Prof. Jens Grabowski

(E-Mail: grabowski@informatik.uni-goettingen.de)
leitet seit 2003 die Forschungsgruppe „Softwaretechnik für verteilte Systeme" an der Georg-August-Universität Göttingen. Seine Forschungsinteressen liegen in den Bereichen der Modellierung und der Qualitätssicherung. Prof. Grabowski hat maßgeblich zur Entwicklung und Standardisierung des UML- Testprofils und der „Testing and Test Control Notation" (TTCN-3) beigetragen.

Philip Makedonski

(E-Mail:makedonski@informatik.uni-goettingen.de)
promoviert in der Gruppe „Softwaretechnik für verteilte Systeme" am Institut für Informatik an der Georg-August-Universität Göttingen. Seine aktuellen Forschungsinteressen liegen in den Bereichen Software-Evolution und Qualitätssicherung.

Thomas Rings

(E-Mail: rings@informatik.uni-goettingen.de)
ist seit 2008 Doktorand am Institut für Informatik an der Georg-August-Universität Göttingen sowie Spezialist-Taskforce-Experte am European Telecommunications Standards Institute in Sophia-Antipolis in Frankreich. Thomas Rings forscht in den Bereichen der Testautomatisierung, Interoperabilitätstesten und Performanztesten in verteilten Systemen.

Benjamin Zeiss

(E-Mail:zeiss@informatik.uni-goettingen.de)
ist Doktorand in der Gruppe „Softwaretechnik für verteilte Systeme" am Institut für Informatik der Georg-August-Universität Göttingen. Seine Interessen in der Forschung sind Qualitätsbewertung und Qualitätsverbesserung von Testspezifikationen sowie modellgetriebenes Testen und modellgetriebene Softwareentwicklung. Seine Forschung wird von der Siemens AG unterstützt.

Systematische Qualitätssicherung für Testartefakte

Es ist in der Softwaretechnik allgemein anerkannt, dass qualitativ hochwertige Software nicht ad hoc entsteht, sondern dass der Softwareerstellung Prinzipien, Methoden und Werkzeuge zugrunde liegen müssen, welche die Erreichung dieses Zieles gewährleisten. Ein Eckpfeiler dieser Methoden ist das Softwaretesten. Die Erstellung von Testartefakten ist in sich selbst gesehen ein spezielles Softwareprojekt und sollte den Prinzipien der Softwareentwicklung folgen, um die Qualität der Testartefakte zu verbessern, da diese Auswirkungen auf die Qualität des getesteten Softwaresystems haben können. In der Praxis ist das Bewusstsein für diese Problematik nicht sehr ausgeprägt, da die Prinzipien der Softwaretechnik vor allem im Testbereich nicht konsequent befolgt werden. Dieser Artikel bietet einen Überblick über existierende Qualitätssicherungsmaßnahmen der Softwaretechnik und diskutiert die Übertragbarkeit auf die Entwicklung von Testartefakten.

Motivation

In der Regel wird bei der Umsetzung einer Software ein großer Teil der Ressourcen für das Testen aufgewendet. Trotzdem wächst nach unserer Erfahrung bei verantwortlichen Projektleitern das Bewusstsein für die Notwendigkeit einer systematischen Qualitätssicherung von Testartefakten nur sehr langsam. Die Erfordernis von Qualitätssicherungsmaßnahmen für Testartefakte ergibt sich unmittelbar aus der Vielzahl der produzierten Artefakte (z. B. Testpläne, Testzweckbeschreibungen, abstrakte und ausführbare Testfälle, Testreports, Fehlermeldungen) und ihrer Größe. Beispielsweise haben standardisierte Testreihen (abstrakte Testfälle) im Telekommunikationsbereich inzwischen Größen von mehr als 80.000 Zeilen Quellcode. Daraus ergeben sich Softwarealterungseffekte und Wartungsprobleme wie man sie auch von der Implementierungsseite großer Softwareprojekte kennt. Dies ist insbesondere ein Problem, wenn sich ändernde Anforderungen auf Tests eines Softwareprojektes auswirken. Zudem bestehen für Tests ggf. andere Anforderungen an die Qualitätscharakteristiken, wie zum Beispiel Wiederverwendbarkeit. Deswegen sind Testartefakte häufig so verfasst, dass sie für verschiedene Testarten geeignet sind.

Wir betrachten das Testen als ein spezielles Softwareprojekt, bei dem Testartefakte für den systematischen Test eines Produkts entwickelt werden. Da die Qualitätssicherung für Softwareprojekte im Allgemeinen

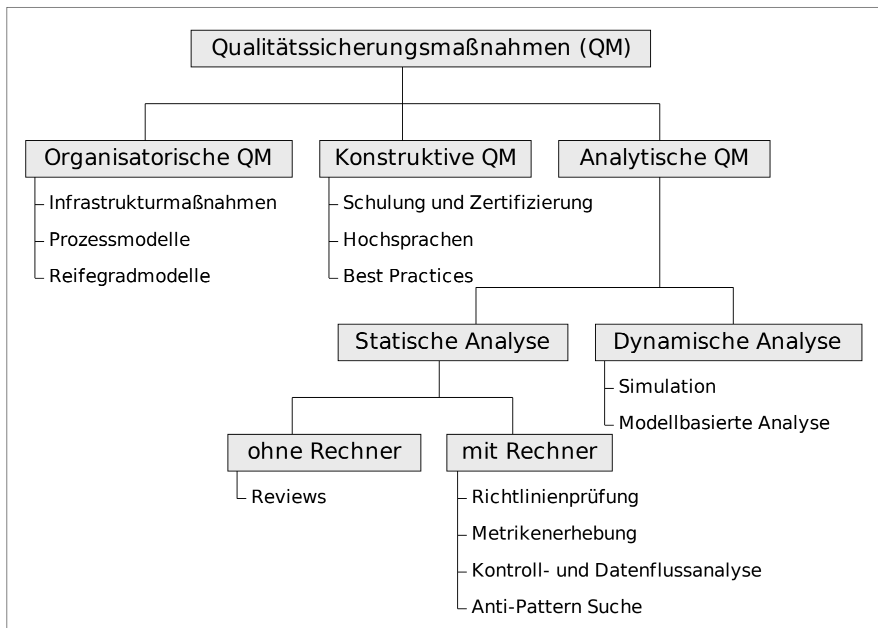

Abbildung 1: Qualitätssicherungsmaßnahmen für das Testen

gut erforscht und verstanden ist, stellt sich die Frage, wie man diese Erkenntnisse auf das Testen übertragen kann. Nachfolgend geben wir einen Überblick über die verschiedenen Maßnahmen zur Qualitätssicherung in der Softwaretechnik und erläutern die Anwendung der Qualitätssicherungsmaßnahmen zur systematischen Qualitätssicherung beim Testen.

Qualitätssicherungsmaßnahmen im Überblick

In der Softwaretechnik unterscheidet man zwischen organisatorischen, konstruktiven und analytischen Maßnahmen zur Qualitätssicherung [LuL07]:

- Organisatorische Maßnahmen: Sicherstellung der systematischen Durchführung der Entwicklung und Qualitätssicherung (z. B. durch Einführung eines Konfigurationsmanagements und Anwendung von Prozessmodellen)
- Konstruktive Maßnahmen: Vermeidung von Fehlern und Mängeln bereits bei der Entwicklung (z. B. durch die Schulung von Mitarbeitern und die Verwendung von Hochsprachen mit den zugehörigen Werkzeugen)
- Analytische Maßnahmen: Auffinden und Beseitigen von Fehlern und Mängeln (z. B. durch Softwaretests und methodisch durchgeführte Reviews)

Im Prinzip lassen sich Qualitätssicherungsmaßnahmen bis auf das Testen, welches die dynamische Analyse beinhaltet, für die systematische Qualitätssicherung von Testartefakten, wie in **Abbildung 1** dargestellt, anwenden. Nachfolgend erläutern wir die in der Abbildung beschriebenen Maßnahmen sowie deren Anwendung zur Qualitätssicherung des Testprozesses.

Organisatorische Maßnahmen

Bei den organisatorischen Maßnahmen wird zwischen Maßnahmen zur Verbesserung der Qualität der Infrastruktur und Maßnahmen zur Verbesserung der Managementqualität unterschieden. Auf die Infrastrukturqualität gehen wir nachfolgend nicht weiter ein, da der Nutzen und

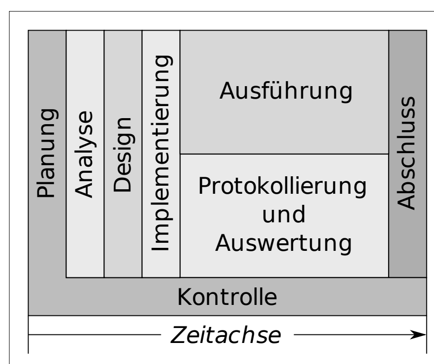

Abbildung 2: Fundamentaler Testprozess

die Notwendigkeit des Einsatzes von Werkzeugen für Testmanagement, Konfigurationsmanagement oder Fehlermanagement bei größeren Testprojekten außer Frage stehen.

Aspekte der Managementqualität sind der Einsatz von Prozessmodellen sowie die Bewertung des eigenen Prozesses mittels Reifegradmodellen. Als Beispiel für ein Testprozessmodell sei hier der fundamentale Testprozess des International Software Testing Qualification Board (ISTQB) [ISTQB] genannt. Wie in **Abbildung 2** beschrieben, gliedert sich der fundamentale Testprozess in die Phasen Planung, Analyse, Design, Ausführung, Protokollierung & Auswertung und Abschluss. Zusätzlich gibt es noch eine phasenübergreifende Kontrolle des gesamten Prozesses. Die während des Prozesses erstellten Artefakte sollen möglichst auf existierenden Standards basieren. Zum Beispiel sind die Regeln für die Erstellung des Testplans (Ergebnis der Planungsphase) und der Testspezifikationen (Ergebnis der Spezifikationsphase) im IEEE Standard 829 festgelegt. Details zum fundamentalen Testprozess des ISTQB finden sich in [Bla08] und [SuL03].

Neben den bekannten Reifegradmodellen für Softwareprozesse (z. B. Capability Maturity Model Integrated (CMMI) und Software Process Improvement and Capability Determination (SPICE)) existieren verschiedene Reifegradmodelle für Testprozesse. Basierend auf den Ideen von CMMI wurde zum Beispiel das Test Maturity Model integrated (TMMi) entwickelt. Ähnlich wie CMMI definiert TMMi fünf verschiedene Reifegrade, die ein Prozess erreichen kann. Bei TMMi heißen diese Reifegrade Initial (Stufe 1), Managed (2), Defined (3), Management and Measurement (4) und Optimization (5). Details zu TMMi finden sich auf den Webseiten der TMMi Foundation [TMMI]. Neben TMMi existieren mit Test Process Improvement (TPI), Critical Testing Processes (CTP) und Systematic Test and Evaluation Process (STEP) noch weitere Reifegradmodelle für Testprozesse.

Konstruktive Maßnahmen

Konstruktive Maßnahmen zur Qualitätssicherung versuchen Fehler und Mängel vor deren Entstehung zu verhindern. Sie beinhalten daher eine Qualifizierung der Mitarbeiter (inkl. Zertifizierung), die Verwendung von Hochsprachen (inkl. der zugehörigen

Werkzeuge) sowie der Anwendung von etablierten Techniken und Verfahren.

Der Erwerb von Zertifikaten durch Schulung und Training sind Praxis gängigen Maßnahmen zur Qualifizierung von Mitarbeitern. Als Beispiel für eine Mitarbeiterschulung im Bereich Testen sei hier das Certified Tester-Programm des ISTQB genannt. Für den Certified Tester existieren mit Foundation Level, Advanced Level und Expert Level drei Ausbildungsstufen. Der Lehrplan der Certified Tester Level Foundation beinhaltet einen Überblick über den fundamentalen Testprozess, diskutiert die unterschiedlichen Testarten (z. B. Abnahme-, System-, Integrations- und Komponententest) und beschreibt die grundlegenden Techniken für die Testfallerstellung (z.B. Anweisungs-, Zweig- und Pfadüberdeckung für das White-Box Testen und die Äquivalenzklassenanalyse für das Black-Box Testen). Zum Erwerb des Certified Tester Advanced Level ist Berufserfahrung im Bereich des Testen notwendig. Der Certified Tester Expert Level befindet sich noch in Vorbereitung. Weitere Informationen zum Certified Tester finden sich in [Bla08] und [SuL03] sowie auf den Webseiten des German Testing Boards (GTB) [GTB].

Üblicherweise werden größere Softwareprodukte in einer Hochsprache wie Pascal, C++ oder Java entwickelt, sodass bestimmte Fehlerquellen bereits durch die Syntax- und Semantikprüfungen eines Compilers ausgeschlossen werden können. Demgegenüber werden bei komplexen Testprojekten eine Vielzahl von Skriptsprachen und proprietären (werkzeugherstellerspezifischen) Testsprachen eingesetzt. Zur Vermeidung einer Abhängigkeit von bestimmten Werkzeugherstellern, der Reduzierung von Schulungskosten und zum Aufbau von Test-Know-how sollten, sofern möglich, standardisierte Testsprachen eingesetzt werden. Der de facto-Standard für den Komponententest (üblicherweise ein Glass-Box Test) sind xUnit-Frameworks. Der bekannteste Vertreter ist JUnit [JUnit] zum Testen von Java-Programmen. Andere Vertreter sind CppUnit (für C++ Programme), DUnit (für Delphi-Programme) oder NUnit (für die .NET Plattform). Für System- und Abnahmetests wurden in den letzten Jahren die Testing and Test Control Notation Version 3 (TTCN-3) vom European Telecommunications Standards Institute (ETSI) entwickelt und standardisiert. Obwohl TTCN-3 im Telekommunikationsbereich entwickelt worden ist, wird TTCN-3 inzwischen in vielen anderen Bereichen wie z. B. Automotive, Medizin und Luftfahrt eingesetzt. TTCN-3 wird vom ETSI permanent gepflegt und weiterentwickelt sowie von verschiedenen Werkzeugherstellern unterstützt. Dazu werden regelmäßig zertifizierte TTCN-3-Schulungen (siehe [GTB]) und TTCN-3-Nutzerkonferenzen angeboten. Den TTCN-3-Standard und weitere Informationen zur Technologie finden sich auf der TTCN-3-Webseite [TTCN].

Um die Entstehung von Fehlern und Mängeln zu vermeiden, sollte man auf bewährte Methoden und Vorgehensweisen (engl. Best Practices) zurückgreifen. Dieses betrifft auch das Testen. Bewährte Methoden und Vorgehensweisen werden üblicherweise in Büchern oder auf Webseiten veröffentlicht. Zwei Beispiele für Best Practices sind Richtlinien und Patterns. Richtlinien vereinheitlichen die erstellten Artefakte, indem sie z. B. Regeln für Namensgebung, Formatierung, Dokumentation oder Kodierungsstil festlegen. Durch die Vereinheitlichung verbessern Richtlinien die spätere Wartung von Artefakten, die Kommunikation zwischen den Mitarbeitern und die Einarbeitung neuer Mitarbeiter. Patterns sind Schablonen für die bewährte Benutzung einer Sprache und beschreiben anerkannt gute Lösungen für häufig auftretende Situationen. Die systematische Verwendung von Patterns verbessert die Struktur und Wartbarkeit von Quellcode. Ebenso existieren Kodierungsrichtlinien und Kataloge mit Patterns für die Testsprachen JUnit (z. B. [Mes07]) und TTCN-3 (z. B. [Din08, Vou05]).

Ein wichtiger Aspekt bei den konstruktiven Qualitätssicherungsmaßnahmen ist eine leistungsstarke Werkzeugunterstützung. Neben einer Unterstützung bei der Programmierung durch z. B. Syntax-Hervorhebungen, intuitive Navigierbarkeit oder automatischer Vervollständigung von Namen, sollte auch die Einhaltung von Richtlinien automatisch prüfbar sein. Somit werden analytische Qualitätssicherungsansätze bereits bei der Erstellung von Artefakten angewandt. Bei der Verwendung von JUnit kann man hierfür auf eine Vielzahl von Open-Source-Werkzeugen (z. B. Checkstyle oder PMD) zurückgreifen. Eine Liste mit zum Teil sehr mächtigen TTCN-3 Werkzeugen findet sich auf der TTCN-3-Webseite [TTCN].

Analytische Maßnahmen

Da Irren bekanntlich menschlich ist, können konstruktive Maßnahmen zur Qualitätssicherung nicht alle Fehler und Mängel verhindern. Analytische Maßnahmen versuchen daher, vorhandene Fehler und Mängel zu finden und zu beseitigen. Bei den analytischen Maßnahmen unterscheidet man zwischen statischer und dynamischer Analyse. Eine statische Analyse prüft ein Artefakt ohne es auszuführen, während eine dynamische Analyse ein Artefakt während dessen Ausführung analysiert.

Bei der statischen Analyse wird zudem zwischen Verfahren, die ohne Rechnerunterstützung und solchen die mit Rechnerunterstützung durchgeführt werden, unterschieden. Das Review ist der Hauptvertreter der statischen Analyse ohne Rechnerunterstützung. Statische Prüfungen mit Rechnerunterstützung umfassen z. B. die automatisierte Prüfung der Einhaltung von Richtlinien, das Berechnen von Metriken, Kontroll- und Datenflussanalysen und das Suchen nach Anti-Patterns (sog. Bad Smells). Anti-Patterns beschreiben eine schlechte Benutzung einer Sprache und sollen durch Einhaltung von Richtlinien vermieden werden. Zur Beseitigung von Anti-Patterns sollte der Quelltext systematisch mithilfe sogenannter Refactorings (z.B. automatische Entfernung nicht verwendeter Definitionen oder Parameter) umgestellt werden.

Statische Analysen sind auch beim Testen weit verbreitet, wobei Reviews häufig die einzige Möglichkeit sind, um informelle Designdokumente oder natürlichsprachliche Beschreibungen von manuellen Tests zu beurteilen. Für das Testframework JUnit und die Testsprache TTCN-3 existieren zudem eine Reihe von Werkzeugen, die eine rechnerunterstützte statische Prüfung ermöglichen. Als Beispiel für ein Open-Source-TTCN-3-Werkzeug sei hier das TTCN-3 Refactoring and Metrics Tool (TRex) [TRex] genannt. TRex ist ein Eclipse-basiertes Werkzeug, das insbesondere das Auffinden von Anti-Patterns, das Erheben von Metriken und das Beseitigen von Anti-Patterns mittels Refactoring unterstützt. Eine TRex-Beispielanalyse von IPv6-Testreihen (Core Protocol v3.1.1) im Umfang von 67580 Zeilen TTCN-3-Quellcode zeigte 124 unbenutzte Parameter. Diese Parameter tauchen in verschiedenen Konstrukten auf. Dies lässt eine manuelle Erkennung solcher Parameter nur mit großem Aufwand realisieren. Die Konstrukte werden wiederum in verschiedenen Kontexten referenziert. Eine manuelle Beseitigung würde im Vergleich zu einer automatischen sehr hohe Kosten verursachen. Weitere Analysen finden sich in [Neu07]. Weitere Werkzeuge zur automatischen Prüfung von

Programmierrichtlinien und Dokumentationserzeugung, die auf TRex basieren, werden vom ETSI verwendet sowie weiterentwickelt. Hinweise zu TRex und zu diesen Werkzeugen finden sich auf der TTCN-3 Webseite [TTCN].

In der Softwaretechnik umfasst die dynamische Analyse insbesondere das Testen selbst. Das Testen von Tests ist sehr problematisch, da man hierfür Tests für Tests entwickeln müsste. Da sich diese Rekursion beliebig fortsetzen lässt, sind Tests für Tests nicht praktikabel. Sofern aber Emulatoren oder Modelle des späteren Softwareprodukts existieren, kann das korrekte funktionale Verhalten eines Tests durch eine gesteuerte Simulation gegen das Modell oder den Emulator geprüft werden. In der Forschung gibt es zudem einige Ansätze, in denen Modelle aus Testfällen generiert werden um Korrektheitseigenschaften nachzuweisen [ZuG09].

Fazit

Zu einer systematischen Qualitätssicherung für Testartefakte gehören organisatorische, konstruktive und analytische Maßnahmen. In diesem Artikel wurde dargestellt, welche Maßnahmen es gibt und wie diese im Testumfeld eingesetzt werden können. Die Praxis zeigt, dass insbesondere für große Softwareprojekte bereits viele Ressourcen für das Softwaretesten aufgewendet werden. Jedoch werden bekannte Qualitätssicherungsmaßnahmen zu selten im Testumfeld konsequent umgesetzt. Qualitätssicherung von Testartefakten sollte nicht als Bürde für den Tester angesehen werden, sondern als Mittel, das es ermöglicht vorausschauend zu agieren und langfristig teure Fehler zu vermeiden oder in einem frühen unkritischen Stadium zu entdecken. Im schlimmsten Fall führt geringe Qualität von Testartefakten die Testentwicklung ad absurdum – nämlich dann, wenn die Testartefakte nicht in der Lage sind existierende Fehler in der Form aufzudecken wie sie es sollten. ■

Referenzen

[Bla08] R. Black: Advanced Software Testing - Vol. 1, Guide to the ISTQB Advanced Certification as an Advanced Test Analyst. Rocky Nook, 2008.

[Din08] G. Din, D. Vega, I. Schieferdecker: Automated Maintainability of TTCN-3 Test Suites Based on Guideline Checking. In: Software Technologies for Embedded and Ubiquitous Systems (LNCS 5287, ISBN 987-354087 7844), Springer, 2008, Seiten: 417-430.

[JUnit] http://junit.org

[GTB] http://www.german-testing-board.de

[ISTQB] http://www.istqb.org

[LuL07] J. Ludewig, H. Lichter: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dPunkt.verlag, 2007.

[Mes07] G. Meszaros: xUnit Test Patterns: Refactoring Test Code. Addison-Wesley Longman, 2007.

[Neu07] H. Neukirchen, M. Bisanz: Utilising Code Smells to Detect Quality Problems in TTCN-3 Test Suites. Proceedings of the 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2007), June 26-29 2007, Tallinn, Estonia. Lecture Notes in Computer Science (LNCS) 4581, Springer, Heidelberg, Juni 2007, 228-243.

[Neu08-1] H. Neukirchen, B. Zeiß, J. Grabowski, P. Baker, D. Evens: Quality assurance for TTCN-3 test specifications. In: Software Testing, Verification and Reliability (STVR), Volume 18, Issue 2. (ISSN 0960-0833) Wiley, 2008, Seiten: 71-97.

[Neu08-2] H. Neukirchen, B. Zeiß, J. Grabowski: An Approach to Quality Engineering of TTCN-3 Test Specifications. In: International Journal on Software Tools for Technology Transfer (STTT), Volume 10, Issue 4. (ISSN 1433-2779), Springer, 2008, Seiten: 309-326.

[SuL03] A. Spillner, T. Linz: Basiswissen Softwaretest. dPunkt.verlag, 2003.

[TMMI] http://www.tmmifoundation.org

[TRex] http://www.trex.informatik.uni-goettingen.de

[TTCN] http://www.ttcn-3.org

[Vou05] A. Vouffo-Feudjio, I. Schieferdecker: Test Patterns with TTCN-3. In: Formal Approaches to Software Testing (ISSN 0302-9743), Springer, 2005, Seiten: 170-179.

[ZuG09] B. Zeiß, J. Grabowski: Analyzing Response Inconsistencies in Test Suites. Tagungsband zur 21st IFIP International Conference on Testing of Communicating Systems and the 9th Int. Workshop on Formal Approaches to Testing of Software (TESTCOM/FATES 2009), 2.-4. November 2009, Eindhoven, Niederlande.