnte hinweg gepflegt, um neue Funktionalität erweitert, an Änderungen in den Geschäftsprozessen oder der Rechtslage angepasst, auf neue Plattformen portiert oder mit anderen Systemen integriert.

Der Qualitätssicherung kommt in Softwareentwicklungsprojekten eine besondere Bedeutung zu. Sie dient einerseits dazu, die funktionale Korrektheit von Systemen zu gewährleisten, um Fehlverhalten, Datenverluste, Systemausfälle etc. zu verhindern. Für betriebliche Informationssysteme kann dies erhebliche wirtschaftliche Schäden bedeuten, im Bereich eingebetteter Software kann dies sogar sicherheitskritische Auswirkungen haben. Darüber hinaus ist auch die Wahrung der langfristigen Verständlichkeit und Wartbarkeit eine wichtige Aufgabe der Qualitätssicherung, um dem schleichenden Qualitätsverfall [Par94] und den damit einhergehenden Wartungsaufwänden sowie dem Risiko eines vorzeitigen Totalverlusts der Investitionen in ein Softwaresystem zu begegnen.

Zur Qualitätssicherung werden heute im Wesentlichen drei Kategorien von Methoden eingesetzt (vgl. Abbildung 1):

1. *Testen:* Tests dienen im Wesentlichen der Sicherstellung der Korrektheit. Dabei kommen sowohl manuelle Tests wie auch automatisierte Tests auf unterschiedlichen Teststufen zum Einsatz.
2. *Reviews:* Hierbei werden Entwicklungsartefakte, wie z.B. Anforderungen, Architekturentwürfe oder Quellcode manuell inspiziert, um Defizite aufzudecken.
3. *Statische Analysen:* Diese Analyseverfahren identifizieren sowohl funktionale Defekte als auch Wartbarkeitsdefizite. Gerade in älteren, aber auch in großen Systemen, finden sich oftmals sehr viele Qualitätsdefizite – oft mehrere Zehntausend. Derartige Verfahren werden meist auf Quellcode eingesetzt, es sind jedoch bereits einige vorhanden, die auch auf anderen Entwicklungsartefakten (Modelle, natürlich-sprachliche Dokumente) genutzt werden können.

In der Praxis stellt sich insbesondere im Hinblick auf beschränkte Ressourcen für die Qualitätssicherung die Auswahl und die Priorisierung der im Projektkontext relevanten Verfahren als großes Problem dar. Neben der Priorisierung der Verfahren selbst, also der Auswahl, ob mehr Aufwand in Tests, Reviews oder statische Analysen investiert werden soll, ist auch die Fokussierung der Aufwände innerhalb einer Methode schwer zu beantworten. Bei Anwendung der Verfahren stellen sich bzgl. einer möglichst optimalen Investition der aufgrund von Zeit- und Kostendruck beschränkten Aufwände folgende Fragen:

1. *Testen:* Welche manuellen Testfälle sollen ausgeführt werden? Welche automatisierten Tests sollen implementiert werden? Welche Testfälle sollen mit dem System gewartet werden und welche Testfälle sind obsolet?

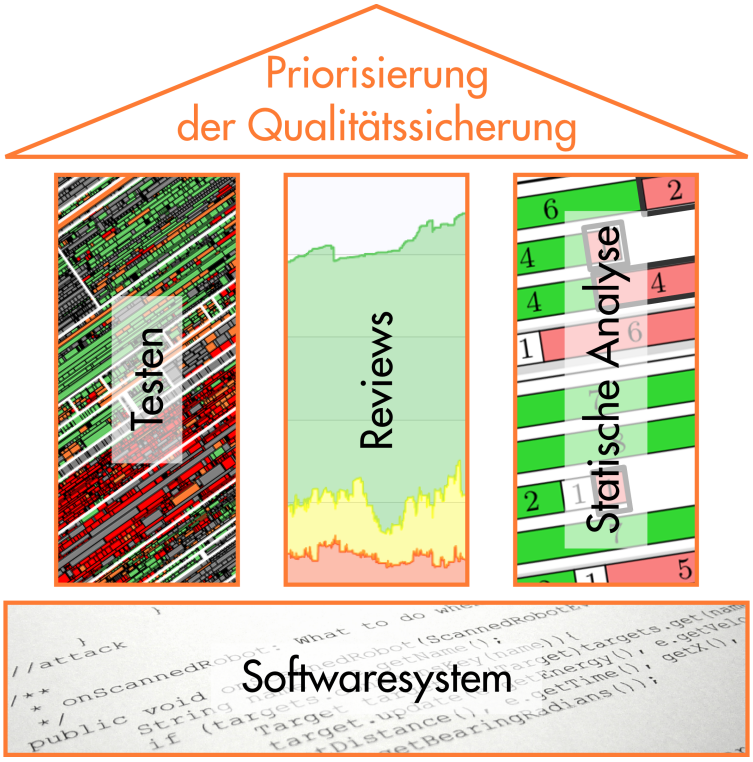

Abbildung 1: Die drei Säulen der Qualitätssicherung

2. *Reviews:* Welche Entwicklungsartefakte bzw. welche Teile davon sollen inspiziert werden?
3. *Statische Analysen:* Welche der oftmals tausenden von den Analysewerkzeugen gefundenen Auffälligkeiten sollen behoben werden?

Diese Fragen zielen auf eine Fokussierung auf die mit einer Methode zu behandelnden Entwicklungsartefakte. Von der Beantwortung dieser Fragen hängt für ein Entwicklungsprojekt die Effektivität der Qualitätssicherungsmaßnahmen in hohem Maße ab.

Im Rahmen des Forschungsprojekts Q-Effekt wurden in Zusammenarbeit der CQSE GmbH und der TU München Verfahren konzipiert, die es Projektleitern und Qualitätssicherungsverantwortlichen ermöglichen, diese komplexen Fragestellungen fundierter und nachvollziehbarer zu beantworten. Dies wird erreicht, indem ihnen strukturiert Kontextinformationen für ihre Entscheidungen zur Verfügung gestellt werden, die wesentlichen Einfluss auf die Effektivität der einzelnen Maßnahmen haben. Diese Kontextinformationen sind heutzutage in Entwicklungsprojekten meist nicht verfügbar oder liegen nur implizit vor. Derartige für die Priorisierung der Qualitätssicherungsmaßnahmen oftmals entscheidende Informationen sind beispielsweise:

1. *Testen* Gibt es Codeteile, die geändert und noch nicht getestet wurden? Welche Tests testen kritische Anforderungen? Welche Codeteile werden von welchen Tests durchlaufen? Gibt es besonders komplexe Komponenten, die zentral im System genutzt werden

und noch unzureichende Testüberdeckung aufweisen? Gibt es Tests, die aufgrund von niedriger Qualität nicht zufriedenstellend ausgeführt werden können?

2. *Reviews* Welche Entwicklungsartefakte wurden seit dem letzten Review stark geändert? Welche Anforderungen wurden schon früher nicht richtig umgesetzt? In welchen Artefakten gibt es Probleme bei der Verständlichkeit? Welche Codeteile hatten in der Vergangenheit die höchste Fehlerdichte?
3. *Statische Analysen* Welche Wartbarkeitsdefizite liegen in Codebereichen, die sich in der Vergangenheit häufig änderten? Welche Defizite sind in generiertem Code und sind deshalb von nachrangiger Bedeutung? Werden funktionale Defekte in nicht von automatischen Tests durchlaufenen Bereichen gefunden?

1.1 Voraussetzungen des Vorhabens

Fast jede Qualitätssicherungsmaßnahe kann helfen, den Qualitätsverfall aufzuhalten oder sogar umzukehren. Allerdings sind so viele mögliche Verfahren und Werkzeuge verfügbar, dass es nicht möglich ist, alle in vollen Tiefe über ein komplettes Softwareprojekt anzuwenden.

1. *Testen* Testsuiten von großen Projekten haben heutzutage Umfänge erreicht, dass deren vollständige manuelle Ausführung mehrere Monate bis Jahre in Anspruch nehmen würde. Bei immer kürzeren Release-Zyklen und auch bei der Erstellung von Hot-Fixes ist eine komplette Ausführung aller Tests meist nicht mehr ökonomisch sinnvoll umsetzbar.
2. *Reviews* Ein vollständiges Review einer kompletten Codebasis kann viele Personenjahre in Anspruch nehmen. Eine Fokussierung der Reviews auf Änderungen ist nur möglich, wenn diese Änderungen und auch der Reviewstatus klar nachverfolgbar sind. Zudem besteht beim Review die Gefahr, dass der Reviewer von relevanteren Problemen abgelenkt wird, wenn zu viele andere Probleme im Code sind. Daher ist eine Verzahnung mit automatischer statischer Analyse essentiell.
3. *Statische Analysen* Heutige Werkzeuge finden auf typischen gewachsenen Codebasen schnell Millionen von Qualitätsdefiziten. Die Behebung und oft schon die Sichtung all dieser Probleme ist oft nicht zu bewältigen. Entsprechend ist eine Priorisierung und Selektion dieser Ergebnisse entscheidend.

Alle diese Probleme führen zu hohen und wenig zielgerichteten Aufwänden bei der praktischen Umsetzung von Qualitätssicherungsmaßnahmen, die im schlechtesten Fall dazu führen, dass die Qualitätssicherung trotz hohem Ressourceneinsatz nicht das gewünschte Ergebnis bringt.

1.2 Aufgabenstellung

Ziel dieses Projekts war es, durch Kombination von Kontextinformationen eine fundiertere und nachvollziehbarere Entscheidung zur Allokation von Qualitätssicherungsaufwänden in

Softwareentwicklungsprojekten zu ermöglichen. Die zentralen Herausforderungen zur Nutzbarmachung derartiger Verfahren, die im Rahmen dieses Forschungsprojekts konzipiert und prototypisch realisiert wurden, sind die Gewinnung der relevanten Kontextinformationen sowie deren geeignete Kombination zur Erlangung möglichst aussagekräftiger Resultate.

Gerade für große Systeme und dynamische Entwicklungsprojekte tendieren viele der relevanten Kontextinformationen dazu, sehr große Volumina anzunehmen. Bei Berücksichtigung aller Entwicklungsartefakte, sowie der kompletten Entwicklungshistorie fallen mehrere Millionen Datensätze an. Üblich sind hier mehrere Millionen Zeilen Programmcode und mehrere tausend andere Artefakte (Tests, Reviewprotokolle, Anwendungsfälle, etc.), die durch mehrjährige Entwicklung leicht in hundert verschiedenen Versionen vorliegen. Darüber hinaus sind die betrachteten Daten sehr heterogen, da Programmcode, natürlichsprachliche Dokumente und auch Modelle untersucht werden. Um technisch effizient mit dem Umfang der Daten umgehen zu können, stellen moderne Techniken aus den Bereichen „Machine Learning“ und „Text Mining“ eine interessante technische Basis dar. Für eine Umsetzung müssen neue Analysetechniken entwickelt und bestehende angepasst werden.

1.3 Planung und Ablauf des Vorhabens

Die Bearbeitung des Projekts wurde in 7 Arbeitspakete unterteilt. Diese Arbeitspakete und der geplante zeitliche Ablauf ist in Abbildung 2 als Gantt-Diagramm dargestellt. Die genauen Inhalte der Arbeitspakete finden sich bei den Ergebnissen in Abschnitt 2.

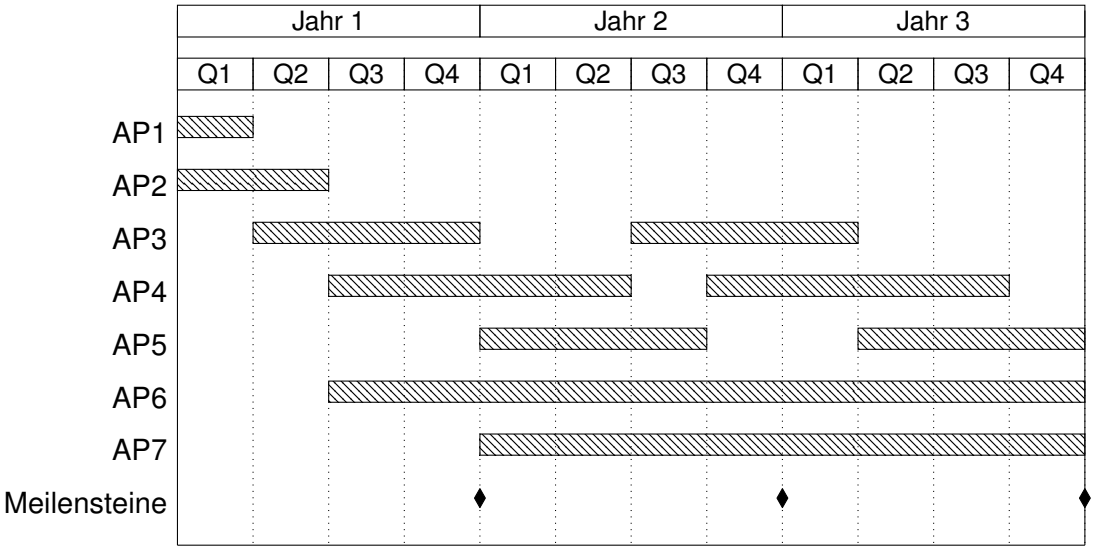

Abbildung 2: Arbeitsplan

1.4 Wissenschaftlicher und technischer Stand

Wissenschaftlicher Stand Die meisten wissenschaftlichen Arbeiten aus dem Bereich der Qualitätssicherung beschäftigen sich mit dem Auffinden von Qualitätsdefiziten [Kos07, RAF04], jedoch nicht mit der Priorisierung von Qualitätsdefiziten und der Steuerung von Qualitätsverbesserungsmaßnahmen. Ebenso beschäftigt sich die Literatur seit dem ersten Artikel von Fagan [Fag76] zwar intensiv mit Verfahren zur Durchführung von Reviews und deren Effektivität und Nutzen [SV01], jedoch kaum mit der optimalen Allokation von Review-Aufwänden [KK09].

Ein Beispiel für ein Verfahren, bei dem Testaufwände priorisiert werden, ist der selektive Regressionstest [GHK+01, RH96]. Hierfür wird durch einen Algorithmus eine minimale und somit schneller durchführbare Menge an Tests ausgewählt, um vorgenommene Systemänderungen abzusichern. Während das Verfahren bei automatisierten Unit-Tests sehr gut funktioniert, zeigt eine Studie der CQSE [JHD+11], dass sich das Verfahren auf manuelle Systemtests nicht direkt übertragen lässt. Ein Ansatz zur Optimierung von Qualitätssicherungsaufwänden wird in [Wag07] vorgestellt. Dieser verwendet ein stochastisches Modell zur Planung der Qualitätssicherung, berücksichtigt aber nur sehr abstrakte Faktoren und lässt die abzusichernden Artefakte außen vor.

Im Bereich der Priorisierung der Ergebnisse von statischen Analysen existieren Arbeiten für die Priorisierung von Code-Clones [VGS13], die sich jedoch nicht einfach auf allgemeine Qualitätsdefizite verallgemeinern lassen. Boogerd und Moonen [BM06] schlagen ein Priorisierungsverfahren basierend auf der Ausführungswahrscheinlichkeit von Quelltextdateien vor. Diese sehr spezielle Sicht lässt jedoch wichtige Parameter, wie etwa die erwarteten Änderungen oder die Wichtigkeit des Codes sowie die Art des Qualitätsdefizits außen vor.

Die genannten Arbeiten zeigen, dass ausgewählte Aspekte der Priorisierung von Qualitätssicherungsmaßnahmen Gegenstand der Forschung sind und waren. Eine systematische Abwägung unter Einbeziehung der Abhängigkeiten der abzusichernden Artefakte und des Entwicklungskontextes, wie sie durch dieses Vorhaben angestrebt wird, wurde jedoch bisher nicht verfolgt.

Im Projekt Quamoco, an dem federführend die TU München beteiligt war, wurden bereits wichtige Grundlagen erarbeitet, die auch im Rahmen dieses Projekts zur Anwendung kamen. Die in Quamoco entwickelte Methodik zur Beschreibung von Qualitätsmodellen stellt eine Basistechnologie zur Definition relevanter Qualitätskriterien dar. Dies stellt einen definitorischen Ansatz zur Qualitätssicherung dar. Für die Bereiche Reviews und statische Analysen stellt Quamoco daher eine Grundlage dar. Aussagen zur Priorisierung von Aufwänden, wie sie in diesem Projekt erarbeitet wurden, wurden in Quamoco nicht erarbeitet.

Das Projekt EvoCon, das von den beiden Hauptantragstellern bearbeitet wurde, entwickelte Analyseverfahren, die auch den zeitlichen Verlauf und insbesondere die Evolution von Softwaresystemen berücksichtigen. Schwerpunkt war die Entwicklung inkrementeller Analyseverfahren, die zum einen eine genau zeitliche Nachverfolgung von Qualitätsproblemen und zum anderen eine Berechnung und Speicherung der kompletten Qualitätshistorie erlauben. Die Priorisierung von Qualitätssicherungsmaßnahmen wurde nur für statische Analysen und nur am Rande betrachtet, die Priorisierung von Test- und Reviewaufwänden war kein Projektinhalt. Für den Bereich der statischen Analyse liefern die Arbeiten aus EvoCon

eine wertvolle Grundlage, da die dort anfallenden Daten zur Priorisierung von Qualitätssicherungsmaßnahmen herangezogen werden können.

Weitere nationale und internationale Forschungsprojekte mit vergleichbaren Zielsetzungen sind nicht bekannt.

Technischer Stand Aktuell fokussieren sich Qualitätssicherungswerkzeuge lediglich auf die Erkennung von Qualitätsdefiziten oder die Messung entsprechender Metriken, die Definition und Ausführung von Testfällen sowie der Unterstützung der Durchführung von Reviews.

Die Firma CAST bietet mit ihrer Application Intelligence Platform ein Werkzeug zur Metrikberechnung und Suche von Fehlermustern. Diese Daten werden dann über eine zentrale Datenbank zusammengeführt und verfügbar gemacht. Coverity bietet eine ganze Palette von Werkzeugen für die dynamische und statische Analyse von Software an. In Deutschland bietet die Axivion GmbH das Werkzeug Bauhaus Suite an, das die Berechnung verschiedener Metriken und die Fehlermustererkennung unterstützt und diese in einem konfigurierbaren Leitstand zusammenführt.

Das Open-Source-Werkzeug SonarQube integriert eine eigene Metrikberechnung mit etablierten Werkzeugen für die Suche nach Fehlermustern für Java (PMD, Findbugs). Die Ergebnisse werden in einer Datenbank erfasst und in einem konfigurierbaren Leitstand dargestellt. Eine kommerzielle Version von SonarQube, die neben Java auch weitere Programmiersprachen unterstützt, wird von der Schweizer Firma SonarSource entwickelt.

ConQAT ist ein Open-Source-Werkzeug für die kontinuierliche Qualitätsanalyse von Software-Systemen und wurde ursprünglich an der Technischen Universität München entwickelt. ConQAT verfolgt einen konsequenten Baukastenansatz, der eine einfache Erweiterung um zusätzliche Analysen unterstützt. Neben der Berechnung verschiedener Qualitätsmetriken und Fehlermuster enthält ConQAT auch eine umfassende Visualisierungskomponente mit der sich komplexe Leitstände für das Qualitäts-Controlling aufbauen lassen. Teamscale ist ein auf ConQAT aufbauendes Werkzeug, das die inkrementelle Historienanalyse, wie sie im Projekt EvoCon erarbeitet wurde, umsetzt. Wegen ihrer Erweiterbarkeit sollen die Werkzeuge ConQAT und Teamscale auch als Basis für das beantragte Projekt Q-Effekt dienen, um die grundlegenden statischen Analysen nicht vollständig neu entwickeln zu müssen.

Im Bereich Reviews sind Werkzeuge zur Erfassung und Koordination von Reviews verbreitet, wie etwa die beiden Open Source Werkzeuge ReviewBoard und Gerrit, oder das von Atlassian verbreitete kommerzielle Werkzeug Crucible. Keines dieser Werkzeuge erlaubt eine systematische Planung und Priorisierung von Reviewaktivitäten durch Kontextinformationen.

Das Thema Testen und die Planung von Tests wird weit verbreitet mit den kommerziellen Werkzeugen HP Quality Center, Tosca oder Microsoft Team Foundation Server behandelt. Diese Werkzuege stützen sich ausschließlich auf Expertenwissen zur Planung und Priorisierung von Testaufwänden. Die objektive Berücksichtigung von Informationen aus dem Projektkontext erfolgt nur implizit durch den Testmanager.

Übergeordnete Fragestellungen der Steuerung von Qualitätssicherungsaktivitäten, zur Klärung welche Aktivitäten für welche Teile eines Systems sinnvollerweise zum Einsatz kommen sollten, werden durch diese Werkzeuge nicht adressiert. Die Nutzung von Kontextinformation zur Steuerung von Qualitätssicherungsaktivitäten ist damit ein vollständig neuartiger Ansatz für ein aktuell nur unzureichend adressiertes Problem. Die in Abschnitt 1 formulierten Fragen, die sich in jedem Entwicklungsprojekt stellen, werden derzeit noch nicht durch existierende Werkzeuge beantwortet. Auch in der gängigen Fachliteratur werden diese in der Praxis höchst relevanten Probleme methodisch kaum adressiert.

1.5 Zusammenarbeit mit anderen Stellen

Das Vorhaben wurde im Verbund von CQSE GmbH und Technischer Universität München bearbeitet. Das Projekt wurde unterstützt durch die Lebensversicherung von 1871 und die Stadtwerke München, die als assoziierte Partner an dem Projekt teilgenommen und für die Evaluierung der Verfahren einen wesentlichen Beitrag geleistet haben. Zudem konnte im Laufe des Projekts die BMW AG als Fallstudiengeber für einzelne Fragestellungen gewonnen werden.

2 Ergebnisse des Projekts

Die Ergebnisse wurden in 7 Arbeitspaketen erarbeitet, die nachfolgend genauer beschrieben sind. Die Arbeitspakete wurden jeweils von beiden Projektpartnern bearbeitet, jedoch liegt die Verantwortung für ein Arbeitspaket jeweils bei einem der Partner.

2.1 AP1: Relevante Kontextinformationen

Im Rahmen dieses Arbeitspakets wurde zusammengestellt, welche im Software Engineering erstellte Artefakte, wie Anforderungsdokumente, Modelle, Code, Tests etc., und welche Kontextinformation relevanten Einfluss auf die Auswahl von Qualitätssicherungsmaßnahmen haben. Untersuchungen bei den Partnern haben gezeigt, dass das Spektrum hier sehr vielfältig ist und sich stark zwischen den Partnern, aber auch zwischen unterschiedlichen Projekten unterscheidet. Die wesentliche Erkenntnis aus diesem Arbeitspaket ist daher, dass ein adaptiver Ansatz notwendig ist, der nicht eine feste Menge von Artefakten oder Kontextinformationen voraussetzt, sondern schrittweise um Informationen angereichert werden kann, sofern oder sobald diese vorhanden sind. Die konkret genutzten Artefakte finden sich in den Beschreibungen der folgenden Arbeitspakete.

2.2 AP2: Auffinden von Kontextinformation und Artefaktbeziehungen

Im Rahmen dieses Arbeitspakets wurden die grundlegenden Techniken und Methoden entwickelt, die es ermöglichen Kontextinformationen und Artefaktbeziehungen (als spezielle Art

von Kontextinformation) zu berechnen bzw. wiederherzustellen, auch wenn diese ggf. nicht unmittelbar explizit in den Artefakten vorhanden sind.

In diesem Rahmen wurden zum einen Techniken erwickelt, um Wissen über die Art des Codes (generierter Code, Test-Code) strukturiert erheben und dokumentieren lassen. Dies basiert zum einen auf Pattern auf den Pfaden der Code-Dateien, aber auch auf Mustern die innerhalb der Dateien gefunden werden können. Zudem wurden Architekturmodelle entwickelt, die sich in Echtzeit mit der Implementierung im Quellcode vergleichen lassen. Somit kann der Architekt in einem Projekt stets überprüfen, ob sein Kontextwissen zur Architektur noch mit dem aktuellen Projektstand übereinstimmt. Hierfür wurden Verfahren entwickelt, die eine Extraktion von Abhängigkeiten sowohl in typisierten als auch heuristisch in untypisierten Programmiersprachen erlauben.

Als spezielle Art der Artefaktbeziehung wurde die Beziehung zwischen Modell und (generiertem) Code in der Modellbasierten Entwicklung betrachtet. Hierfür konnte als Fallstudiengeber die BMW AG gewonnen werden, bei der C-Code aus Matlab/Simulink-Modellen generiert wird. Herausforderung dabei ist, dass viele Analyseverfahren (statische Analyse, Test-Coverage, etc.) auf dem generierten Code ausgeführt werden, der Entwickler aber üblicherweise nur das Modell kennt und bearbeitet, während der generierte Code für ihn nicht verständlich ist. Im Rahmen des Arbeitspakets wurde ein Verfahren entwickelt, mit dem sich ein Mapping zwischen Code und Modell aufbauen lässt. Mit diesem Mapping können dann Analyseergebnisse vom Code auf das Modell übertragen und dem Entwickler zugänglich gemacht werden.

Weiterhin wurden zwei Techniken entwickelt, die zwischen Artefakten, die in natürlicher Sprache verfasst sind, Ähnlichkeiten detektieren. Dazu wird semantische Ähnlichkeit durch syntaktische Ähnlichkeit angenähert, indem die Worte, die in Artefakten enthalten sind, ausgewertet und verglichen werden. Mittels der Techniken konnten erfolgreich Beziehungen zwischen Anforderungs- und Testartefakten, sowie Fehlerbeschreibungen hergestellt werden. Auch konnten Artefaktbeziehungen zu Quellcode nachvollzogen werden.

2.3 AP3: Methoden zur Priorisierung von Review-Aktivitäten

In diesem Arbeitspaket wurde die Anwendung der in AP2 entwickelten Techniken zur Priorisierung und Planung von Review-Aktivitäten betrachtet. Dabei werden sowohl dokumentenzentrierte Reviews (Anforderungen, natürlichsprachliche Testfälle), modellzentrierte Reviews (z.B. Architekturmodelle) und Reviews des Quellcodes berücksichtigt.

Als Basis wurden zunächst Konzepte erarbeitet, wie sich geplante und erfolge Reviews dokumentieren und nachverfolgen lassen. Dies ist die Voraussetzung für eine bessere Priorisierung dieser Aktivitäten. In weiteren Verlauf wurde bei den Partnern beobachtet, dass Reviews zunehmend in dedizierten Werkzeugen (Crucible, Gerrit) durchgeführt werden oder die Reviews stärker im Versionskontrollsystem auf Basis von Feature-Branches erfolgen (verbreitet v.a. bei GitHub, Gitlab).

Als Konsequenz wurde in diesem Arbeitspaket stark daran gearbeitet, wie sich Informationen aus diesen Reviewwerkzeugen für die Entwickler aufbereiten lassen. Dies wurde exemplarisch für Crucible umgesetzt, wo die Entwickler nun mittels verschiedener erhobener

Metriken den Fortschritt der Reviews und Review-Lücken transparent gespiegelt bekommen. Ergänzend wurde untersucht, wie sich Daten über den Qualitätsstand des Systems dem Nutzer direkt beim Review bereitgestellt werden können. Dies wurde speziell für das populäre frei verfügbare Review-Werkzeug Gerrit untersucht.

Zur Priorisierung von Review-Aktivitäten auf natürlichsprachlichen Dokumenten, wurden zunächst Indikatoren eroben, welche Qualitätsdefekte in Anforderungs- und Testartefakten anziegen. Darauf aufbauend konnten automatische Techniken zur Detektion dieser Qualitätsindikatoren entwickelt werden. Mit diesen Techniken können automatisch detektierbare Qualitätsdefekte vom Autor behoben werden, sodass diese Defekte nicht mehr in einem Review beachtet werden müssen.

2.4 AP4: Methoden zur Priorisierung der durch statische Analysen identifizierten Qualitätsdefizite

Das Arbeitspaket 4 befasste sich mit der Priorisierung von Qualitätsdefiziten aus statischen Analysen. Ein Schwerpunkt des Arbeitspakets war eine konsistente Verwaltung und Behandlung von solchen Defiziten sowohl auf Code und auf Modellen. Für die Modelle wurde exemplarisch die im Automotive-Bereich weit eingesetzte Modellierungssprache Matlab/Simulink gewählt. Hierfür wurde an Konzepten gearbeitet, wie sich Analyseergebnisse aus generiertem Code auf die Ausgangsmodelle für die Codeerzeugung übertragen lassen. Die Abbildung auf das Modell, also die Darstellung, die den Entwicklern besser vertraut ist, erlaubt eine bessere Abschätzung der Relevanz von Qualitätsdefiziten und ermöglicht damit eine bessere Priorisierung. Die Konzepte wurden exemplarisch für Matlab/Simulink und die dort verfügbaren Codegeneratoren realisiert und evaluiert.

Nach Feedback der assoziierten Partner wurde ein weiterer Schwerpunkt auf den Umgang mit Qualitätsdefiziten im Rahmen von Branching/Merging gelegt, da Branching/Merging für moderne Entwicklungsprozesse essentiell ist, gleichzeitig aber von bestehenden Ansätzen für das Qualitätscontrolling nicht oder nur unzureichend unterstützt wird. Die Herausforderung dabei ist der Umgang mit einer großen Anzahl von Branches, wie sie beim Einsatz des populären Versionskontrollsystems Git oft entstehen. Wichtig ist hier die Nachverfolgung von Qualitätsdefiziten über Branch- und Merge-Operationen hinweg. Für die Priorisierung muss nämlich unterschieden werden zwischen Qualitätsdefiziten aus dem Hauptzweig, solchen die durch den Merge aus dem Feature-Branch übernommen wurden und solchen die durch den Merge erst neu entstanden sind. Ebenso müssen Defizite die als irrelevant oder falsch auf einem Branch markiert wurden, analog auf anderen Branches markiert werden. Hierzu wurde untersucht, wie ein entsprechender Blacklisting-Mechanismus unter diesen Rahmenbedingungen aussieht.

Darüber hinaus wurden Daten über die tatsächliche Nutzung von Softwaresystemen herangezogen, um Qualitätsdefekte zu priorisieren. Mit diesen Daten, die eher dem Feld der dynamischen Analyse zuzuordnen sind, können Entwickler entscheiden, ob ein von Qualitätsdefekten betroffener Teil der Quellcodes überhaupt noch angepasst werden sollte, oder entfernt werden sollte, da er nicht genutzt wird.

Zusammen erlauben diese Verfahren dem Entwickler eine deutlich besser informierte Entscheidung über die tatsächliche Priorität von Qualitätsdefiziten aus der statischen Analyse.

2.5 AP5: Methoden zur Priorisierung von Testaktivitäten

In Arbeitspaket 5 wurde an Verfahren gearbeitet, die einen Abgleich von durchgeführten (Regressions-)Tests mit den Änderungen am Quellcode ermöglichen. Dadurch lassen sich dann perspektivisch noch durchzuführende Tests identifizieren, bzw. auswählen welche verbleibenden Tests noch priorisiert werden müssten, um Testlücken zu schließen.

Hierfür wurden zum einen Konzepte für die zuverlässige Erkennung von relevanten Änderungen (ohne typische einfache Refactorings) entwickelt. Diese Änderungen werden dann mit Testergebnissen auf mehreren Teststufen korreliert.

Außerdem wurde die Fehleranfälligkeit von Methoden in Bezug auf deren Quellcode untersucht. Dazu wurden mittels maschinellem Lernen Kombinationen von Methodenattributen zur entsprechenden Klassifizierung von Methoden berechnet. Die Klassifizierung ermöglicht es, „triviale“ Methoden, die so gut wie nie Fehler enthalten, in der Menge der zu testenden Änderungen herunterzupriorisieren. Unsere Untersuchung hat ergeben, dass somit abhängig vom Studienobjekt etwa 32-44% der Methoden mit niedrigerer Priorität getestet werden können.

Ergänzend wurde ein Mutation-Testing-Ansatz entwickelt, um für eine Methode zu bestimmen, von welchen Testfällen sie nur „scheingetestet“ ist. Dies trifft zu, wenn eine Methode zwar von einem Test durchlaufen wird, dieser aber nicht fehlschlägt, selbst wenn die gesamte Logik der Methode entfernt wird. Wenn das Entfernen der gesamten Logik einer Methode nicht erkannt wird, ist davon auszugehen, dass ein Testfall subtilere Fehler in einer Methode erst recht nicht erkennen kann. Durch den Einbezug dieser Informationen in die Testpriorisierung können für den zu testenden Code ineffektive Testfälle zurückgestellt oder ausgeschlossen werden.

Wie in AP4 spielt auch hier das Thema Branching/Merging eine zentrale Rolle. Für die entwickelten Verfahren wurden Konzepte entwickelt, wie diese hierfür angepasst werden müssen, um die Entwicklungs- und Testabteilungen auch in einem modernen Feature-Branch-basierten Entwicklungsprozess optimal zu unterstützen.

Wichtig war dabei auch, die entwickelten Verfahren so zu verbessern, dass sie mit den umfangreichen Datenmengen, wie sie in den Projekten der assoziierten Partner anfallen, überhaupt umgehen können. Dazu wurde beispielsweise untersucht, wie gut die Ergebnisse von Mutationstests mit weniger aufwändigen Verfahren und Metriken (wie zum Beispiel der minimalen Aufrufdistanz zwischen Methoden und Testfällen) angenähert werden können.

Weiterhin wurden Techniken zur Qualitätssicherung von automatisierten Systemtests entwickelt. Mit diesen Techniken können Wartungsaktivitäten auf diesen Tests gesteuert werden.

Außerdem wurden Abdeckungsinformationen von Systemtests benutzt, um ungetestete Änderungen im Quellcode zu identifizieren. Damit können gezielt Testfälle ausgewählt und ausgeführt werden, um neuerliche Änderungen am Quellcode zu testen.

2.6 AP6: Prototypische Umsetzung

Im Rahmen dieses Arbeitspakets wurden die entwickelten Konzepte soweit umgesetzt, dass eine Erprobung in Testumgebungen und bei den assoziierten Partnern möglich wurde. Diese Prototypen sind die Voraussetzung für die Evaluierung und erlaubten eine schnelle Verbesserung der entwickelten Verfahren durch kurzfristiges Feedback aus dem Praxiseinsatz. Hierzu wurden zum einen die Algorithmen und Verfahren der anderen Arbeitspakete umgesetzt werden, um eine Untersuchung der Laufzeiteigenschaften zu erlauben, insbesondere zur Beantwortung der Frage, ob die Analysen schnell genug sind, um die anfallenden Datenmenge zeitnah verarbeiten zu können. Ergänzend waren auch rein technische Probleme zu lösen, wie etwa die Anbindung an bestehende Versionsverwaltungssysteme, um Code-Änderungen auslesen zu können und somit den Prototypen unter realistischen Bedingungen testen zu können.

Die Prototypen bauen auf den vorhandenen Werkzeugen ConQAT und Teamscale auf. Viele Analyseverfahren mussten jedoch komplett neu umgesetzt werden. Beispiele hierfür sind die Analyse der natürlichsprachigen Dokumente, die Trace-Link-Recovery, sowie die Änderungsverfolgung auf Modellen und Texten. Zudem waren Arbeiten notwendig, um die benötigten Informationen aus den verschiedensten Dateiformaten (Word und PDF für Textdokumente, Architekturmodelle, Simulink-Modelle, Testbeschreibungsformate, etc.) zu extrahieren, da andernfalls ein Erprobungseinsatz auf realistischen Projekten bei den Evaluierungspartnern nicht möglich gewesen wäre. Zudem liegen die entsprechenden Dokumente meist in spezialisierten Ablagesystemen (HP Quality Center, Team Foundation Server, SharePoint, Wiki-Systeme etc.). Für einen automatisierten Zugriff auf diese Daten müssen entsprechende Verfahren zumindest rudimentär unterstützt werden.

Darüber hinaus mussten diverse bestehende Technologien für die Analysen natürlichsprachlicher Dokumente erprobt werden, um Laufzeit, Exaktheit und Korrektheit dieser Technologien zu bewerten. Besonders für einen praxistauglichen Einsatz spielen diese Faktoren eine zentrale Rolle. So wurden diese Technologien prototypisch in die oben genannten Werkzeuge integriert.

2.7 AP7: Evaluierung

Die entwickelten Verfahren wurden auf Basis des in AP6 entwickelten Prototypen im Rahmen dieses Arbeitspakets evaluiert. Der erste Schritt für die Evaluierung war der Einsatz des Prototypen in der internen Entwicklung bei der CQSE, da hier am schnellsten Erfahrungen gesammelt werden und Ansätze überarbeitet werden konnten. Zudem wurden stabile Versionen der Prototypen über einen längeren Zeitraum bei den assoziierten Partnern LV1871 und SWM im Rahmen von deren Entwicklungsprojekten über einen Zeitraum von mehreren Monaten erprobt. Ein wesentlicher Schwerpunkt der Evaluierung in allen drei Umgebungen war der Umgang mit Branching/Merging und dem Tracking von Qualitätsdefiziten in diesem Umfeld, da dies die Basis für die anderen entwickelten Verfahren darstellt.

Für die Evaluierung der Gerrit-Integration und der Abbildung von Code zum Modell konnten weitere Partner gewonnen werden, die entsprechende Systeme und Modelle für Experimente bereitstellen konnten.

Die Ergebnisse der Evaluierung flossen in Form von gemeinsamen Feedback-Sitzungen an die Projektpartner zurück. Auf Basis des Feedbacks wurden gemeinsam geeignete Anpassungen der Prozesse und Werkzeuge entwickelt und für eine weitere Iteration der Evaluierung umgesetzt.

3 Zahlenmäßiger Nachweis

Das Projekt war für eine Laufzeit von 36 Monaten angelegt. Tabelle 1 schlüsselt die tatsächlichen Kosten des Gesamtvorhabens auf. Neben den Personalkosten fielen Reisekosten an, um die Ergebnisse des Vorhabens auf einschlägigen Konferenzen und Tagungen zu präsentieren. Darüber hinaus wurde ein Werkstudent (i.d.R. 12 Std./Woche) über die Projektlaufzeit beschäftigt, welcher für Implementierungs- und Testarbeiten der im Rahmen des Projekts zu entwickelnden Prototypen eingesetzt wurde. Im Laufe des Projekts wurden durch die CQSE GmbH aufgrund der vielversprechenden Zwischenergebnisse zusätzliche Mittel eingebracht.

Die TU München plante ursprünglich mit Aufwänden von 194.999,00 Euro, bei einer Förderquote von 100%. Die tatsächlichen Ausgaben betrugen 193.807,77 Euro.

	CQSE GmbH	**TU München**
Personal	738.870,11 Euro	184.686,52 Euro
Reisekosten	7.930,54 Euro	9.121,25 Euro
Werkstudenten	24.160,00 Euro	–
Summe Kosten	**770.960,65 Euro**	**193.807.77 Euro**
Bewilligte Förderquote	**60%**	**100%**
Zuwendung	**434.309,00 Euro**	**193.807.77 Euro**

Tabelle 1: Kostenkalulation

4 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Softwareentwicklung findet heute in fast jedem mittleren und größeren Unternehmen statt. Technologieunternehmen, wie etwa in der Automobil-, Avionik- oder der Automatisierungstechnikbranche entwickeln und warten Software in immer größerem Ausmaß. Eine effizientere Qualitätssicherung und langlebigere Systeme sind in diesen Branchen ein signifikanter Erfolgsfaktor. Aber auch im Bereich der Banken und Versicherungen wird in großem Stil Software entwickelt, entweder durch interne Teams (ggf. mit Unterstützung externer Kräfte) oder durch Zulieferer (z.B. über Softwarehäuser und Individualsoftwareentwickler). In beiden Szenarien sind die Ergebnisse dieses Projekts relevant, um einerseits die Entwickler bei der Qualitätssicherung besser zu unterstützen und andererseits den Qualitätsstand zugelieferter Software bereits während der Entwicklung im Blick zu haben.

Durch eine transparentere, einfacher und kurzfristiger durchführbare Messung und Bewertung der Softwarequalität besteht für softwareentwickelnde Unternehmen in Deutschland nicht nur die Möglichkeit Software höherer Qualität zu produzieren, sondern auch ihren nationalen und internationalen Auftraggebern nachzuweisen, dass die von ihnen entwickelten Systeme “Made-in-Germany” einem hohen Qualitätsanspruch gerecht werden. Systeme hoher Qualität zeichnen sich durch eine gute Änderbarkeit und Anpassbarkeit sowie niedrige Wartungskosten aus. Diese Faktoren können genutzt werden, um sich im Markt von Anbietern aus Billiglohnländern abzugrenzen. Bisher war die innere Qualität eines Softwaresystems für die Auftraggeber sehr intransparent. Daher war der Preis der Entwicklung oftmals das Hauptentscheidungskriterium der Auftragsvergabe, was zu starken Outsourcing- und Offshoring-Tendenzen führt. Durch eine bessere Transparenz von Qualitätseigenschaften kann ein Umdenken in Richtung einer langfristigeren Vergabestrategie erreicht und damit eine verbesserte Position der Softwareentwicklung in Deutschland erzielt werden.

Wissenschaftlich-technisches Risiko Die systematische Erfassung, Nutzung und Kombination von Kontextinformation aus Softwareentwicklungsprojekten für die Priorisierung von Qualitätsdefiziten waren zu Beginn des Projekts wissenschaftliches Neuland. Es gab nur vereinzelte punktuelle wissenschaftliche Publikationen in diesem Bereich und keine praktischen Erfahrungen. Darüber hinaus handelt es sich hierbei um ein sehr herausforderndes Forschungsgebiet, für das höchstqualifizierte Mitarbeiter benötigt werden, die sowohl Erfahrungen im Bereich der statischen und dynamischen Analyse mitbringen, als auch über die notwendigen Kenntnisse effizienter Algorithmen zur Extraktion der benötigten Daten („data mining“, maschinelles Lernen).

Wirtschaftliches Risiko Für ein Unternehmen stellt die Erforschung derartiger Verfahren ein hohes Investitionsrisiko da, weil bisher auf diesem Gebiet sehr wenige Vorarbeiten existieren. Insbesondere KMUs können dieses Risiko aufgrund ihrer Größe nur in begrenztem Umfang tragen, ohne dabei ihre finanzielle Basis zu gefährden.

Notwendigkeit staatlicher Förderung Aufgrund der genannten Risiken ist eine Entwicklung allein aus Eigenmitteln wirtschaftlich kaum vertretbar. Eine erfolgreiche Realisierung dieses Ansatzes wäre dagegen nicht nur für die Vermarkter derartiger Werkzeuge ein Erfolg, sondern auch für die vielen Software-entwickelnden Unternehmen, die von einer effizienteren und effektiveren Qualitätssicherung und damit geringeren Entwicklungs- und -wartungskosten, langlebigerer Software und letztendlich einem höheren Return-On-Investment ihrer Softwareentwicklung profitieren würden.

Durch eine zielgerichtetere und effektivere Steuerung von Qualitätssicherungsmaßnahmen lassen sich Softwareprodukte von höherer Qualität mit niedrigeren Kosten erstellen. Dadurch werden die entsprechenden Techniken auch für KMUs zugänglich. Software ist heutzutage ein entscheidender Innovationstreiber in fast allen Industriezweigen (z.B. Automobil, Avionik, Automatisierungstechnik etc.). Auch im Bereich des Finanzsektors (Versicherungen, Banken etc.) und vielen weiteren Branchen ist die Beherrschung einer effizienten Softwareentwicklung ein entscheidender Wettbewerbsvorteil. Ein signifikanter Teil des Aufwands für Softwareentwicklung fließt in qualitätssichernde Maßnahmen. Somit stellen effiziente und effektive

Qualitätssicherungsverfahren für die Softwareentwicklung einen entscheidenden Erfolgsfaktor dar. Aufgrund dieser massiven Hebelwirkung in verschiedenste Domänen hinein ist eine Förderung der Entwicklung derartiger Technologien und Methoden im nationalen Interesse.

Derartige Verfahren in Deutschland zu entwickeln und zu vermarkten ist ein entscheidendes Signal für die deutsche Softwarebranche. Die hohen Qualitätsansprüche der Softwareentwicklung in Deutschland werden damit unterstrichen. Somit steht dieses Vorhaben auch im öffentlichen Interesse und bedarf öffentlicher Förderung.

5 Voraussichtlicher Nutzen

5.1 Wirtschaftlicher Nutzen

Die CQSE GmbH sieht das Thema der Qualitätsanalysen als strategisch wichtigste Entwicklung, welche die mittel- und langfristige Konkurrenzfähigkeit des Unternehmens sichert. Aus diesem Grund war die CQSE die treibende Kraft bei der Beantragung dieses Projekts.

Für die CQSE ist es wichtig, neue Entwicklungsparadigmen (z.B. Feature-Branch-basierte Entwicklung) und Trends (z.B. Modellbasierte Entwicklung) bedienen zu können. Aus dem direkten Feedback unserer Kunden lässt sich ein großer Bedarf nach einer entsprechenden Unterstützung erkennen.

Dank der engen Abstimmung zwischen dem Forschungsprojekt und dem Entwicklungsteam, konnten die ersten Ergebnisse bereits in die Produkte der CQSE übernommen werden. Einige der entwickelten Konzepte, speziell die Priorisierung von Testfällen, eröffnen neue Geschäftsmöglichkeiten, die das derzeit stark an Entwickler gerichtete Angebot der Firma auch in Richtung der Testabteilungen öffnet. Diese verbreiterte Zielgruppe erlaubt ein weiteres Wachstum der CQSE. Durch die weitere Entwicklung und unterstützende Beratung in diesem Umfeld plant die CQSE innerhalb der nächsten Jahre ihre Mitarbeiterzahl auf ca. 50-60 Mitarbeiter zu erhöhen. Auch langfristig wird mit einem hohen Wachstum geplant, das jedoch nur mit einem verbreiterten Angebot und einer guten Unterstützung für moderne Entwicklungsparadigmen realisierbar ist. Der Grundstein hierfür wurde in diesem Projekt gelegt.

5.2 Wissenschaftlicher Nutzen

Sowohl die TU München als auch die CQSE GmbH haben im Rahmen des Vorhabens in den Bereichen der Qualitätsanalyse von Softwaresystemen wichtige und international anerkannte wissenschaftliche Beiträge in Form von Veröffentlichungen und Vorträgen geleistet (vgl. Abschnitt 7).

Bereits jetzt werden die Ergebnisse aus Q-Effekt von der TU München genutzt, um in bilateralen Forschungsprojekten Industriepartner zu beraten. Es ergeben sich dadurch einige Möglichkeiten zur Akquise von Drittmittelprojekten für die TU München.

Dabei werden durch die in Q-Effekt entwickelnde Ergebnisse Blickwinkel auf das Thema Qualität eingenommen, da nun auch der Kontext von Artfefakten systematisch einbezogen werden kann. Das bildet die Basis für eine Reihe möglicher anschließenden Forschungsarbeiten.

6 Fortschritt bei anderen Stellen

Ein Fortschritt auf dem bearbeiteten Gebiet bei anderen Stellen, der sich mit den Themen des Vorhabens überschneidet, ist nicht bekannt.

7 Erfolgte und geplante Veröffentlichungen

7.1 Erfolgte Veröffentlichungen

Die nachfolgend aufgeführten Veröffentlichungen wurden innerhalb des Projekts erarbeitet und publiziert.

Jakob Rott, Rainer Niedermayr, Elmar Juergens, Dennis Pagano:
Ticket Coverage: Putting Test Coverage into Context
Proceedings of the 8th Workshop on Emerging Trends in Software Metrics (WETSoM'17), 2017.

Rainer Niedermayr, Elmar Juergens, Stefan Wagner:
Will My Tests Tell Me If I Break This Code?
Proceedings of the International Workshop on Continuous Software Evolution and Delivery (CSED'16), 2016.

Veronika Bauer, Tobias Völke and Sebastian Eder:
Combining Clone Detection and Latent Semantic Indexing to Detect Re-implementations.
Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER'16), 2016.

Jonas Eckhardt, Andreas Vogelsang, Henning Femmer:
An Approach for Creating Sentence Patterns for Quality Requirements.
Proceedings of the 6th International Workshop on Requirements Patterns (RePa'16), 2016.

Henning Femmer, Daniel Mendez Fernandez, Stefan Wagner, Sebastian Eder:
Rapid Quality Assurance with Requirements Smells.
Journal of Systems and Software, 2016.

Jonas Eckhardt, Andreas Vogelsang, Henning Femmer, and Philipp Mager:
Challenging incompleteness of performance requirements by sentence patterns.
Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16), 2016.

Roman Haas, Benjamin Hummel:
Deriving Extract Method Refactoring Suggestions for Long Methods.
Software Quality Days 2016 (SQD'16), 2016.

Nils Göde:
Quality Control in Action.
Softwaretechnik-Trends, Vol. 35, 2015.

Timo Pawelka, Elmar Juergens:
Is This Code Written in English? A Study of the Natural Language of Comments and Identifiers in Practice.
Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME'15), 2015.

Mohammad R. Basirati, Henning Femmer, Sebastian Eder, Martin Fritzsche, Alexander Widera:
Understanding Changes in Use Cases: A Case Study.
Proceedings of the 23rd International Requirements Engineering Conference (RE'15), 2015.

Daniela Steidl, Florian Deissenboeck:
How Do Java Methods Grow?
Proceedings of the 15th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'15), 2015.

Rainer Niedermayr, Tobias Roehm, Stefan Wagner:
Poster: Identification of Methods with Low Fault Risk
Proceedings of the 40th International Conference on Software Engineering Companion (ICSE'18), 2018.

7.2 Geplante Veröffentlichungen

Die folgenden Veröffentlichungen sind schon konkret geplant, allerdings sollen weitere Forschungsarbeiten auf den Ergebnissen von Q-Effekt aufbauen und zu Veröffentlichungen führen.

Rainer Niedermayr, Tobias Roehm, Stefan Wagner:
Too Trivial To Test? An Inverse View on Defect Prediction to Identify Methods with Low Fault Risk
Submitted to the 14th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'18) (currently under review).

Rainer Niedermayr, Stefan Wagner:
Is the Stack Distance Between Test Case and Method Correlated With Test Effectiveness?
Planned for submission to the 12th International Conference on Software Testing, Verification and Validation (ICST'19).

Rainer Niedermayr:
Evaluation and Improvement of Software Test Suites
Dissertation, Universität Stuttgart

Literatur

[BM06] C. Boogerd and L. Moonen. Prioritizing software inspection results using static profiling. In *SCAM '06*, 2006.

[Fag76] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, 15(3):182–211, 1976.

[GHK+01] T.L. Graves, M.J. Harrold, J.M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM TOSEM*, 2001.

[JHD+11] Elmar Jürgens, Benjamin Hummel, Florian Deissenboeck, Martin Feilkas, Christian Schlögel, and Andreas Wübbeke. Regression test selection of manual system tests in practice. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR'11)*, pages 309–312, 2011.

[KK09] Sami Kollanus and Jussi Koskinen. Survey of software inspection research. *The Open Software Engineering Journal*, 3:15–34, 2009.

[Kos07] Rainer Koschke. Survey of research on software clones. In *Duplication, Redundancy, and Similarity in Software*, 2007.

[Par94] David Lorge Parnas. Software aging. In *Proc. of the 16th International Conference on Software Engineering (ICSE'94)*, 1994.

[RAF04] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. A comparison of bug finding tools for java. In *Proc. of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, 2004.

[RH96] G. Rothermel and M.J. Harrold. Analyzing regression test selection techniques. *IEEE TSE*, 1996.

[SV01] H. Siy and L. Votta. Does the modern code inspection have value? In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, 2001.

[VGS13] R.D. Venkatasubramanyam, S. Gupta, and H.K. Singh. Prioritizing code clone detection results for clone management. In *IWSC '13*, 2013.

[Wag07] Stefan Wagner. *Cost-Optimisation of Analytical Software Quality Assurance*. Dissertation, TU München, 2007.