Ing. Alexander Sinko, B.Sc.

Qualitätssteigerung in der Softwareentwicklung durch risikobasiertes Testen in allen Entwicklungsphasen

MASTER THESIS

zur Erlangung des akademischen Grades

Master of Science

Studium: Universitätslehrgang Business Manager/in

Alpen-Adria-Universität Klagenfurt

Gutachter
Ao.Univ.-Prof. Mag. Dr. Gernot Mödritscher
Alpen-Adria-Universität Klagenfurt
Institut für Unternehmensführung

Klagenfurt, August 2022

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- die eingereichte wissenschaftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe,
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, ein-schließlich signifikanter Betreuungshinweise, vollständig offengelegt habe,
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder sinngemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben (z.B. in Fußnoten) ersichtlich gemacht habe,
- die eingereichte wissenschaftliche Arbeit bisher weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und
- bei der Weitergabe jedes Exemplars (z.B. in gebundener, gedruckter oder digitaler Form) der wissenschaftlichen Arbeit sicherstelle, dass diese mit der eingereichten digitalen Version übereinstimmt.

Ich bin mir bewusst, dass eine tatsachenwidrige Erklärung rechtliche Folgen haben wird.

Alexander Sinko e.h. Hard, August 2022

Version 1.1: Leicht überarbeitete Fassung!

Danksagung

An dieser Stelle möchte ich all jenen Personen danken, die mir während meines Studiums und bei der Anfertigung dieser Masterthesis sowohl fachlich als auch persönlich zur Seite gestanden sind.

Für die fachliche Beratung zur Master-Thesis:

Gernot Mödritscher
Markus Mangiapane
Janine Hofmann

Für Korrekturen und Anregungen zur Master-Thesis:

Janine Hofmann
Anne-Kathrin Dietel

Für persönlichen Beistand während des Studiums:

Janine Hofmann
Andrea Riem u. dem Team von Friseur Veilchen
Ramona Fleisch u. Nala
Andrea Vogl u. Ramses
Wolfgang Zeller u. Gang

Für die Inhalte des Lehrganges allen Dozenten und Vortragenden, insb.:

Gernot Mödritscher und der Alpen-Adria-Universität Klagenfurt
Rita Faullant und dem M/O/T
Alexander Sitter
Beatrix Hohengartner
Christian Hartl
Florian Halser
Gerhard Giermaier
Günther Mosshammer
Jürgen Schelling
Martina Heinzle
Rahel Schomaker
Stephan Berchtold
Veronika Spatzenegger
Wolfram Auer

Für die Betreuung während des Studiums dem WIFI Vorarlberg, insb.:

Marzellina Feuerstein
Barbara Zraunig

Zusammenfassung

Eine der größten Herausforderungen in Projekten ist allgemein der Umgang mit Ressourcen. Besonders das Testen in Softwareprojekten findet meist unter hohem Zeitdruck und mit eingeschränkten Ressourcen statt. Risikobasiertes Testen (RBT) stellt eine Möglichkeit dar, aufgrund der Betrachtung (Identifikation und Bewertung) von Risiken, die Testaktivitäten vorrangig auf die wichtigen Teile eines Systems zu konzentrieren. In dieser Arbeit werden im ersten Teil die Grundlagen zur Softwareentwicklung, Softwarequalität und der Stellenwert des Testens vorgestellt. Im zweiten Teil wird zuerst auf die gegebene Heterogenität in der Literatur eingegangen, anschließend werden die Erkenntnisse und Strategien aus den verschiedenen Ansätzen in der Literatur hinsichtlich des RBT zusammengefasst und aufgezeigt, wie diese grundsätzlich angewendet werden können. Zum Abschluss werden Forschungslücken und Widersprüche in der Literatur aufgezeigt.

Abstract

One of the biggest challenges in projects in general is the usage of resources. Especially the testing in software projects most often takes place under high time pressure and with limited resources. Risk-based testing (RBT) is an option to focus software testing activities on the important parts of a system by considering (identifying and evaluating) their risks. In the first part of this thesis, the (basic) principles for software development, software quality and the im-portance of testing for software are presented. In the second part, the given heterogeneity in the literature is presented, strategies and conclusions of the different approaches regarding RBT are summarised and discussed. It will be shown how the approaches basically can be applied. Finally, research gaps and contradictions in the literature are pointed out.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

EU	Europäische Union	
GUI	Graphical User Interface	| dt. grafische Benutzerschnittstelle, Benutzeroberfläche
IT	Information Technology	| dt. Informations-Technologie
MVP	Minimum Viable Product	| dt. minimal funktionsfähiges Produkt
RBT	Risk-Based Testing	| dt. risikobasiertes Testen
TBR	Test-Based Risk Analysis	| dt. testbasierte Risikoanalyse
TDD	Test-Driven Development	| dt. testgetriebene Entwicklung
UI	User Interface	| dt. Benutzerschnittstelle
UX	User Experience	| dt. Nutzererfahrungen

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird in diesem Dokument auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten ausdrücklich gleichermaßen für alle Geschlechteridentitäten.

1 Einleitung

1.1 Ausgangssituation

Eine der größten Herausforderungen in Projekten ist allgemein das Ausbalancieren der Faktoren Ressourcen (Personal, Geld, …), Zeit (Termine) sowie Leistung (Ziel(e) in geforderter Qualität).[1] In Softwareprojekten, also Projekten zur Entwicklung von Software, kommt der Punkt Agilität[2] hinzu. Das bedeutet, dass während der Projektdurchführung die Ziele oft mehrfach angepasst werden. Anforderungen fallen weg, kommen hinzu oder werden in ihrer Priorität geändert. Dies führt zwar zu mehr Flexibilität, wodurch auf die gewonnenen Erkenntnisse während des Projekts eingegangen werden kann. Es erschwert aber die Projektplanung, da dementsprechend oft die Verfügbarkeit von Ressourcen und der Zeitplan geprüft und ggf. adaptiert werden müssen.[3]

Besonders das Testen findet in Softwareprojekten meist unter hohem Zeitdruck und mit eingeschränkten Ressourcen statt. Dies sorgt dafür, dass häufig lediglich ein Teil aller Testfälle ausgeführt wird.[4] Als Ursache dafür gibt Dowie[5] gleich mehrere Gründe an: So wird der Testaufwand entweder aufgrund fehlender Planung unterschätzt oder es kommen keine bzw. ungeeignete Vorgehensweisen und Schätzmethoden zum Einsatz. Zudem wird das Testen bei der Zuteilung von Ressourcen meist nachrangig behandelt. Und zuletzt werden Testphasen oft als Puffer für Verzögerungen bei der Entwicklung herangezogen.[6]

Das Testen von Software ist allerdings unerlässlich. Da die Abwesenheit von Fehlern in einem Programm nicht bewiesen werden kann, ist das Testen die einzige Möglichkeit, um zu prüfen, ob das Programm unter realen Bedingungen seine Spezifikationen erfüllt.[7] Ziel ist es, vor der Inbetriebnahme sicherzustellen, dass die Verwendung des Softwareprodukts keine negativen Auswirkungen hat.[8]

Für ein gutes Projektmanagement ist es daher notwendig, sich mit den beschriebenen Herausforderungen bei der Ressourcenplanung auseinanderzusetzen.

1.2 Problemstellung

Eine Möglichkeit sich diesem Ressourcenproblem anzunehmen ist risikobasiertes Testen (engl. „risk-based testing“, RBT). Risikobasiertes Testen ist ein Konzept, bei dem die Risiken eines Softwareprodukts die Entscheidungsfindung in allen Testphasen seines Entstehens unterstützen soll.[9]

---

[1] Vgl. Mödritscher (2020), S. 421
[2] Vgl. Fowler/Highsmith (2001), o.S.
[3] Vgl. Böhm (2019), S. 10
[4] Vgl. Felderer/Ramler (2014), S. 544
[5] Dowie (2009)
[6] Vgl. Dowie (2009), S. 1 f.
[7] Vgl. Mayr (2005), S. 257
[8] Vgl. ISO/IEC/IEEE 29119-1:2013 (2013), S. 13
[9] Vgl. Foidl/Felderer (2018), S. 809

Die Grundidee ist, angesichts der knappen Ressourcen und der mangelnden Zeit, den wichtigen (risikoreichen) Teilen und Aspekten eines Systems besondere Aufmerksamkeit zu schenken.[10]

Grundlegende Ansätze für risikobasiertes Testen in Softwareprojekten sind bereits in der Literatur beschrieben. Als Beispiele seien hier exemplarisch allgemeine Herangehensweisen und Ansätze von Bach[11], Ottevanger[12], Redmill[13] oder Veenendaal[14] erwähnt, sowie die Fallstudie von Amland[15] für eine Bankanwendung im Privatkundensegment und die Vorgehensmethode von Felderer und Ramler[16] für den industriellen Einsatz. Jedoch unterscheiden sich diese Arbeiten hinsichtlich Risikobestimmung/-bewertung teilweise erheblich voneinander. Zudem gibt es – sofern überhaupt erwähnt – große Unterschiede bei der Ausarbeitung und Anwendung von Softwaretests. Und zuletzt zeigen nur wenige Ansätze auf, wie sich die aufgezeigten Methoden in den Softwareentwicklungsprozess eingliedern lassen.

Ein weiterer vernachlässigter Aspekt ist der durch die Risikoidentifikation, -bewertung und -beherrschung entstehende Mehraufwand.[17] Wird das risikobasierte Testen rein der Testphase eines Softwareprojekts zugeordnet, fallen diese Aufwände wie in Abbildung 1 gezeigt direkt zu Lasten des eigentlichen Testens. In Folge muss der Mehraufwand für das risikobasierte Testen durch ein effizienteres Testen ausgeglichen werden.

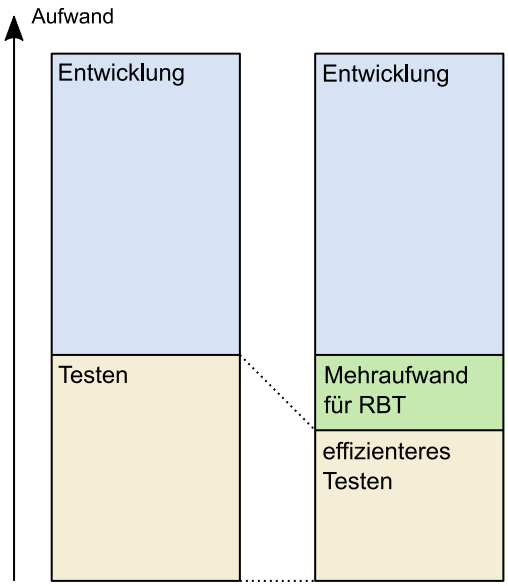

Abbildung 1: Mehraufwand für risikobasiertes Testen[18]

[10] Vgl. Ottevanger (1999), S. 1
[11] Bach (1999)
[12] Ottevanger (1999)
[13] Redmill (2004); Redmill (2005)
[14] Veenendaal (2011)
[15] Amland (2000)
[16] Felderer/Ramler (2014)
[17] Vgl. Felderer et al. (2012), S. 175
[18] Quelle: Verfasser

1.3 Zielsetzung

In dieser Arbeit werden zuerst die verschiedenen, in der Literatur vorhandenen Ansätze zu risikobasiertem Testen gesichtet und hinsichtlich eines übereinstimmenden Konzeptes geprüft. Anschließend werden verschiedene Möglichkeiten und Strategien ausgearbeitet, mit welchen risikobasiertes Testen den Softwareentwicklungsprozess zusätzlich unterstützen kann. Zum Abschluss werden die so gewonnenen Erkenntnisse in die Vorgehensmodelle der Softwareentwicklung integriert. Auf diese Weise soll das risikobasierte Testen die Softwareentwicklung in all ihren Phasen unterstützen.

Ziel ist es, unter anderem die Projektleitung hinsichtlich einer höheren Planungssicherheit in Bezug auf Aufwandsschätzung bezüglich des Testens zu unterstützen. Außerdem soll es der Qualitätssicherung ermöglicht werden, selbst agiler vorzugehen und Testressourcen auf die wesentlichen Bereiche der Software konzentrieren zu können. Dadurch soll die Qualität in jenen Teilen der Software gesteigert werden, die maßgeblich für das Softwareprodukt und in Folge den Projekterfolg verantwortlich sind. Gleichzeitig soll aufgezeigt werden, wie dabei der administrative Mehraufwand möglichst geringgehalten werden kann, um möglichst wenig zusätzliche Ressourcen zu benötigen.

Im Rahmen dieser Arbeit sollen folgende zwei Fragen beantwortet werden:

Frage 1:
Wie kann risikobasiertes Testen in allen Phasen der Softwareentwicklung eingesetzt werden?

Frage 2:

Wie kann dabei der administrative Aufwand des risikobasierten Testens geringgehalten werden?

2 Literaturreview

2.1 Die moderne Softwareentwicklung

2.1.1 Definition von Software

Für ein besseres Verständnis dieser Arbeit ist es hilfreich, sich vorab das Ergebnis eines Softwareprojekts bewusst zu machen, da dieses je nach Betrachter sehr unterschiedlich sein kann. Software definiert sich daher wie folgt:

> *„Mit Software bezeichnen wir die Gesamtheit aller Programme und Daten für eine bestimmte Informationsverarbeitungsaufgabe in ablauffähiger Form, zugeschnitten auf vorbestimmte Hardware, zusammen mit der für den Betrieb und die Weiterentwicklung wesentlichen Dokumentation.“* [19]

Somit steht am Ende eines Softwareprojekts mindestens ein Programm. Diese sind wiederum definiert als:

> *„Ein Programm ist eine in einer Programmiersprache abgefasste Verarbeitungsvorschrift (Algorithmus), die auf einer Rechenanlage (Computer) unter Nutzung und Festlegung von Datenformaten ausgeführt werden kann.“* [20]

Zusammengefasst ist das Ergebnis ***ein Set an Verarbeitungsvorschriften***, mit welchen Daten bearbeitet (erfasst, manipuliert, ausgetauscht, dargestellt, …) werden. Es gilt somit in einem Softwareprojekt diese Verarbeitungsvorschriften zu bestimmen und in einer für Hardware ausführbaren Sprache zu formulieren (= implementieren).

2.1.2 Herausforderungen der Softwareentwicklung

Um die Herausforderungen ganzheitlich zu betrachten, muss der ***Stellenwert von Software*** in und für Unternehmen betrachtet werden. Auf die zunehmende Digitalisierung und deren Gründe einzugehen ist an dieser Stelle nicht zielführend, sehr wohl aber auf die damit verbundene zunehmende Wertigkeit von Software in und für Unternehmen.[21] Wurde die IT – und Software im Speziellen – früher vor allem noch zur Unterstützung im Unternehmen eingesetzt, ist sie heute oft integraler Bestandteil von Geschäftsmodellen und den dazugehörenden Services.[22]

Zudem verlangt die ***Schnelllebigkeit*** der heutigen Zeit vielen Unternehmensbereichen einiges ab. Abkürzungen wie VUCA (volatility, uncertainty, complexity, ambiguity) oder BANI (brittle, anxiety, non-linearity, incomprehensible) versuchen die aktuelle Wirtschaftswelt zu beschreiben. Digitalisierung in all seinen Facetten wird als einer der wesentlichen Erfolgsfaktoren gesehen. Von „Industrie 4.0“ bis zur allumfänglichen „digitalen Transformation“. Automatisierung, Optimierung und

---

[19] Broy/Kuhrmann (2021), S. 17
[20] Broy/Kuhrmann (2021), S. 18
[21] Vgl. Alt et al. (2017), S. 10 f.; Broy/Kuhrmann (2021), S. 7 f.; Leimeister (2015), S. 2 ff.
[22] Vgl. Alt et al. (2017), S. 1; Erner (2019), S. 134

vor allem die Beherrschung von Komplexität sowie eine schnelle Adaptierung an eine sich stetig entwickelnde Umwelt sind die Ziele.[23]

Es hat sich gezeigt, dass eine starke Einbeziehung von Benutzern durch frühes Bereitstellen von ausführbarer Software eine schnelle Rückmeldung über deren Erfahrungen ermöglicht. Diese Erkenntnisse fließen in Folge wieder in die weitere Entwicklung ein und führen zur stetigen Verbesserung der Nutzungserfahrung.[24]

Das ausgefeilteste Softwaresystem ist nutzlos, wenn es von den vorgesehenen Nutzern nicht bedient werden will oder gar kann. Die ***Gebrauchstauglichkeit*** (engl. „usability“) bildet dabei das zu erfüllende Minimum. Der Benutzer muss in der Lage sein, ein für ihn gewünschtes Ziel möglichst effektiv und effizient zu erreichen. Wenn er während dieses Prozesses keine negativen Erfahrungen macht, ist davon auszugehen, dass er zufrieden gestellt ist – die Gebrauchstauglichkeit ist gegeben.[25]

Das ***Benutzererlebnis*** (engl. „user experience“, UX) hingegen ist wesentlich umfassender. Neben der Zufriedenstellung spielt das Erlebnis während und durch die Bedienung eine wesentliche Rolle. Dies beginnt mit den Erwartungen eines Nutzers an ein System vor dessen Bedienung und seinen Vorstellungen zum Ergebnis. Geht über die tatsächlich gemachten Erfahrungen während der Benutzung. Und endet schließlich mit dem Gefühl nach der Interaktion mit dem System. Da es sich dabei um eine subjektive Wahrnehmung handelt, ist bei der Umsetzung der Benutzerschnittstellen entsprechend auf die verschiedenen Benutzergruppen einzugehen.[26]

In der Konsequenz führen diese Faktoren – Änderungen des Umfelds und Erkenntnisgewinn – dazu, dass sich die ***Anforderungen an die Software während der Entstehung ändern***. Dies ist der Grund für Agilität, der Fähigkeit „beweglich“ genug zu bleiben, um Pläne und letztlich die Software an neue Gegebenheiten anzupassen. Da stetiges Umplanen jedoch Ressourcen und somit Geld und Zeit kosten,[27] ergibt sich daraus, dass eine möglichst späte Spezifikation von Details für die Umsetzung sinnvoll ist.[28]

Die Kunst ist es somit, ein Projekt so zu korrigieren und zu steuern, dass am Ende von Zeit und Ressourcen ein adäquates Softwareprodukt steht.

2.1.3 Stakeholder und ihre Anforderungen

Zu Beginn eines jeden Projekts steht eine Vision. Software soll ein Problem lösen, Unterstützung bieten oder etwas steuern. Jede Software soll Nutzen stiften, indem es die Bedürfnisse der Personen stillt, die direkt oder indirekt vom Einsatz der Software betroffen sind.[29]

---

[23] Vgl. Erner (2019), S. 133; Moskaliuk (2019), S. 1 ff.; Starker/Peschke (2021), S. 61 ff.
[24] Vgl. Broy/Kuhrmann (2021), S. 101
[25] Vgl. Geis/Tesch (2019), S. 14 ff.
[26] Vgl. Broy/Kuhrmann (2021), S. 201 f.; Geis/Tesch (2019), S. 17 ff.
[27] Vgl. Böhm (2019), S. 10 f.
[28] Vgl. Witte (2019), S. 65
[29] Vgl. Ebert (2019), S. 22 f.

Somit gilt es zuerst, alle von der Software tangierten ***Stakeholder*** (dt. „Interessensgruppen“) und deren Interessensvertreter ausfindig zu machen.[30] Das Übersehen von Stakeholdern und damit all ihrer Blickwinkel und spezifischen Anforderungen, kann für die Entwicklung, die Einführung oder den Betrieb der Software erhebliche Beeinträchtigungen bedeuten.[31] Besonders Interessensgruppen, die von der Software nur peripher tangiert werden, laufen Gefahr, übersehen zu werden. Als Beispiele seien die Rechtsabteilung genannt, die für die Einhaltung rechtlicher Rahmenbedingungen im Unternehmen verantwortlich ist, die IT-Administration, die Sicherheitsrisiken durch den Einsatz der Software ausschließen können muss, oder regulatorische Stellen, die gesetzliche Rahmenbedingungen überprüfen.[32]

Anschließend sind deren Bedürfnisse zu erfassen und in Anforderungen an die Software zu überführen.[33] Diese einfach anmutende Aufgabe – die ***Anforderungserhebung*** (engl. „requirements engineering“) – birgt bereits einige Herausforderungen und verlangt daher ein sauberes und methodisches Vorgehen. Dies ist auch der Grund, weshalb dieser Bereich mittlerweile eine eigene Disziplin in der Softwareentwicklung darstellt.[34] Dabei werden alle Eigenschaften und Bedingungen festgelegt, welche für die erfolgreiche Entwicklung, Einführung und den Betrieb der Software nötig sind. Des Weiteren alle Eigenschaften, die für die Einhaltung von Verträgen, Normen oder Spezifikationen erforderlich sind.[35] Es gilt zu beachten, dass die Anforderungen aus dem Blickwinkel des jeweiligen Stakeholders zu verfassen sind.[36] Daher ist ein grundlegendes Domänenwissen Voraussetzung, um mit den jeweiligen Stakeholdern auf Augenhöhe kommunizieren zu können. Oft sind diese Personen selbst Experten in ihrem Fachgebiet, daher erschließen sich die Ziele ihrer Anforderungen für Softwareentwickler nicht immer auf Anhieb. Die Folge können ***falsche Annahmen*** sein und damit verbunden oft ein nicht oder mangelhaftes Erfüllen der Anforderungen.[37]

Ebenfalls kann es vorkommen, dass Stakeholder ***Zwischenergebnisse*** als Bedürfnisse angeben. Dies sind Ergebnisse von Teilaufgaben, welche für die Erfüllung einer Aufgabe im aktuellen Arbeitsprozess nötig sind. Welche allerdings durch den Einsatz von Software wegfallen bzw. automatisiert werden können.[38]

Eine weitere Schwierigkeit besteht darin, ***implizierte Bedürfnisse*** sichtbar zu machen. In vielen Fachbereichen sind Dinge so selbstverständlich, dass diese bei der Erhebung von Anforderungen nicht erwähnenswert scheinen.[39] Noch größer ist die Herausforderung beim Erkennen ***latenter Bedürfnisse***. Also Bedürfnisse, deren sich ein Stakeholder (noch) nicht bewusst ist. Wobei gerade das Erfüllen dieser Bedürfnisse für Begeisterung sorgen kann.[40]

---

[30] Vgl. Geis/Tesch (2019), S. 98
[31] Vgl. Redmill (2004), S. 6
[32] Vgl. Geis/Tesch (2019), S. 98
[33] Vgl. Ebert (2019), S. 53; ISO/IEC/IEEE 15288:2015 (2015), S. 51 ff.; ISO/IEC/IEEE 12207:2017 (2017), S. 59 ff.; ISO/IEC/IEEE 29148:2018 (2018), S. 10
[34] Vgl. Droste/Merz (2019), S. 4; Ebert (2019), S. 4 ff.; Geis/Polkehn (2018), S. 1 ff.
[35] Vgl. Witte (2019), S. 61
[36] Vgl. ISO/IEC 25010:2011 (2011), S. 7
[37] Vgl. Bourque/Fairley (2014), S. 1 ff.; Broy/Kuhrmann (2021), S. 257 f.; Ebert (2019), S. 7 f.
[38] Vgl. Geis/Polkehn (2018), S. 44 f.
[39] Vgl. Witte (2019), S. 64
[40] Vgl. Broy/Kuhrmann (2021), S. 224; Geis/Polkehn (2018), S. 149

Zuletzt sei noch die ***Immunisierungsfalle*** erwähnt. Dabei werden Anforderungen unbewusst so spezifiziert, dass sie bereits eine angedachte Lösung implizieren. Davon können sowohl Stakeholder bei der Erhebung wie auch Softwareentwickler bei der Erfassung von Anforderungen betroffen sein. Als Beispiel dient der Mischhebel einer Dusche: Der Benutzer fordert eine Möglichkeit die Wassertemperatur mittels Armatur einzustellen. Das eigentliche Bedürfnis ist allerdings bei der gewünschten Wassertemperatur duschen zu können.[41]

Sind die Anforderungen erhoben, wird die nächste Herausforderung erkennbar: Die unterschiedlichen Ziele der Stakeholder haben mitunter ***widersprüchliche Anforderungen*** zur Folge. Eine definierte Struktur für Anforderungen hilft, diese aufzudecken, anschließend Prioritäten zu setzen und Kompromisse auszuarbeiten.[42]

Den Abschluss bildet dann die Frage, welche der erhobenen ***Anforderungen mit Software erfüllt werden können*** und sollen. Eine umfassende Anforderungserhebung zeigt bereits ein erstes Bild über Abläufe in der dafür nötigen Nutzungsumgebung. Es gilt somit zu klären, welche Aufgaben des Gesamtsystems durch Informationstechnologie (Soft- und Hardware), welche durch Menschen und welche durch nicht-informationstypische Technik zu erfüllen sind.[43]

Unter Umständen ist es nötig, vorab überhaupt die Machbarkeit einer angedachten technischen Lösung zu prüfen. Dafür können Machbarkeitsstudien erstellt werden, bei welchen angedachte Lösungen unter Praxisbedingungen überprüft werden. Die daraus gewonnenen Erkenntnisse helfen sicherzustellen, dass entsprechende Konzepte mit hinreichender Sicherheit funktionieren.[44]

Klar ist, dass Software immer zu einer betrieblichen Gesamtlösung beitragen muss.[45]

2.1.4 Stellgrößen in Softwareprojekten

Wurden die Ziele hinreichend definiert, liegt die zweite große Herausforderung darin, die geforderte Software in entsprechender Qualität in einem annehmbaren Aufwand zu realisieren.[46] Auch das Projektmanagement von Softwareprojekten agiert im Spannungsfeld des sogenannten „magischen Dreiecks im Projektmanagement“ (siehe Abbildung 2) mit den drei Eckpunkten:[47]

- Umfang (Funktionalität in geforderter Qualität)
- Ressourcen (Kosten, Personal, Infrastruktur)
- Zeit (Termine)

[41] Vgl. Geis/Polkehn (2018), S. 36
[42] Vgl. ISO/IEC/IEEE 12207:2017 (2017), S. 60; Witte (2019), S. 75
[43] Vgl. Broy/Kuhrmann (2021), S. 206
[44] Vgl. Brandt-Pook/Kollmeier (2020), S. 126
[45] Vgl. Broy/Kuhrmann (2021), S. 206
[46] Vgl. Broy/Kuhrmann (2021), S. 27
[47] Böhm (2019), S. 11; Brandt-Pook/Kollmeier (2020), S. 146; Broy/Kuhrmann (2021), S. 12; Ebert (2019), S. 281; Mödritscher (2020), S. 421; Witte (2019), S. 229

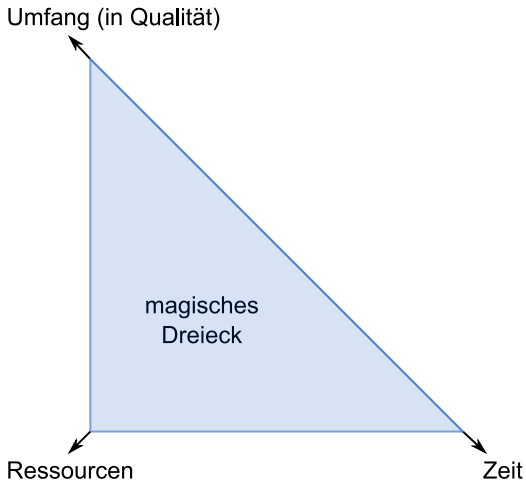

Abbildung 2: Magisches Dreieck im Projektmanagement[48]

Dabei veranschaulicht das magische Dreieck den Zusammenhang dieser drei Parameter. Eine Änderung an einem Parameter hat direkte Auswirkungen auf zumindest einen oder sogar beide anderen Parameter. Da Projekte grundsätzlich einen gewissen Grad an Unsicherheit beinhalten, wird eine Möglichkeit zur Steuerung benötigt.[49]

In klassischen Projekten wird der ***Umfang*** frühzeitig festgelegt. Er umfasst die Summe aller zu erbringenden Leistungen und Ergebnisse in angemessener Qualität eines Projekts. Die erforderlichen Ressourcen sowie die benötigte Zeit werden zuerst geschätzt und dienen anschließend auch zur Steuerung des Projekts.[50] In agilen Projekten hingegen werden Ressourcen und Zeit fixiert, wohingegen der Umfang gezielt angepasst werden kann. Diese Flexibilität ist von Anfang an eingeplant und ermöglicht eine Skalierung in beide Richtungen. Zudem wird sichergestellt, dass dem Kunden ein „Minimal Viable Product“ (MVP, dt. „minimal funktionsfähiges Produkt“) geliefert wird.[51]

Dabei spielt der Faktor ***Zeit*** eine immer wichtigere Rolle. Die Entwicklungszeit bzw. „Time-to-Market“ (dt. „Zeitraum bis zur Markteinführung“) gewinnt besonders im Hinblick auf die damit verbunden Opportunitätskosten an Bedeutung. Durch digitale Ökosysteme wie den mobilen App-Stores erwarten Kunden heutzutage eine schnelle Verfügbarkeit von Lösungen und eine stetige Aktualisierung dieser.[52] Dazu kommt, dass Terminschätzungen oft zu optimistisch sind. Die Auslieferung bzw. Markeinführung von Software erfolgt daher oft verspätet.[53]

Die ***Ressourcen*** bilden die größte Herausforderung. Sie umfassen alle für die Projektumsetzung nötigen Dinge wie Mitarbeiter, Infrastruktur und insbesondere die damit verbundenen Kosten. Da

[48] Quelle: In Anlehnung an Mödritscher (2020), S. 421
[49] Vgl. Ebert (2019), S. 280
[50] Vgl. Böhm (2019), S. 10; Brandt-Pook/Kollmeier (2020), S. 146; Broy/Kuhrmann (2021), S. 20; Ebert (2019), S. 280
[51] Vgl. Ebert (2019), S. 280
[52] Vgl. Alt et al. (2017), S. 2; Böhm (2019), S. 9
[53] Vgl. Broy/Kuhrmann (2021), S. 9

alle Faktoren direkt oder indirekt auf diesen Bereich einwirken, muss dieser für die Einhaltung des gesetzten Budgetrahmens ständig überwacht werden.[54]

Harry M. Sneed erweitert das „magische Dreieck“ und trennt in seinem „Teufelsquadrat“ den Umfang auf. Er schafft damit einen vierten Parameter, die „Qualität“ (siehe Abbildung 3). Der Parameter „Umfang“ umfasst nach Sneed nur noch funktionale Anforderungen, während alle nicht-funktionalen Anforderungen (= Qualitätsanforderungen) dem Parameter „Qualität“ zugeordnet werden.[55]

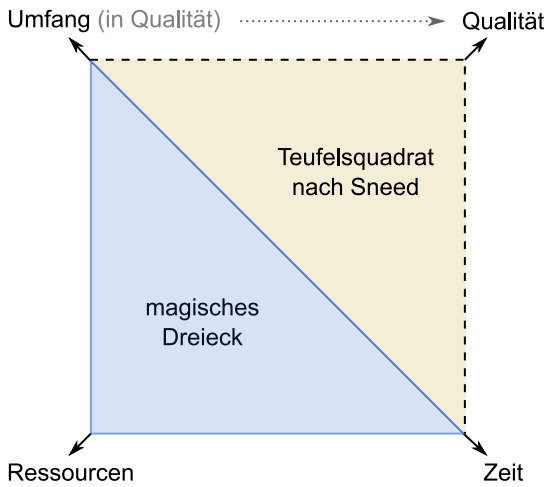

Abbildung 3: Erweiterung des magischen Dreiecks zum Teufelsquadrat nach Sneed[56]

[54] Vgl. Aichele/Schönberger (2014), S. 19; Broy/Kuhrmann (2021), S. 20
[55] Vgl. Aichele/Schönberger (2014), S. 19; Broy/Kuhrmann (2021), S. 207 f.; Sneed (2005), S. 38
[56] Quelle: In Anlehnung an Sneed (2005), S. 6

2.2 Qualität von Software

2.2.1 Definition von Softwarequalität

Das Wort „Qualität“ wird im modernen Sprachgebrauch meist dazu verwendet, die besondere Güte von Produkten oder Dienstleistungen hervorzuheben. In der Industrie und Technik steht der Begriff allerdings für eine neutrale, nicht wertende Form.[57]

So definiert der Standard IEEE 829-2008 (Standard for Software and System Test Documentation)[58] Qualität als:

> *„**(A)** The degree to which a system, component, or process meets specified requirements. **(B)** The degree to which a system, component, or process meets customer or user needs or expectations.“* [59]

Sinngemäß ins Deutsche übersetzt somit als (A) den Grad, in welchem ein System, eine Komponente oder ein Prozess spezifizierte Anforderungen erfüllt. Und (B) den Grad, in welchem ein System, eine Komponente oder ein Prozess Kunden- oder Benutzerbedürfnisse oder -erwartungen erfüllt.“

Im Standard ISO/IEC 25010:2011 (Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models)[60] wird Qualität definiert als:

> *„The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value.“* [61]

Sinngemäß übersetzt bedeutet dies: Die Qualität eines Systems ist der Grad, in welchem dieses System die spezifizierten und implizierten Bedürfnisse der verschiedenen Interessensgruppen (Stakeholder) erfüllt und ihnen damit einen Mehrwert bietet.

Im Standard ISO/IEC/IEEE 24765:2010 (Systems and software engineering – Vocabulary)[62] finden sich noch weitere Definitionen, welche allgemein alle auf die Erfüllung von Bedürfnissen und Anforderungen von Stakeholdern (insb. Kunden und Benutzer) abzielen.[63]

2.2.2 Qualität messbar machen

Zusammengefasst ist Qualität somit die Erfüllung der Bedürfnisse (Wünsche, Ziele, Interessen) verschiedener Interessensgruppen. Um den Grad dieser Erfüllung bestimmen zu können, benötigt es prüf- und messbare Kriterien, sogenannte ***Qualitätsmerkmale***. Diese werden bestimmt, indem

---

[57] Vgl. Sternad/Mödritscher (2018), S. 17 f.
[58] IEEE 829-2008 (2008)
[59] IEEE 829-2008 (2008), S. 9
[60] ISO/IEC 25010:2011 (2011)
[61] ISO/IEC 25010:2011 (2011), S. 2
[62] ISO/IEC/IEEE 24765:2010 (2010)
[63] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 287

die Bedürfnisse in einem ersten Schritt in Anforderungen an ein zu entwickelndes System überführt werden (siehe 2.1.3 „*Stakeholder und ihre Anforderungen*“).[64]

Um die Qualität einer Software vollumfänglich bestimmen zu können, müssen daher zuerst alle die Software betreffenden Stakeholder ermittelt werden. Dazu zählen unter anderem die drei Benutzergruppen:[65]

- Primäre Benutzer: Interagieren direkt mit dem System, um ihre Aufgaben zu erfüllen bzw. ihre Ziele zu erreichen.
- Sekundäre Benutzer: Wirken unterstützend, indem sie das System installieren, betreiben, administrieren, verwalten, warten u.v.m.
- Indirekte Benutzer: Agieren nicht mit dem System, verwenden jedoch dessen Ergebnisse.

Weiters sind andere tangierende Personen und Gruppen zu berücksichtigen. Dies sind beispielsweise die Organisationen, die die Software entwickeln, betreiben und/oder warten; oder die bereits erwähnten regulatorische Stellen, welche für die Einhaltung gesetzlicher und/oder branchenspezifischer Rahmenbedingungen verantwortlich sind.[66]

Alle diese Gruppen haben die Software betreffend ihre individuellen Bedürfnisse. Bei den daraus resultierenden Anforderungen handelt es sich meist um ***funktionale Anforderungen***. Diese beschreiben was ein System tun soll.[67] Daneben existieren ***Qualitätsanforderungen*** (auch nichtfunktionale Anforderungen genannt), welche qualitative Eigenschaften eines Systems beschreiben. Im Gegensatz zu funktionalen Anforderungen werden diese jedoch meist nur vage beschrieben. Üblicherweise stehen diese beide Arten von Anforderungen in engem Zusammenhang. So können funktionale Anforderungen implizit Qualitätsanforderungen bedingen.[68] Am Ende bildet dieses Set an prüf- und messbaren Kriterien (auch Softwaremetriken[69] genannt), die ***Qualitätsmerkmale***, mit welchen die Qualität einer Software erst messbar wird.[70]

Werden diese Qualitätsmerkmale nicht, mangel- oder gar fehlerhaft festgehalten, ergeben sich daraus mitunter kostenintensive Änderungen und Nachbesserungen.[71] Um diese möglichen Nacharbeiten zu vermeiden, ist es somit vorab sinnvoll, die Anforderungen einer Validierung zu unterziehen. Die zentrale Fragestellung dabei ist, ob das richtige System entwickelt wird und ob alle erfassten Anforderungen zutreffend und vollständig erfasst wurden.[72]

In der Literatur werden neben den funktionalen Anforderungen und den Qualitätsanforderungen teilweise noch weitere Arten von Anforderungen genannt.[73] Da allen Anforderungen jedoch gemein

---

[64] Vgl. Broy/Kuhrmann (2021), S. 254
[65] Vgl. ISO/IEC 25010:2011 (2011), S. 5 f.
[66] Vgl. ISO/IEC/IEEE 29148:2018 (2018), S. 9 f.
[67] Vgl. Ebert (2019), S. 30 f.
[68] Vgl. Ebert (2019), S. 30 f.
[69] Vgl. Broy/Kuhrmann (2021), S. 62
[70] Vgl. Broy/Kuhrmann (2021), S. 207 f.; ISO/IEC 25010:2011 (2011), S. 28 f.; ISO/IEC 25010:2011 (2011), S. 2
[71] Vgl. Ebert (2019), S. 87; Ebert (2019), S. 8
[72] Vgl. Broy/Kuhrmann (2021), S. 295 f.
[73] Vgl. Broy/Kuhrmann (2021), S. 207 f.; ISO/IEC/IEEE 29148:2018 (2018), S. 15; Witte (2019), S. 62

ist, dass sie über mess- und prüfbare Qualitätsmerkmale verfügen müssen, um verifizierbar zu sein, werden diese Kategorisierungen im weiteren Verlauf dieser Arbeit vernachlässigt.[74]

2.2.3 Qualitätsmerkmale nach den ISO/IEC Qualitätsmodellen

Die Normungsreihe ISO/IEC 250xx unterscheidet insgesamt drei Qualitätsmodelle, die zusammen einen Rahmen bilden, um (nach eigenen Angaben) alle Qualitätsmerkmale von Software abzudecken.[75] Dies sind zum einen das „Quality in use model" (dt. „Modell der Nutzungsqualität") und das „Product quality model" (dt. „Modell der Produktqualität"), welche beide im Standard ISO/IEC 25010:2011[76] definiert sind. Weiters noch das „Data quality model" (dt. „Model der Datenqualität"), welches im Standard ISO/IEC 25012:2008[77] beschrieben ist.[78] Zu erwähnen ist, dass wie in Abbildung 4 ersichtlich, die Produktqualität und die Datenqualität Teil der Nutzungsqualität sind.

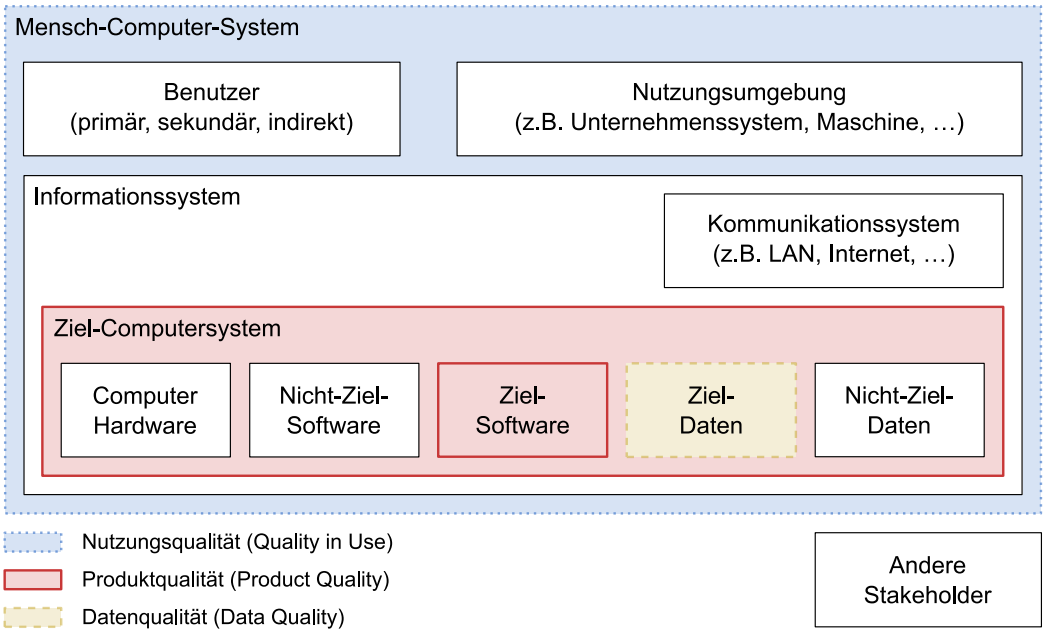

Abbildung 4: Zusammenhang der Qualitätsmodelle nach ISO/IEC 20510:2011[79]

Diese Qualitätsmodelle sollen dahingehend unterstützen, dass durch die gegebenen Qualitätsmerkmale die Qualitätsanforderungen auf Vollständigkeit geprüft werden können. Dabei sollen insbesondere die verschiedenen Blickwinkel der unterschiedlichen Interessensgruppen berücksichtigt werden. Ziel ist die Unterstützung bei der Spezifikation und der Bewertung von Software.[80]

---

[74] Vgl. Broy/Kuhrmann (2021), S. 296 f.; ISO/IEC 25010:2011 (2011), S. 2
[75] Vgl. ISO/IEC 25010:2011 (2011), S. 2
[76] ISO/IEC 25010:2011 (2011)
[77] ISO/IEC 25012:2008 (2008)
[78] Vgl. ISO/IEC 25010:2011 (2011), S. 2 f.
[79] Quelle: In Anlehnung an ISO/IEC 25010:2011 (2011), S. 5
[80] Vgl. ISO/IEC 25010:2011 (2011), S. 1

Die ***Nutzungsqualität*** beschreibt fünf Qualitätsmerkmale, die den Grad des Einflusses des gesamten Systems auf dessen Stakeholder messen. Der Fokus wird hier auf die gesamte Mensch-Computer-Interaktion gelegt.[81] Dabei ist zu beachten, dass diese Merkmale an die unterschiedliche Verwendung der verschiedenen Interessensgruppen geknüpft sind.[82]

- **Effektivität** (effectiveness)
  Die Genauigkeit und Vollständigkeit, mit welcher ein benutzerspezifiziertes Ziel erreicht wird.
- **Effizienz** (efficiency)
  Die aufgewendeten Ressourcen, mit welchen ein benutzerspezifiziertes Ziel erreicht wird.
- **Zufriedenheit** (satisfaction)
  Der Grad der Erfüllung der Bedürfnisse von Benutzern oder anderen Stakeholdern während der Verwendung im vorgesehenen Kontext. Dazu gehört u.a. der wahrgenommene Nutzen oder das Vertrauen in das erwartete Verhalten bzw. die Ergebnisse.
- **Risikofreiheit** (freedom of risk)
  Die Reduktion oder Vermeidung von gesundheitlichen, wirtschaftlichen oder ökologischen Risiken durch die Verwendung.
- **Kontextabdeckung** (context coverage)
  Der Grad, in welchem ein System in jedem spezifizierten Kontext und darüber hinaus verwendet werden kann (z.B. auf sehr kleinem Bildschirm, mit langsamer oder getrennter Datenverbindung, ohne Eingabemöglichkeit mittels Maus).

Die ***Produktqualität*** beschreibt acht Produktmerkmale eines Systems. Jedes Hauptmerkmal ist wiederum in Qualitätsmerkmale[83] unterteilt.[84]

- **Funktionale Eignung** (functional suitability)
  - Vollständigkeit (functional completeness)
  - Korrektheit (functional correctness)
  - Angemessenheit (functional appropriateness)
- **Leistungseffizienz** (performance efficiency)
  - Zeitverhalten (time behavior)
  - Ressourcenverbrauch (resource utilization)
  - Kapazität (capacity)
- **Kompatibilität** (compatibility)
  - Koexistenz (co-existence)
  - Interoperabilität (interoperability)
- **Benutzbarkeit** (usability)
  - Angemessene Verständlichkeit (appropriateness recognizability)
  - Erlernbarkeit (learnability)
  - Bedienbarkeit (operability)

[81] Vgl. ISO/IEC 25010:2011 (2011), S. 4
[82] Vgl. ISO/IEC 25010:2011 (2011), S. 3; ISO/IEC 25010:2011 (2011), S. 8 ff.
[83] Die Definitionen können bei Bedarf in der Norm nachgeschlagen werden.
[84] Vgl. ISO/IEC 25010:2011 (2011), S. 10 ff.; ISO/IEC 25010:2011 (2011), S. 3

    - Fehlertoleranz bzgl. Anwenderfehlern (use error protection)
    - Ästhetik der Benutzeroberfläche (user interface aesthetics)
    - Barrierefreiheit (accessibility)
- **Zuverlässigkeit** (reliability)
    - Ausgereiftheit (maturity)
    - Verfügbarkeit (availability)
    - Fehlertoleranz (fault tolerance)
    - Wiederherstellbarkeit (recoverability)
- **IT-Sicherheit** (security)
    - Vertraulichkeit (confidentiality)
    - Integrität (integrity)
    - Nachweisbarkeit (non-repudiation)
    - Verantwortlichkeit (accountability)
    - Authentizität (authenticity)
- **Wartbarkeit** (maintainability)
    - Modularität (modularity)
    - Wiederverwendbarkeit (reusability)
    - Analysierbarkeit (analyzability)
    - Modifizierbarkeit (modifiability)
    - Prüfbarkeit (testability)
- **Übertragbarkeit** (portability)
    - Anpassungsfähigkeit (adaptability)
    - Installierbarkeit (installability)
    - Ersetzbarkeit (replaceability)

Die ***Datenqualität*** beschreibt Qualitätsmerkmale der erzeugten und ausgegebenen Daten in einem bestimmten Verwendungskontext.[85]

- **Richtigkeit** (accuracy)
  Die wahren Werte der richtigen Daten werden korrekt wiedergegeben.
- **Vollständigkeit** (completeness)
  Alle erwarteten Daten (Attribute und verbundenen Entitäten) einer Entität sind vorhanden.
- **Widerspruchsfreiheit** (consistency)
  Die Daten sind widerspruchsfrei und kohärent zu anderen Daten.
- **Glaubwürdigkeit** (credibility)
  Nutzer sehen die Daten als wahr und glaubwürdig an.
- **Aktualität** (currentness)
  Die Daten geben den aktuellen Zustand zum entsprechenden Zeitpunkt (historisch bzw. aktuell) wieder.

[85] ISO/IEC 25012:2008 (2008) zitiert nach Gualo et al. (2021), S. 2

2.2.4 Zielkonflikte, Widersprüche und Wechselwirkungen

Wie bereits erwähnt, haben die verschiedenen Stakeholder ganz individuelle Bedürfnisse an eine Software. Oft sind diese divergierend, wodurch eine erfolgreiche Projektumsetzung nur durch das Erfassen und Lösen dieser Konflikte ermöglicht wird.[86] Werden diese Konflikte ignoriert oder nicht erkannt, werden sie auf die Ebene der Anforderungen getragen und verursachen dort Widersprüche mit den Zielen oder Anforderungen anderer Stakeholder. Freilich kann es im Laufe eines Projekts immer wieder zu Widersprüchen und Abhängigkeiten zwischen Anforderungen kommen.[87] Dies hängt zum einen mit der menschlichen Komponente zusammen, dass Anforderungen stark von der Interpretation und den Vorlieben der jeweiligen Vertreter der Stakeholder abhängen.[88] Allerdings können bei der Erfassung von Anforderungen auch Fehler gemacht worden sein. Und zuletzt können Erkenntnisse und Änderungen, die sich im Laufe eines Projekts ergeben können, zur Anpassung von Anforderungen führen.[89]

Somit stellt die Validierung der Anforderungen für jedes Softwareprojekt einen wesentlichen Beitrag zu dessen Erfolg dar. Geprüft werden dabei unter anderem Kriterien wie die Vollständigkeit, Korrektheit, Eindeutigkeit, Widerspruchsfreiheit, Überprüfbarkeit und Priorisierung einer jeden Anforderung.[90]

Qualitätsmerkmale können sich gegenseitig positiv wie negativ beeinflussen. Dies kann sowohl bi- wie auch unidirektional erfolgen. Auf der Ebene dieser Qualitätsmerkmale gilt es somit zu beachten, dass es für Softwareprojekte nicht möglich ist alle Qualitätsmerkmale in deren Ausprägungen zu definieren und messbar zu machen. Daher sollte auch hier eine Priorisierung aufgrund der Projektziele bzw. Ziele der Stakeholder erfolgen.[91]

Zum Beispiel schränkt eine höhere Zuverlässigkeit, bedingt durch die dafür nötigen Prüfungen hinsichtlich Eingaben, Laufzeitumgebungen und ähnlichem, die Leistungseffizienz ein. Wobei eine höhere Wartbarkeit sich hinsichtlich besserer Fehlerbehebungsrate positiv auf die Zuverlässigkeit auswirkt.

---

[86] Vgl. Ebert (2019), S. 58; ISO/IEC/IEEE 12207:2017 (2017), S. 60
[87] Vgl. Ebert (2019), S. 81
[88] Vgl. Ebert (2019), S. 51
[89] Vgl. Ebert (2019), S. 7
[90] Vgl. Broy/Kuhrmann (2021), S. 295; Broy/Kuhrmann (2021), S. 29; Ebert (2019), S. 79 f.
[91] Vgl. ISO/IEC 25010:2011 (2011), S. 5; Mayr (2005), S. 126

2.3 Testen zur Sicherung der Softwarequalität

2.3.1 Die Notwendigkeit zu testen

Dijkstra merkte bereits 1976 an, dass das Testen von Software sehr effektiv das Vorhandensein von Fehlern (sog. „Bugs“) aufzeigen kann, während es „hoffnungslos inadäquat“ ist, um deren Abwesenheit zu zeigen.[92]

Mittlerweile ist es allgemein akzeptiert, dass die Erstellung von „perfekter“ Software nicht möglich ist. Das Testen von Software ist daher vor der Auslieferung nötig, um vorab möglichst viele Fehler zu entdecken und in Folge den Kunden vor negativen Auswirkungen durch den Einsatz der Software zu bewahren.[93]

2.3.2 Herausforderungen bei Softwaretests

Der Aufwand für das Testen von Software wird meist stark unterschätzt. Grundsätzlich übertrifft der Aufwand des Testens jenen der eigentlichen Entwicklung erheblich.[94]

Für ein möglichst vollständiges Testen müssen

- alle Programmfunktionen ausgelöst,
- alle Prüfungen und Kontrollen angesprochen,
- alle Sonderfälle aus der Sicht der Programmierung berücksichtigt,
- alle Programmanweisungen ausgeführt,
- alle Programmverzweigungen einbezogen und
- alle Programmschleifen aktiviert und durchlaufen

werden.[95] Zudem ist die Software nicht nur hinsichtlich der gewünschten Ergebnisse zu prüfen (= Positiv-Test). Es ist auch sicherzustellen, dass die Software bei unerwarteten Eingaben oder Systemzuständen angemessen reagiert (= Negativ-Test).[96]

Tests müssen als Strichproben gesehen werden. Um das Testen effektiv zu gestalten, gilt es entsprechend gute Testfälle zu generieren. Das Ziel ist dabei, die Wahrscheinlichkeit einen Fehler zu finden zu maximieren. Von besonderem Interesse sind dabei schwerwiegende Fehler, deren Auswirkungen einen hohen Schaden anrichten können. Bei der Bemessung der Schadenshöhe ist die relative Wichtigkeit von Projektzielen zu beachten.[97] Im Umkehrschluss bedeutet dies, dass fehlende oder mangelhafte Testfälle zu einem höheren Risiko für die Stakeholder durch fehlerhafte Software sorgen können.[98] Die wesentlichen zwei Dinge für das Testen sind zum einen lauf- und somit testbare Software,[99] sowie ein vorab erstellter Testplan. Die gesamte Koordination des

---

[92] Vgl. Dijkstra (1976), S. 20
[93] Vgl. ISO/IEC/IEEE 29119-1:2013 (2013), S. 13; ISTQB (2018), S. 14; Witte (2019), S. 17
[94] Vgl. Broy/Kuhrmann (2021), S. 491; Witte (2019), S. 63
[95] Vgl. Leimeister (2015), S. 300
[96] Vgl. Witte (2019), S. 174 f.
[97] Vgl. Broy/Kuhrmann (2021), S. 467; ISO/IEC 25010:2011 (2011), S. 5; ISO/IEC/IEEE 29119-1:2013 (2013), S. 24; Witte (2019), S. 12
[98] Vgl. Witte (2019), S. 13
[99] Vgl. Broy/Kuhrmann (2021), S. 465

Testens gilt als wesentlicher Erfolgsfaktor bei Projekten, weshalb die frühe Einbindung des Testmanagements empfehlenswert ist.[100]

2.3.3 Grundlagen und Begriffe

Die Abbildung 5 gibt einen Überblick über die im Anschluss erörterten Begrifflichkeiten zum Bereich der Softwaretests. Sie zeigt, wie aufgrund der festgelegten Teststrategie in Kombination mit den vorhandenen Testobjekten, Testspezifikationen festgelegt werden. Die für die Durchführung benötigte Umgebung inkl. der nötigen Ressourcen werden im Testrahmen festgelegt. Durch diese grobe Planung ist es möglich einen Testplan zu erstellen.

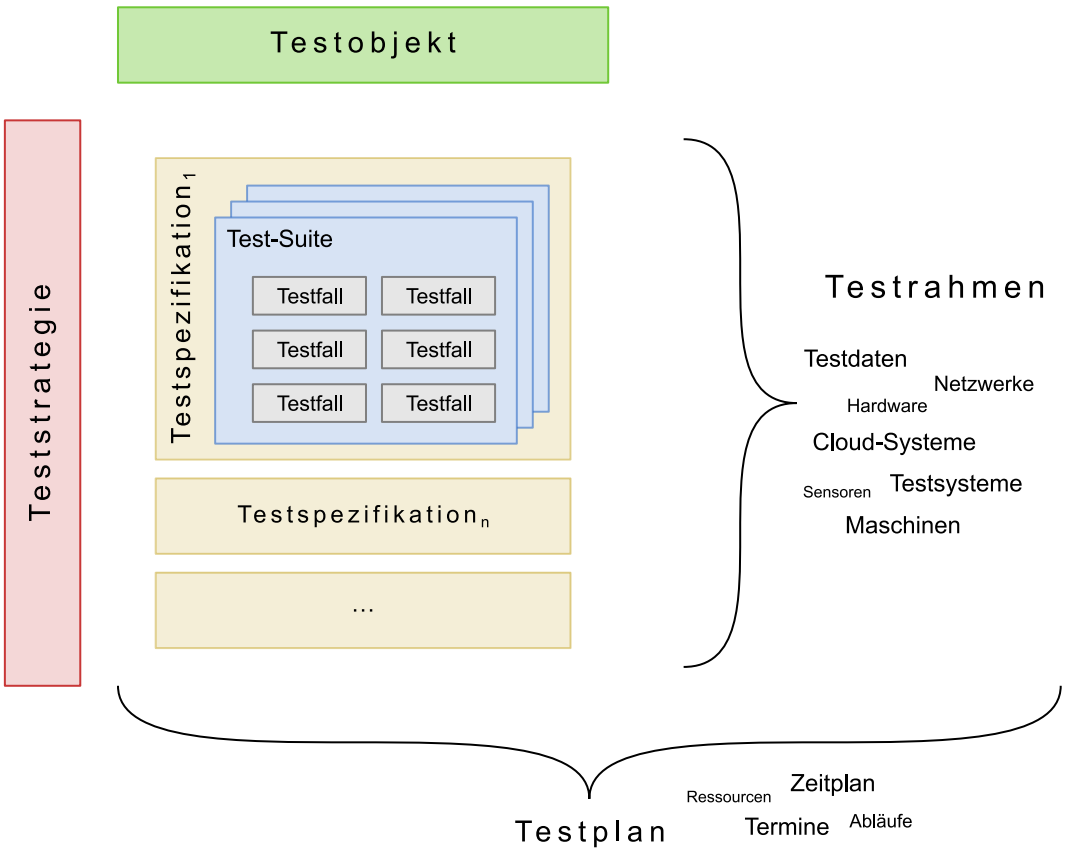

Abbildung 5: Überblick Begrifflichkeiten[101]

[100] Vgl. Droste/Merz (2019), S. 7
[101] Quelle: Verfasser

Testobjekt

Bei einem Testobjekt (engl. „test item“) handelt es sich um einen klar abgegrenzten Bereich, der getestet werden soll. Dies kann sowohl eine Funktion, ein Modul, ein Feature, ein ganzes System oder ein abgeschlossener Arbeitsablauf sein.[102]

Teststrategie

In der Teststrategie (engl. „test approach“) werden die Testaktivitäten festgelegt.[103]

Dafür werden aus den Erwartungshaltungen der einzelnen Stakeholder Ziele abgeleitet, die wiederum priorisiert werden. Hierbei sind bereits Projekt- und Produktrisiken zu berücksichtigen. Dadurch wird die Teststrategie auf den Kundennutzen ausgerichtet.[104]

Zudem werden die für die Testaktivitäten nötigen Verantwortlichkeiten beschrieben; die Testverfahren bzw. Testmethoden, die zur Anwendung kommen können; die Fehlerklassen, in welche entdeckte Anomalien (siehe Absatz „*Anomalie*“) eingeordnet; sowie das System, mit welchem Anomalien erfasst und deren Behandlung (Prüfung, Behebung, erneute Testung, …) verfolgt werden.[105]

Testspezifikation

Die Testspezifikation (engl. „test design“) definiert, wie die Überprüfung eines Testobjekts stattzufinden hat. Dies umfasst die Testverfahren und -methoden, deren genaue Anwendung auf das Testobjekt und die dafür benötigten Komponenten (z.B. Systeme, Module, Testdaten). Je nach Beschaffenheit des Testobjekts, kann es Sinn machen, mehrere Testspezifikationen auf verschiedene Bereiche eines Testobjekts anzuwenden.[106]

Neben den durch Abhängigkeiten bestimmten Voraussetzungen (Startbedingungen), ab wann das Testen eines Testobjekts überhaupt möglich ist, sind auch Abbruchbedingungen festzulegen. Sowohl Einflüsse von außen (Fehler in anderen Systemen) wie auch von innen (grundlegende Fehler im Testobjekt) können dafür sorgen, dass ein weiteres Testen nicht möglich bzw. sinnvoll ist.[107]

Test-Suite

Wird auf ein Testobjekt ein Set an Testfällen mit demselben Testverfahren angewendet, werden diese Testfälle üblicherweise zu einer Test-Suite zusammengefasst.[108]

---

[102] Vgl. Broy/Kuhrmann (2021), S. 467; ISO/IEC/IEEE 24765:2010 (2010), S. 370; ISO/IEC/IEEE 29119-2:2021 (2021), S. 6; Veenendaal (2011), S. 7
[103] Vgl. ISO/IEC/IEEE 29119-1:2013 (2013), S. 11; Ramler/Felderer (2015), S. 356 f.
[104] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 17
[105] Vgl. Droste/Merz (2019), S. 115; ISO/IEC/IEEE 29119-2:2021 (2021), S. 17 ff.; Ramler/Felderer (2015), S. 361; Witte (2019), S. 49 ff.
[106] Vgl. Broy/Kuhrmann (2021), S. 475; ISO/IEC/IEEE 29119-2:2021 (2021), S. 8; Witte (2019), S. 132
[107] Vgl. Witte (2019), S. 137
[108] Vgl. Broy/Kuhrmann (2021), S. 465; IEEE 829-2008 (2008), S. 51

Es gilt darauf zu achten, dass besonders manuelle Tests übersichtlich gehalten werden, da sich andernfalls gezeigt hat, dass Tests unvollständig oder überhaupt nicht ausgeführt wurden.[109]

Zudem haben Test-Suites im Bereich der automatisierten Tests besondere Bedeutung. Sie bündeln einzelne Unit-Tests (siehe 2.3.4 „*Methoden zur Qualitätssicherung*“ Absatz „*Unit Tests*“) zu Einheiten, welche wiederum die Grundlage für Regressionstests (siehe 2.3.4 „*Methoden zur Qualitätssicherung*“ Absatz „*Regressionstests*“) bilden.[110]

Testfall

Ein Testfall (engl. „test case“) überprüft ein einzelnes Testobjekt in einem festgelegten Umfeld (Zustände abhängiger Komponenten, Bedingungen zur Ausführung, eingegebene Daten usw.) auf ein spezifisches Ergebnis, eine spezifische Eigenschaft oder ein spezifisches Verhalten.[111] Dabei gilt es nicht nur auf erwartetes Verhalten bei gültigen Vorbedingungen oder Eingaben zu überprüfen (Positiv-Testfälle), sondern auch bei ungültigen Vorbedingungen oder falschen Eingaben (Negativ-Testfälle / Robustheitstest).[112] Zudem ist sicherzustellen, dass ein Testfall nicht durch Seiteneffekte unerwartete Änderungen in anderen Teilen des Systems verursacht.

Zu beachten ist außerdem, dass die Testfälle immer wieder verbessert und gewartet werden müssen. So können entdeckte Fehler, Variationen im Testablauf oder Anpassungen in der Software (Fehlerbehebung in anderen Modulen, Weiterentwicklungen) Auswirkungen auf die einzelnen Testfälle haben.[113]

Anomalie

Eine Anomalie ist ein Abweichen des Resultats eines Testfalls von dessen erwartetem Ergebnis.[114] Der Grund der Anomalie kann dabei ein Defekt (= Fehler) in dem zu testenden System sein. Allerdings könnten auch andere Gründe die Anomalie verursacht haben. Denkbar wären unter anderem das nötige Vorbedingungen des Testfalls nicht oder unzureichend beschrieben worden sind. Es könnte allerdings auch ein Fehler beim Erstellen des Testrahmens oder bei der Ausführung des Tests passiert sein.[115]

Beim Auftreten einer Anomalie gilt es somit zuerst deren Ursache festzustellen. Anschließend können adäquate Maßnahmen ergriffen werden, um deren zukünftiges Auftreten zu verhindern.

---

[109] Vgl. Broy/Kuhrmann (2021), S. 483
[110] Vgl. Broy/Kuhrmann (2021), S. 488; Witte (2019), S. 287 f.
[111] Vgl. Broy/Kuhrmann (2021), S. 465; IEEE 829-2008 (2008), S. 11; ISO/IEC/IEEE 24765:2010 (2010), S. 368; ISO/IEC/IEEE 29119-1:2013 (2013), S. 7
[112] Vgl. Broy/Kuhrmann (2021), S. 466
[113] Vgl. Witte (2019), S. 13
[114] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 16
[115] Vgl. Broy/Kuhrmann (2021), S. 466

Fehler

Ein Fehler als Ursache einer Anomalie ist somit nicht zwangsläufig ein Defekt der Software. In der Literatur werden drei Arten von Fehlern unterschieden:[116]

- Fehler (engl. „error“)
- Defekt (engl. „fault“ oder „bug“)
- Betriebsstörung (engl. „failure“)

Unter dem Begriff ***Fehler*** werden alle Arten von Ursachen und Umständen (falsche Daten, Fehlbedienungen, äußere Einflüsse usw.) verstanden, die Anomalien auslösen. Fehler können aufgrund von Defekten oder Betriebsstörungen auftreten, diese allerdings auch auslösen bzw. aufdecken.

Verhält sich ein System unerwartet oder anders als spezifiziert, liegt ein ***Defekt*** vor. Ob der Defekt durch eine fehlerhafte Implementierung oder eine fehlerhafte und/oder ungenügende Spezifikation verursacht wurde, ist dabei irrelevant.

Eine ***Betriebsstörung*** ist ein Zustand, in welchem ein System nicht mehr in der Lage ist, innerhalb seiner Parameter gültige Ergebnisse zu liefern. Dieser Zustand kann durch Fehler wie auch durch äußere Umstände (z.B. Hardwaregebrechen aufgrund von elektromagnetischer Einstrahlung) verursacht werden.

Testrahmen

Der Testrahmen (engl. „test environment“) legt die Testumgebung fest, mit welcher die Testobjekte getestet werden.[117]

Dies betrifft die nötigen Testsysteme (z.B. Hardware, Betriebssysteme, Laufzeitumgebungen) sowie unter Umständen Varianten davon. Zusätzlich müssen jene Teile der Software festgelegt werden, die für dessen Ausführung benötigt werden. So ergeben sich Abhängigkeiten der Testobjekte untereinander, damit Tests überhaupt durchgeführt werden können. Im Idealfall sind dies bereits erstellte und getestete Teile der Software. Stehen diese jedoch noch nicht zur Verfügung, können Dummies („Mocks“, „Stubs“, „Driver“, …) zum Einsatz kommen. Dies sind Objekte, mit deren Hilfe das Verhalten der benötigten Teile emuliert werden kann. Da Dummies ebenfalls zu entwickeln sind, können sie demzufolge auch fehlerhaft sein, was beim Testen berücksichtigt werden muss. Das Verwenden solcher Hilfsobjekte ist ebenfalls im Testrahmen zu definieren.[118]

Testplan

Im Testplan werden alle Testmaßnahmen in einen logischen und zeitlichen Ablauf gebracht. Es werden die erforderlichen Tests für die zu testenden Bereiche/Teile festgelegt. Weiters werden die dafür benötigten Ressourcen (z.B. Räumlichkeiten, Personal) und die Infrastruktur (z.B. Testumgebung, Testdaten, Schnittstellen) organisiert.[119]

---

[116] Vgl. Broy/Kuhrmann (2021), S. 466 f.; ISTQB (2018), S. 15 f.
[117] Vgl. IEEE 829-2008 (2008), S. 11; ISO/IEC/IEEE 24765:2010 (2010), S. 369
[118] Vgl. Broy/Kuhrmann (2021), S. 475; Witte (2019), S. 14
[119] Vgl. Droste/Merz (2019), S. 121; ISO/IEC/IEEE 29119-1:2013 (2013), S. 9 f.

Als Ausgangslage hierfür bietet sich die Teststrategie an. Aufgrund der darin festgelegten Tätigkeiten können Aufwands- und Ressourcenschätzungen erfolgen. In Abstimmung mit dem Projektplan lassen sich anschließend die Testaktivitäten einordnen. Zudem ermöglicht dies bereits die Prüfung, ob die Planung realistisch ist.[120]

Der Testplan ist somit eng verwoben mit dem Projektplan, da für die Tests einerseits ein entsprechender Entwicklungsstand erreicht sein muss, andererseits auch das Testen rechtzeitig zu den geplanten Meilensteinen erfolgen muss. In der Praxis wird gerne übersehen, dass die beim Testen entdeckten Fehler behoben und die Änderungen – besser noch der gesamte Bereich – anschließend nochmal geprüft werden müssen, was in der Folge wieder zum Entdecken von Fehlern führen kann. Zudem kann es zu Überschneidungen hinsichtlich der Ressourcen kommen. Beispielsweise kann Hardware, die üblicherweise den Softwareentwicklern zur Verfügung steht, temporär für die Tests herangezogen werden. Dementsprechend steht diese für die Entwicklung in diesem Zeitraum nicht zur Verfügung.[121]

2.3.4 Methoden zur Qualitätssicherung

In diesem Abschnitt werden ausgewählte Methoden zur Qualitätssicherung von Software vorgestellt. Dies soll einen Überblick über die Vielfalt der Ansatzpunkte und Möglichkeiten geben. Die Reihenfolge wurde so gewählt, dass die Methoden vom Entwurf über die Implementierung bis zur Ausführung und Abnahme der Software führen. Dennoch handelt es sich um einen Auszug, welcher keinen Anspruch auf Vollständigkeit erhebt.

Testen benötigt ausführbaren Quellcode,[122] weshalb es sich bei den ersten zwei Methoden (Reviews und Debugging) strenggenommen um keine Testmethoden handelt.

Reviews

Im Zuge eines Reviews wird ein Arbeitsergebnis (z.B. Anforderungen, Softwarearchitektur, Quellcode) einer Gruppe von Fachpersonal präsentiert und von diesen geprüft bzw. freigegeben.[123] Zweck dieser Begutachtung ist die Verbesserung der Qualität. Da grundsätzlich jedes Arbeitsergebnis begutachtet werden kann, existiert keine einheitliche Methode für Reviews an sich. Jedoch existieren definierte Verfahren für spezielle Reviews (z.B. Code-Reviews).[124]

Die Vorteile von Reviews sind die frühe Erkennung von Ungenauigkeiten, Widersprüchen oder Fehlern. Reviews regen zur Diskussion an, offenbaren alternative Möglichkeiten, zeigen Schwachstellen, tragen zum Verständnis bei und ermöglichen Softwarearchitekten und Softwareentwicklern an Erfahrungen und Fehlern zu wachsen. Zudem übernehmen alle Teilnehmer die Verantwortung für das Arbeitsergebnis.[125]

---

[120] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 22; Ramler/Felderer (2015), S. 362
[121] Vgl. Broy/Kuhrmann (2021), S. 475; Droste/Merz (2019), S. 118 ff.; Witte (2019), S. 132
[122] Vgl. Broy/Kuhrmann (2021), S. 465
[123] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 308
[124] Vgl. Brandt-Pook/Kollmeier (2020), S. 21
[125] Vgl. Broy/Kuhrmann (2021), S. 381 ff.; Witte (2019), S. 205

Bei Untersuchungen zu den Ergebnissen von Reviews hat sich zudem gezeigt, dass nur zirka ein Viertel der gefundenen Probleme funktionsbezogen sind und somit direkte Auswirkungen auf ein System haben. Die übrigen Dreiviertel betreffen wartungsrelevante Themen, welche erst bei einer Anpassung des Systems ausschlaggebend werden.[126]

Die Korrektur entdeckter Fehler oder das direkte Verbessern des Quellcodes ist in den Reviews nicht vorgesehen.[127]

Debuggen

Das Debuggen (engl. „debugging“) bezeichnet grundsätzlich das Finden und Beheben eines Fehlers oder Mangels in einem Programm.[128] Dies geschieht üblicherweise über die Rückverfolgung der Zustandsänderungen eines Systems durch die Ausführung des Quellcodes. Dazu findet entweder während der Ausführung eine Aufzeichnung statt (= „reversible execution“[129]) oder es werden die einzelnen Anweisungen im Quellcode schrittweise ausgeführt (= „single-step operation“[130]).

Der Unterschied zum Testen besteht darin, dass beim Testen Fehler gefunden, allerdings nicht behoben werden. Der große Vorteil liegt im wesentlich geringeren Aufwand durch Dokumentation, Kommunikation und Nachverfolgung der Behebung, wenn Softwareentwickler ihren Quellcode auf diese Weise vorab sorgfältig prüfen.[131]

Unit Tests

Ein Unit Test ist im Wesentlichen ein automatisierter Testfall. Wie der Name schon andeutet, wird dabei eine „Unit“ (dt. „Einheit“, „Bestandteil“, „Bauteil“, …) getestet.[132]

Auf Ebene der Codierung umfasst ein Unit Test nur eine Funktion.[133] In der Literatur werden jedoch Modul- oder Komponententests ebenfalls als Unit Tests bezeichnet. Wobei diese Begriffe teilweise synonym verwendet werden.[134] Andererseits wird jedoch explizit auf den Unterschied hingewiesen. Jedoch werden die Begriffe „Modul“ und „Komponente“ in der Literatur ebenfalls vertauscht.[135]

Für diese Arbeit ist eine präzise Abgrenzung des Begriffs „Unit Test“ nicht nötig. Daher wird folgend ein Unit Test als der automatisierte Test einer Einheit (Funktion, Modul oder Komponente) definiert.

Unit Tests bilden die Grundlage für das automatisierte Testen und in Folge auch für Regressionstests.[136]

---

[126] Vgl. Broy/Kuhrmann (2021), S. 468 ff.
[127] Vgl. Mayr (2005), S. 248
[128] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 94; Mayr (2005), S. 245; Witte (2019), S. 110
[129] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 308
[130] Vgl. ISO/IEC/IEEE 24765:2010 (2010), S. 328
[131] Vgl. Mayr (2005), S. 245 f.
[132] Vgl. Broy/Kuhrmann (2021), S. 485; ISO/IEC/IEEE 24765:2010 (2010), S. 386; Witte (2019), S. 75
[133] Vgl. Broy/Kuhrmann (2021), S. 485
[134] Vgl. Droste/Merz (2019), S. 225
[135] Vgl. Mayr (2005), S. 267; Witte (2019), S. 76
[136] Vgl. Broy/Kuhrmann (2021), S. 485

Automatisiertes Testen

Ganz allgemein sind automatisierte Tests alle Arten von Tests, die wiederholt automatisiert durchgeführt werden können. Dies umfasst im einfachsten Fall einen Unit Test (siehe vorherigen Absatz) und kann bis zum Einsatz von Testrobotern gehen.[137]

Bezüglich des Aufwands muss berücksichtigt werden, dass automatische Tests selbst nach ihrer Erstellung eine Testphase benötigen.[138] Da es sich dabei üblicherweise um manuell geschriebenen Quellcode handelt, könnte dieser Fehler enthalten, was es bei Auftreten von Anomalien zu prüfen gilt. Zudem müssen automatisierte Tests gewartet und ggf. erweitert werden. Dies kann durch Änderungen im Quellcode bedingt werden, wenn Erweiterungen oder Fehlerbehebungen stattfinden, oder wenn im Einsatz Fehler entdeckt werden, welche noch nicht durch Testfälle abgedeckt sind.[139]

Ebenfalls gilt es zu beachten, dass automatisierte Tests bereits bei der Erstellung der Systemarchitektur der Software berücksichtigt werden müssen, um nötige Schnittstellen für das Testen bereitzustellen. Zudem werden für die Tests häufig vordefinierte Testdaten benötigt, welche wiederum einer eigenen Verwaltung bedürfen.[140]

Wird beispielweise die Durchführung eines Unit Tests betrachtet, so sind drei Schritte nötig:[141]

- Vorbereiten – Objekt erstellen und mittels Testdaten in den richtigen Zustand versetzen
- Testen – die zu testende Funktion ausführen und die Ergebnisse erfassen
- Verifizieren – die Ergebnisse mit den erwarteten Daten abgleichen

Besonders der initiale Aufwand ist dafür verantwortlich, dass sich automatisierte Tests erst mittelfristig auszahlen.[142] Wobei Unit Tests und der Einsatz entsprechender Test-Frameworks mittlerweile gängige Praxis sind, sieht es auf Seiten von automatisierten Akzeptanztests noch wenig zufriedenstellend aus.[143]

Regressionstests

Regressionstests beugen mittels gezielter Wiederholung von Softwaretests einer schleichenden Verschlechterung (= Regression) der Softwarequalität durch Änderungen vor.[144] Ob die Änderungen am Quellcode (z.B. Erweiterungen, Fehlerbehebungen) oder der Umgebung der Software (z.B. Einsatz neuer Hardware) erfolgt, ist dabei nicht von Bedeutung.[145]

Regressionstests müssen nicht zwangsweise automatisiert erfolgen, jedoch ist dies im Sinne der eindeutigen Wiederholbarkeit, der schnelleren Lieferung von Testergebnissen und der damit verbundenen mittelfristig höheren Wirtschaftlichkeit zu empfehlen.[146] Üblicherweise wird bei der

---

[137] Vgl. Witte (2019), S. 231
[138] Vgl. Witte (2019), S. 234
[139] Vgl. Witte (2019), S. 236 f.
[140] Vgl. Witte (2019), S. 236; Witte (2019), S. 234
[141] Vgl. Rau/Schuster (2021), S. 128 f.
[142] Vgl. Witte (2019), S. 236
[143] Vgl. Brandes/Heller (2017), S. 9
[144] Vgl. Broy/Kuhrmann (2021), S. 500; ISO/IEC/IEEE 24765:2010 (2010), S. 295
[145] Vgl. Droste/Merz (2019), S. 226
[146] Vgl. Droste/Merz (2019), S. 174

Durchführung von Regressionstests nicht nur ein einzelner Testfall ausgeführt, sondern eine ganze Test-Suite (siehe 2.3.3 „*Grundlagen und Begriffe*“ Absatz „*Test-Suite*“).[147] Wichtig ist dabei, dass nicht nur die unmittelbar geänderten Teile der Software getestet werden, sondern auch die damit verbundenen Teile (Module, Komponenten, …). So können eventuell produzierte Folgefehler direkt erkannt werden.[148]

Smoke-Test

Beim Smoke-Test (dt. „Rauchtest“) wird die grundlegende Funktion eines Systems oder einer Komponente geprüft. Der Name stammt ursprünglich aus dem handwerklichen Bereich für Gas- und Wasserinstallationen und bezeichnete eine erste Leckageprüfung nach Neuinstallationen oder Reparaturen. Sinn dieses Tests ist die Sicherstellung, dass die Hauptfunktionen insoweit funktionieren, dass mit umfänglichen Testaktivitäten begonnen werden kann.[149]

Exploratives Testen

Per Definition handelt es sich beim explorativen Testen um simultanes Lernen, Entwerfen und Ausführen von Testfällen. Das heißt, dass die Testfälle nicht vorab in einem Testplan festgelegt wurden, sondern direkt beim Testen erdacht, ausgeführt und ggf. angepasst werden.[150]

Das explorative Testen ergänzt dabei das strukturierte Testen dahingehend, dass es den Testern die Freiheiten gewährt, abseits von Testplänen zu testen. Dies kann im frühen Stadion der Entwicklung eingesetzt werden, um einen ersten Eindruck der Qualität einer Software zu erlangen. Zudem ermöglicht es ein Testen, wenn die Anforderungen nur in mäßiger Qualität ausgearbeitet wurden. Und zuletzt lassen sich mit Hilfe dieser Methode komplexe Kombinationen von Geschäftsfällen, Spezialfälle und Fehlbedienungen überprüfen, wie sie im täglichen Gebrauch vorkommen können, aber in ihrem vollen Umfang nicht in den Anforderungen beschrieben sind.[151]

Wichtig ist, dass die Tests für die Rekonstruktion lückenlos und nachvollziehbar dokumentiert und/oder aufgezeichnet werden. Zudem bedarf es erfahrener Tester, die intuitiv Schwachstellen in Software lokalisieren können und auf Situationen und Erkenntnisse während des Testens eingehen können.[152]

Usability-Tests

Bei einem Usability-Test (dt. „***Gebrauchstauglichkeits-Test***“) werden repräsentative Benutzer (sog. Testteilnehmer) beim Erfüllen bestimmter Aufgaben mit einem interaktiven System beobachtet. Als repräsentativ gelten Benutzer, die dem Profil der jeweiligen Benutzergruppe (z.B. Endnutzer, Administrator) entsprechen. Diese Test-Methode ermöglicht das Aufdecken von Problemen hinsichtlich der Bedienung bzw. Benutzung des Systems. Dadurch werden Messungen von

[147] Vgl. Broy/Kuhrmann (2021), S. 488
[148] Vgl. Witte (2019), S. 120
[149] Vgl. Droste/Merz (2019), S. 68; Droste/Merz (2019), S. 141; Droste/Merz (2019), S. 226
[150] Vgl. Bourque/Fairley (2014), S. 4 ff.; ISO/IEC/IEEE 24765:2010 (2010), S. 135
[151] Vgl. Witte (2018), S. 120; Witte (2019), S. 188 ff.
[152] Vgl. Bourque/Fairley (2014), S. 4 ff.; Witte (2019), S. 190

Effektivität, Effizienz oder Zufriedenheit ermöglicht.[153] Sehr beliebt ist diese Test-Methode für das Testen der Benutzeroberfläche (engl. „graphical user interface", GUI) bei Webanwendungen.[154]

Ein Usability-Test besteht im Allgemeinen aus 4 bis 25 Testsitzungen. Bei einer Testsitzung erfüllt ein einzelner Testteilnehmer die ihm gestellten, für die Benutzergruppe typischen Aufgaben. Für Erkenntnisse hinsichtlich von Problemen bei der Usability sind mindestens vier Testsitzungen nötig, für Aussagen hinsichtlich Effektivität, Effizienz und Zufriedenheit mindestens zwanzig.[155]

Hier wird ersichtlich, wie zeit- und mitunter kostenaufwendig (z.B. durch Rekrutieren und Aufwandsentschädigungen für Fachpersonal wie Ärzte, Piloten) Usability-Tests sind. Aus diesem Grund können vor den Usability-Tests ***Usability-Inspektionen*** sinnvoll sein. Bei einer Usability-Inspektion begutachten Experten für Benutzerschnittstellen (engl. „user interface", UI) und Nutzererfahrungen (engl. „user experience", UX) ein System hinsichtlich seiner Schwächen in Bezug auf Dialogprinzipien, Heuristiken, Gestaltungsregeln und Nutzeranforderungen.[156]

[153] Vgl. Geis/Tesch (2019), S. 170
[154] Witte (2019), S. 168 ff.
[155] Vgl. Geis/Tesch (2019), S. 175 f.
[156] Vgl. Geis/Tesch (2019), S. 170

2.4 Vorgehensmodelle in der Softwareentwicklung

2.4.1 Grundlagen und Motivation

Wie der Name schon sagt, beschreibt ein Vorgehensmodell das Vorgehen, um eine Art von Problemen bzw. Aufgaben adäquat und wiederholbar lösen zu können. Die vorgegebene Struktur soll dabei einen Rahmen schaffen, um für Stabilität bei der Projektplanung und -durchführung zu sorgen. Es werden darin Rollen für Aufgaben festgelegt, sowie teilweise auch Methoden, wie Aufgaben zu erledigen sind. Es verbindet folglich die Projektorganisation (Management) und die Projektumsetzung (Technik) miteinander.[157]

Vorgehensmodelle helfen bei der Verringerung der Komplexität durch das Aufbrechen des Projekts in einzelne, leichter beherrschbare Teile. Sie vereinfachen die Kommunikation durch Zuteilung von Rollen, was wiederum zu klaren Ansprechpartnern führt. Durch die Einführung von Begrifflichkeiten wird die Ausdrucksweise innerhalb von Projekten klarer und es kommt damit zu weniger Missverständnissen. Zudem machen sie Projekte vergleichbar, was in Folge für eine Steigerung der Qualität durch die Verbesserungsmöglichkeiten des Modells sorgt.[158]

Für diese Arbeit ist ein grundlegendes Verständnis über unterschiedliche Arten von Vorgehensmethoden nötig. Der Fokus liegt dabei nicht auf der Umsetzung bzw. Anwendung dieser Methoden, sondern ihrer Bestandteile (= Phasen).

2.4.2 Sequenzielles Vorgehen – Phasenmodelle

Bei sequenziellem Vorgehen werden die einzelnen Abschnitte (= Phasen) eines Projekts klar definiert und der Reihe nach (= sequenziell) abgearbeitet. Es findet eine Abgrenzung statt, wann und mit welchen Ergebnissen ein Abschnitt beendet und der nächste gestartet werden darf.[159]

Ein solcher Übergang wird entweder als „Meilenstein“ oder zunehmend auch als „Gate“ (dt. „Tor“, „Pforte“, …) bezeichnet. Wobei letzteres verdeutlichen soll, dass dieser Übergang mit einer bewussten Entscheidung des Projektmanagements gewährt wird.[160]

Die ***Vorteile*** von Phasenmodellen liegen in der einfachen Struktur und der damit verbundenen Übersichtlichkeit und Kontrolle. Zumindest solange die Lösung klar definiert werden kann und das Projekt an sich vergleichsweise risikoarm ist.[161]

Die Einschränkung der Vorteile lässt schon den Rückschluss auf die ***Nachteile*** zu. Da bei diesem Vorgehen erst zu einem späten Zeitpunkt ein Produkt bereitgestellt werden kann, führt dies auch erst spät zu Erkenntnissen. Die Folge sind aufwendige und damit teure Korrekturen.[162] Zudem

[157] Vgl. Aichele/Schönberger (2014), S. 29; Brandt-Pook/Kollmeier (2020), S. 4; Broy/Kuhrmann (2013), S. 86; Broy/Kuhrmann (2021), S. 84
[158] Vgl. Brandt-Pook/Kollmeier (2020), S. 5; Broy/Kuhrmann (2021), S. 84
[159] Vgl. Broy/Kuhrmann (2021), S. 86 f.
[160] Vgl. Madauss (2020), S. 115
[161] Vgl. Broy/Kuhrmann (2013), S. 90; Broy/Kuhrmann (2021), S. 94
[162] Vgl. Mayr (2005), S. 82 f.; Rau/Schuster (2021), S. 7

erschwert diese Kombination aus späten Erkenntnissen und mangelnder Flexibilität auch das Erkennen und Reagieren auf Planungsfehler oder Risiken.[163]

Wasserfallmodell

Der berühmteste Vertreter dieses Bereichs ist das Wasserfallmodell. Es handelt sich dabei um kein rein sequenzielles Vorgehen, da eine Rückkoppelung der einzelnen benachbarten Phasen durchaus zulässig ist.[164] Die grundlegenden Abschnitte des Wasserfallmodells sind, wie in Abbildung 6 zu sehen ist: Zielerforschung, Anforderungserhebung, Grobkonzept, Feinkonzept, Implementierung, Integration, Einführung und Betrieb, sowie den jeweils dazugehörenden Qualitätsprüfungen.[165]

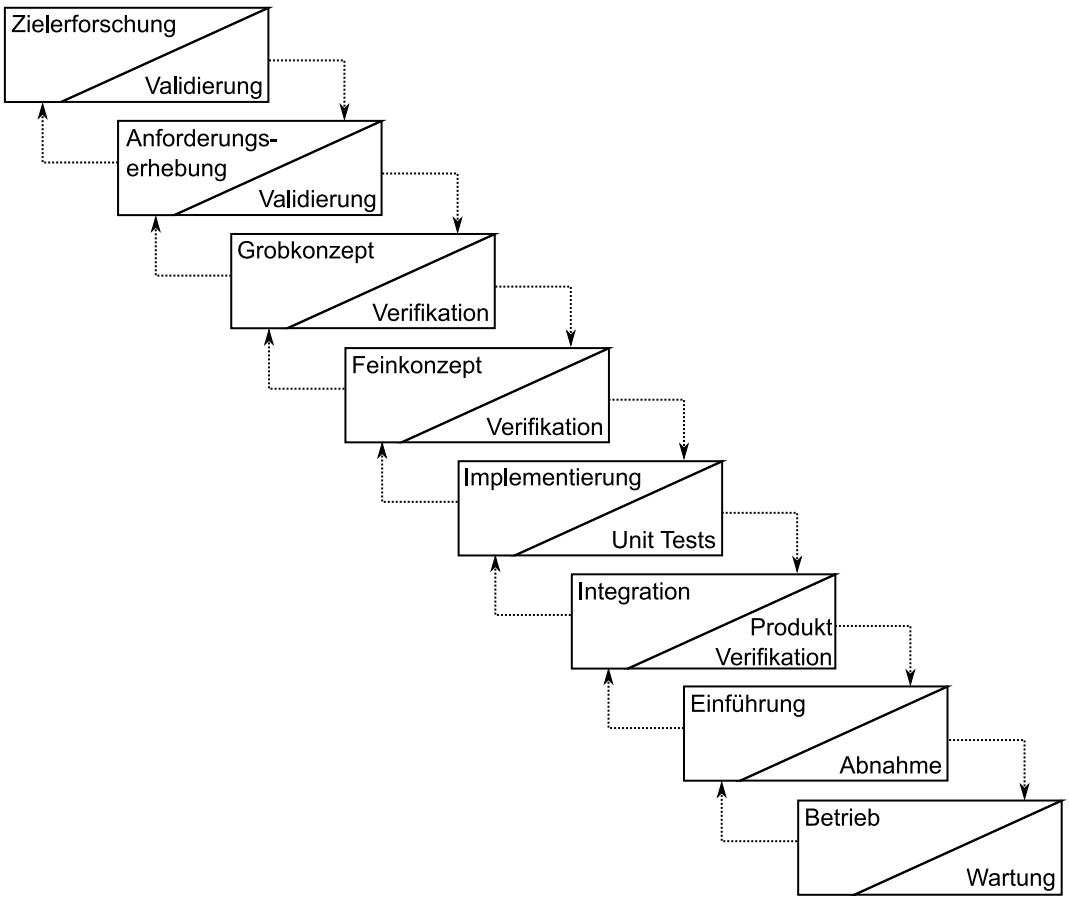

Abbildung 6: Wasserfallmodell[166]

[163] Vgl. Broy/Kuhrmann (2013), S. 90 f.
[164] Vgl. Broy/Kuhrmann (2013), S. 89
[165] Vgl. Boehm (1988), S. 62 f.
[166] Quelle: In Anlehnung an Boehm (1988), S. 62 und Mayr (2005), S. 84

2.4.3 Iterativ-inkrementelles Vorgehen – Spiralmodell

Der Vollständigkeit halber sei erwähnt, dass in der Literatur zwischen inkrementellen, iterativen und iterativ-inkrementellen Methoden unterschieden wird.[167] Bei ***inkrementellem Vorgehen*** wird ein Grundsystem sukzessive zu einem fertigen Produkt erweitert. Bei ***iterativem Vorgehen*** werden die immer gleichen standardisierten Aktivitäten bei jeder Iteration durchlaufen. In der Praxis werden die beiden Vorgehen üblicherweise miteinander kombiniert.[168] Aus diesem Grund ist eine Abgrenzung im Zuge dieser Arbeit nicht nötig und wird daher vernachlässigt.

Zusammengefasst findet das ***iterativ-inkrementelle Vorgehen*** somit in Zyklen (iterativ) statt, wobei jeder neue Zyklus auf den Ergebnissen des vorherigen aufbaut (inkrementell).[169] Werden diese (Teil-)Ergebnisse als Prototypen aufgefasst, die sukzessiv verbessert und erweitert werden, so kann der Schluss gezogen werden, dass auch das ***evolutionäre Prototyping*** ein iterativ-inkrementelles Vorgehen ist.[170]

Zu den ***Vorteilen*** dieses Vorgehens zählen unter anderem das schnelle Feedback aufgrund der gelieferten Zwischenprodukte. Missverständnisse, sowie falsche oder gar fehlende Anforderungen können so bereits sehr früh entdeckt werden. Eng verwoben damit ist auch die Möglichkeit dieses Feedback in Form von neuen oder geänderten Anforderungen, sowie geänderten Prioritäten dieser, in die Entwicklung einfließen zu lassen. In Folge wird dadurch auch das Gesamtrisiko des Projekts vermindert. Zudem führt das frühe und enge Einbinden der Anwender zu einer Erhöhung der Akzeptanz der entwickelten Software. Weiters muss der Gesamtumfang des Projekts nicht von vorhinein klar sein, da bereits Zwischenprodukte stabilisiert und zu Produkten ausgebaut werden können.[171]

Als ***Nachteil*** ist die Gefahr anzuführen, dass die präsentierten Prototypen (Zwischenergebnisse) zu Fehleinschätzungen bei Auftraggebern und/oder dem Management führen. Dies kann zu einem falsch-negativen Eindruck führen. Zum Beispiel durch bereits implementierte Funktionalität, die im Prototyp aber noch nicht sichtbar ist und somit scheinbar fehlt. Jedoch kann auch ein falsch-positiver Eindruck erweckt werden. Zum Beispiel durch exemplarisch umgesetzte Funktionen, die den Eindruck einer bereits weitgehenden Lösung vermitteln. Besonders der letztere Fall kann aufgrund von Druck seitens Kunden und/oder Management dazu führen, dass letztlich ein Prototyp statt eines geprüften Produkts geliefert wird.[172]

Spiralmodell

Der bekannteste Vertreter dieses Vorgehens ist das Spiralmodell nach Boehm[173] (siehe Abbildung 7). Dieses sieht in jeder Iteration die Phasen Zielerforschung, Evaluierung, Umsetzung und Planung der nächsten Iteration vor. Bei der Zielerforschung werden Ziele, die Möglichkeiten zur Erreichung sowie die geltenden Restriktionen erarbeitet. In der Evaluierungsphase werden Risiken

[167] Vgl. Mayr (2005), S. 83 ff.
[168] Vgl. Broy/Kuhrmann (2021), S. 89
[169] Vgl. Mayr (2005), S. 90
[170] Vgl. Mayr (2005), S. 90 ff.
[171] Vgl. Broy/Kuhrmann (2021), S. 93; Mayr (2005), S. 92; Rau/Schuster (2021), S. 8
[172] Vgl. Broy/Kuhrmann (2021), S. 93; Mayr (2005), S. 92
[173] Boehm (1988)

bestimmt und bewertet. Ggf. werden Machbarkeit, Leistungsfähigkeit und Sinnhaftigkeit von Lösungen mittels Prototypen getestet. Die eigentliche Umsetzung erfolgt anschließend nach dem Wasserfallmodel (Entwicklung, Überprüfung) in der Umsetzungsphase. Abschließend wird die nächste Iteration vorbereitet.[174]

Besonders die Risikobeherrschung bildet einen elementaren Teil dieses Modells. Ergibt sich bei der Risikoanalyse eines Zyklus ein erhöhtes Risiko, wird versucht dies mittels Erstellung und Bewertung von Prototypen zu minimieren.[175]

Abbildung 7: Spiralmodell[176]

[174] Vgl. Boehm (1988), S. 64 f.
[175] Vgl. Boehm (1988), S. 65; Keller (2019), S. 33
[176] Quelle: In Anlehnung an Boehm (1988), S. 64 und Broy/Kuhrmann (2021), S. 91

2.4.4 Agiles Vorgehen

In der Literatur werden zudem oft agile „Vorgehensmodelle“ erwähnt.[177] Dabei wird zum einen das agile Manifest[178] erwähnt, welches (lediglich) die agilen Werte zum Ausdruck bringt. Zum anderen werden Beispiele wie „Scrum“[179] oder „Extreme Programming“[180] angeführt. Wobei es sich bei diesen Beispielen um Vorgehensmethoden für die Softwareentwicklung handelt, welche lediglich Teile der Umsetzung in einem Softwareprojekt beschreiben. Die vollständige Abhandlung von Softwareprojekten ist mit solchen Vorgehensmethoden nicht möglich.

So setzt ***Scrum*** zum Start bereits ein priorisiertes Backlog voraus.[181] Zudem repräsentiert der „Product Owner“ (dt. „Produktverantwortlicher“) die gesamte Außenwelt und somit alle Stakeholder und deren Interessen in einer Person. Durch ihn werden Anforderungen erstellt, priorisiert und letztlich auch abgenommen. Das Scrum-Team ist folglich außer Lage, die gegebenen Anforderungen selbst zu validieren. Lediglich eine Verifizierung aufgrund von gegebenen Akzeptanzkriterien ist möglich. Am Ende eines jeden Sprints erfolgt der „Sprint Review“ in welchem das (perfekte) Ergebnis präsentiert wird. Rollen oder Vorgehensweisen für Test- oder Qualitätsmanagement existieren nicht.[182]

Das Konzept der ***DevOps*** – einem englischen Kunstwort aus „Development“ (dt. „Entwicklung“) und „Operations“ (dt. „Betrieb“) – ist ebenfalls kein vollständiges Vorgehensmodell. Befähigt durch agile Methoden, können Softwareentwickler binnen weniger Wochen neue Versionen von Software fertigen. Dem gegenüber steht der Betrieb, der tendenziell Änderungen in der Software aus Gründen der Stabilität kritisch eingestellt ist, da jede Änderung das Risiko für Fehler beinhaltet. Mit dem Konzept der DevOps wird vorrangig über Qualitätssicherungsmaßnahmen versucht diesen Zielkonflikt zu lösen. Dabei arbeiten die Bereiche Entwicklung, Qualitätssicherung und Betrieb eng zusammen, mit dem Bestreben das Risiko negativer Auswirkungen durch neue Versionen einer Software möglichst gering zu halten.[183]

2.4.5 Kombination von Vorgehensmodellen und -methoden

So unterschiedlich wie Software ist, so unterschiedlich sind auch die Vorgehensmodelle bzw. -methoden. Dabei ist es wichtig zu beachten, dass sich Vorgehensmethoden nicht zwangsweise ausschließen. Je nachdem welche Phase der Entwicklung eine Vorgehensmethode adressiert, kann sie in den verschiedenen Vorgehensmodellen mit anderen Vorgehensmethoden kombiniert werden. In der Literatur wird explizit darauf hingewiesen, dass gerade diese ***hybriden Ansätze*** zum Erfolg führen.[184]

---

[177] Vgl. Aichele/Schönberger (2014), S. 37 ff.; Broy/Kuhrmann (2021), S. 94 ff.; Rau/Schuster (2021), S. 10 ff.
[178] Beck et al. (2001); Fowler/Highsmith (2001)
[179] Weiterführende Literatur: Schwaber/Sutherland (2013)
[180] Weiterführende Literatur: Beck (2005); Beck (2012)
[181] Vgl. Rau/Schuster (2021), S. 13; Schwaber/Sutherland (2013), S. 12 f.
[182] Vgl. Brandes/Heller (2017), S. 2 f.
[183] Vgl. Alt et al. (2017), S. 24; Brandes/Heller (2017), S. 19 f.; Broy/Kuhrmann (2021), S. 519 f.
[184] Vgl. Kuhrmann et al. (2017), S. 30; Mayr (2005), S. 82

Welches Vorgehensmodell für ein Projekt gewählt wird und mit welchen Vorgehensmethoden dies ergänzt wird, muss nicht von Anfang an klar sein. Auch hier hat sich eine eigene gewisse Beweglichkeit während der Projektdurchführung bewährt. Je nach Projektverlauf ist es sinnvoll, das zugrundeliegende Vorgehensmodell anzupassen. Bewährt hat sich vor allem ein Mix aus traditionellem Vorgehensmodell und agilen Vorgehensmethoden.[185]

So wird beispielsweise in der Praxis das Wasserfallmodell gerne mit Scrum verbunden.[186] Ein Teil der Planung findet nach dem Wasserfallmodell statt. Dies ermöglicht den groben Rahmen festzulegen, wie er in vielen Unternehmen für das Controlling nötig ist. Jedoch bereits mit dem Bewusstsein, dass sich Änderungen ergeben werden. In der Implementierungsphase wird dann die Vorgehensmethode Scrum angewendet. Dies bringt die nötige Flexibilität in die Projektphase, um für adäquate Ergebnisse zu sorgen.[187]

Zudem haben Studien gezeigt, dass die Kombination von verschiedenen Vorgehensmodellen und -methoden bereits weit verbreitete Praxis ist.[188]

[185] Vgl. Broy/Kuhrmann (2021), S. 86
[186] Vgl. Tell et al. (2019), S. 109
[187] Vgl. West et al. (2011), S. 10
[188] Vgl. Kuhrmann et al. (2017); Tell et al. (2019)

2.5 Risikobasiertes Testen

2.5.1 Definition: Risiko im Kontext der Softwareentwicklung

Für das risikobasierte Testen ist ein grundlegendes Verständnis von Risiko im Kontext der Softwareentwicklung nötig. In IEEE 829-2008 wird Risiko definiert als:

> *„**(A)** The combination of the probability of occurrence and the consequences of a given future undesirable event. Risk can be associated with software and/or systems. **(B)** The combination of the probability of an abnormal event or failure and the consequence(s) of that event or failure to a system's components, operators, users, or environment.“* [189]

Risiko ist demnach eine Kombination aus den zwei Faktoren ***Eintrittswahrscheinlichkeit*** und ***Auswirkung*** (üblicherweise Schadenshöhe) eines möglichen Ereignisses.

Während die Risikobewertung im Bereich von Versicherungen und Banken aufgrund von „großen Zahlen“ (Erfahrungswerte aus der Vergangenheit) erfolgt, fehlt diese Bewertungsgrundlange im Bereich des Softwareprojektmanagements. So gibt es für eine spezifische Software in der Entwicklung keine historischen Daten, auf denen sich eine Prognose für die Zukunft ableiten ließe.[190]

Folglich sind Auswertungen lediglich über Kennzahlen (z.B. Metriken, Vergangenheitswerte ähnlicher Projekte) möglich, jedoch keine vollständige statistische Auswertung. Bei Schätzungen (auch durch technische Experten) unterliegt der Mensch zudem oft kognitiven Verzerrungen[191], was in Folge zu ungenauen oder falschen Einschätzungen führt.[192]

2.5.2 Definition: Risikobasiertes Testen

Risikobasiertes Testen wird in ISO/IEC/IEEE 29119-2:2013 wie folgt definiert:

> *„testing […] in which the management, selection, prioritisation, and use of testing activities and resources is consciously based on corresponding types and levels of analyzed risk“* [193]

Das risikobasierte Testen ist demnach ein Konzept, bei welchem zuvor bestimmte und bewertete ***Risiken die Grundlage für Entscheidungen*** in allen Phasen ***des Testprozesses*** bilden. Dazu gehören unter anderem die Auswahl und Priorisierung der zu testenden Objekte (Module, Komponenten, Systeme, Dokumentationen, …) sowie die darauf anzuwendenden Teststrategien.[194] Durch die mit Softwareprojekten einhergehende flexible Anpassung und Priorisierung von Anforderungen ergibt sich, dass wiederkehrend auf neue Risiken geprüft und bestehende Risiken neu

---

[189] IEEE 829-2008 (2008), S. 10
[190] Vgl. Redmill (2004), S. 6
[191] Weitere Literatur: Glaser (2019); Kahneman (2016); Kahneman et al. (2021)
[192] Vgl. Redmill (2004), S. 4 f.
[193] ISO/IEC/IEEE 29119-2:2021 (2021), S. 3
[194] Vgl. Foidl/Felderer (2018), S. 809

bewertet werden müssen. Dies kann in Folge dazu führen, dass Teststrategien angepasst werden müssen.[195]

Ziel dieses Vorgehens ist es, angesichts des Spannungsfelds von Anforderungen und Testressourcen, möglichst effektiv zu testen. Durch die Risikobewertung werden die fehleranfälligsten (Eintrittswahrscheinlichkeit) und die erfolgskritischsten (Schadenshöhe) Teile sichtbar gemacht. Infolgedessen wird genau diesen Bereichen besondere Aufmerksamkeit beim Testen geschenkt.[196]

Es drängt sich an dieser Stelle die Frage auf: Ist Testen nicht allgemein durch Risiken motiviert? Risiken sind potenzielle Probleme. Probleme, die Anwendern, Administratoren u.v.a. erspart bleiben sollen. James Bach beantwortet diese Frage mit einer Analogie zu Lebensmitteln: Alle Menschen müssen Essen. Allerdings leben diese unter normalen Umständen nicht von Mahlzeit zu Mahlzeit. Sie schenken ihrem Essen nur dann besondere Aufmerksamkeit, wenn sie an Allergien oder Unverträglichkeiten leiden, wenn sie dazu neigen zu viel zu essen oder wenn sie Gefahr laufen, dass ihnen die Lebensmittel ausgehen. Also nur unter besonderen Umständen wird das Leben um das Essen herumgeplant. Dasselbe gilt für die Betrachtung von Risiken beim Testen.[197]

2.5.3 Grundlegende Vorgehensweise

In der Literatur hat sich hinsichtlich des risikobasierten Testens keine einheitliche Vorgehensweise herauskristallisiert. Jedoch gibt es Überschneidungen bezüglich der Tätigkeiten und den damit verbundenen Ergebnissen. Im Folgenden werden daher diese Tätigkeiten, deren Artefakte sowie die Zusammenhänge grundlegend beschrieben. Ebenfalls sei an dieser Stelle auf die Abschnitte 2.5.4 „*Risikoidentifikation und -bewertung*“ und 2.5.5 „*Testen*“ verwiesen, in welchen die unten angeschnittenen Bereich vertieft erörtert werden, da sie zentral für das risikobasierte Testen sind.

Das elementarste Artefakt ist das ***Risikoobjekt*** (engl. „risk item“), welches teilweise nur mit Risiko (engl. „risk“) bezeichnet wird. Generell wird unter einem Risikoobjekt alles verstanden, was einer Risikobewertung unterzogen wird. Dies kann zum Beispiel eine Anforderung, eine Komponente oder ein unbedingt zu vermeidender Fehler sein.[198]

Das zweite wichtige Artefakt sind die ***Testfälle***, die während des Testprozesses ausgeführt werden. Aufgrund dieser findet die eigentliche Risikominimierung statt. In der Anwendung des risikobasierten Testens sind verschiedene Ansätze zu erkennen. Jedoch ist allen gemeinsam, dass die Testfälle mit den Risikoobjekten verknüpft werden müssen.[199]

Ganz allgemein wird somit bei risikobasiertem Testen ein Risikoobjekt mit Testfällen verbunden. In der Regel ist ein Testfall wiederum mit einer Risikobewertung beurteilt, die von der Risikobewertung des Risikoobjekts abgeleitet wird. Mittels dieser Risikobewertungen können Risikoklassen

---

[195] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 17 f.
[196] Vgl. Alam/Khan (2013), S. 34; Dahiya et al. (2020), S. 200; Kloos et al. (2011), S. 28; Ottevanger (1999), S. 1
[197] Vgl. Bach (1999), S. 2
[198] Vgl. Foidl/Felderer (2018), S. 811 f.
[199] Vgl. Ramler/Felderer (2015), S. 358

gebildet werden. Diese dienen dazu, Risikoobjekte und/oder Testfälle zu vergleichen und dadurch den Einsatz von Ressourcen zu bestimmen.[200]

2.5.4 Risikoidentifikation und -bewertung

Risiken bestehen in der Regel aus einem oder einer Kombination von Ereignissen und den damit verbundenen Folgen (üblicherweise Schäden). In einem ersten Schritt gilt es die Risiken aufzudecken (= Risikoidentifikation). Anschließend wird jedes Risiko für sich bewertet. Üblicherweise werden dafür die Schadenshöhe und die Eintrittswahrscheinlichkeit einzeln bestimmt und zum Risikowert kombiniert. Beide Werte können sowohl formal auf Grundlage eines Modells oder informell auf Grundlage von Schätzungen bestimmt werden. Zudem können sowohl quantitative wie auch qualitative Skalen verwendet werden.[201] Alternativ zur Bewertung über die zwei Faktoren Schadenshöhe und Eintrittswahrscheinlichkeit, kann die Risikobewertung auch direkt erfolgen.[202]

Im Zusammenhang mit der Softwareentwicklung sind folgende drei Arten von Risiken vorhanden. Dem ***Produktrisiko*** gilt das Hauptaugenmerk. Im Vordergrund steht die Erfüllung der Anforderungen und die damit verbundenen Qualitätsmerkmale des Produkts. Das ***Projektrisiko*** wird vorrangig von der Verfügbarkeit von Ressourcen wie Fachkräften, Zulieferern, Budget und Terminen bestimmt. Eng damit verbunden ist das ***Prozessrisiko***, das sich auf die internen Abläufe des Projekts bezieht. Darunter fällt das gesamte Projektmanagement (Planung, Überwachung, Kontrolle, Dokumentation, …) und die damit verbundene Flexibilität aufgrund von Erkenntnissen oder sich ändernder Rahmenbedingungen.[203]

Als Hilfsmittel für die ***Risikoidentifikation*** können verschiedene Ansätze gewählt werden. Der Blick von innen nach außen beschäftigt sich mit der Frage „Was kann hier fehlschlagen?“ aus der Sicht des Produkts.[204] Dem gegenüber steht der Blick von außen nach innen, bei welchem potenzielle Risiken auf deren Relevanz für die aktuelle Software geprüft werden.[205] Beispiele für solche Risiken sind zum einen Qualitätskriterien (siehe 2.2.3 „*Qualitätsmerkmale nach den ISO*/IEC“) oder generische Risiko-Listen, welche den Projektmitgliedern eine Diskussionsgrundlage bieten.

Beispiele für die Punkte einer solchen Risiko-Liste sind:[206]

- Komplex: alles, was unverhältnismäßig groß, kompliziert oder verworren ist
- Neu: alles, was noch keine Geschichte im Produkt hat
- Geändert: alles, was manipuliert oder "verbessert" wurde
- Vorgelagerte Abhängigkeit: alles, dessen Ausfall zu einem kaskadenartigen Ausfall im übrigen System führt
- Nachgelagerte Abhängigkeit: alles, was besonders empfindlich auf Ausfälle im übrigen System reagiert

[200] Vgl. Felderer et al. (2015), S. 34
[201] Vgl. Großmann et al. (2020), S. 42
[202] Vgl. Felderer et al. (2015), S. 38
[203] Vgl. Yoon/Choi (2011), S. 193
[204] Vgl. Bach (1999), S. 3
[205] Vgl. Bach (1999), S. 5
[206] Vgl. Bach (1999), S. 5 f.

- Kritisch: alles, dessen Ausfall erhebliche Schäden verursachen könnte
- Präzise: alles, was seine Anforderungen genau erfüllen muss
- Populär: alles, was häufig verwendet wird
- Strategisch: alles, was von besonderer Bedeutung ist
- Drittanbieter: alles Zugekaufte
- Verteilt: alles, was zeitlich oder räumlich verteilt ist, aber dessen Elemente zusammenarbeiten müssen
- Fehlerhaft: alles, was für Probleme bekannt ist
- Kürzlich gescheitert: alles, was in der jüngsten Vergangenheit gescheitert ist

Als Grundlage können zudem auch Systeme wie Anforderungs- oder Defekt-Management-Systeme dienen, da aus dokumentierten Abweichungen oder Defekten Risiken abgeleitet werden können.[207]

Nach der Analyse des Systems auf mögliche Risiken, müssen diese entsprechend bewertet werden. Für die ***Risikobewertung*** werden hierfür üblicherweise die zwei Parameter Schadenshöhe und Eintrittswahrscheinlichkeit zum Risikowert kombiniert. Dies kann auf zweierlei Arten erfolgen. Zum einen kann der Risikowert (R, engl. „risk“) über die Multiplikation der Eintrittswahrscheinlichkeit (P, engl. „probability“) mit der Schadenshöhe (I, engl. „impact“) berechnet werden ($R = P \times I$). Diese Aggregation der Information ermöglicht die lineare Priorisierung der Risiken sowie eine einfache Gruppierung. Zum zweiten kann der Risikowert als Dupel aus den zwei Werten Schadenshöhe und Eintrittswahrscheinlichkeit gesehen werden ($R = [P, I]$). Dies ermöglicht die Einteilung in eine entsprechende Risikomatrix.[208] Die Abbildung 8 (a) zeigt zur Veranschaulichung eine Tabelle mit sechs Risikoobjekten, der dazugehörenden Risikoeinschätzung, den errechneten Risikowerten sowie der zugeordneten Klassen laut Risikomatrix. Die dazugehörende Risikomatrix ist in Abbildung 8 (b) dargestellt.

[207] Vgl. Ramler/Felderer (2015), S. 358
[208] Vgl. Ramler/Felderer (2015), S. 359 ff.

(a)

Risikoobjekte

	P	I	R	Klasse
A	10	20	200	I
B	30	45	1350	I
C	75	30	2250	II
D	5	90	450	III
E	25	75	1875	III
F	70	80	5600	IV

(b)

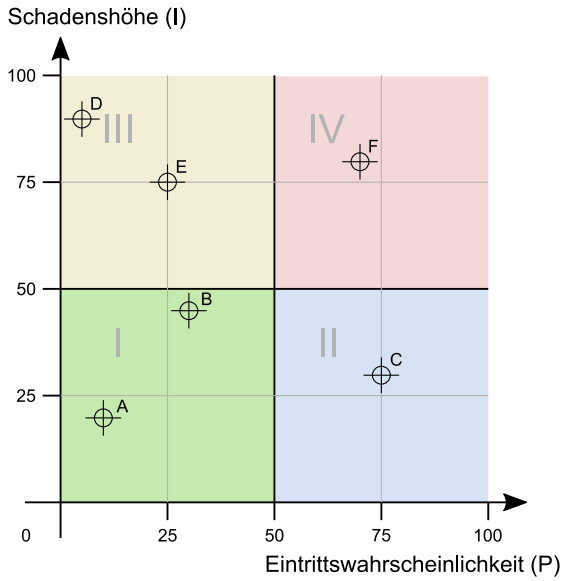

Abbildung 8: Risikobewertung mittels Multiplikation und als Dupel in einer Risikomatrix[209]

Besonders in frühen Phasen eines Softwareprojekts können fehlende Daten dazu führen, dass die Eintrittswahrscheinlichkeit nicht ermittelt werden kann. In diesem Fall empfiehlt Redmill die Risikobewertung in einem ersten Schritt nur aufgrund der Schadenshöhe durchzuführen.[210]

Die ***Schadenshöhe*** wird durch wirtschaftliche Kriterien bestimmt.[211] Dies sind Kriterien, die entweder die Kundenbedürfnisse und damit die dazugehörigen Anforderungen betreffen oder den Unternehmenswert.[212] In der Folge bedeutet dies, dass ohne wirtschaftliche Konsequenzen kein Schaden vorliegt (Schadenshöhe = 0).[213]

Es muss jedoch beachtet werden, dass die Auswirkungen für unterschiedliche Stakeholder unterschiedlich hoch ausfallen können. Hierbei sind auch etwaige Opportunitätskosten mit einzubeziehen.[214]

Anzumerken ist zudem, dass nicht jeder Schaden direkt mit Geld bemessen werden kann. So sind Reputationsverlust, Umweltschäden oder gar Menschenleben ohne teils aufwendige Versicherungsmathematik nur schwer bezifferbar. Ähnlich verhält es sich mit Kosten für Entschädigungen oder Rechtskosten. Diese mangelnde Vergleichbarkeit kann bei der Risikobewertung ein Problem darstellen.[215] Eine Möglichkeit, diesem Umstand zu begegnen, ist die Verwendung von qualitativen Skalen (z.B. Klassifizierung in gering, mittel und hoch).[216]

---

[209] Quelle: Verfasser
[210] Vgl. Redmill (2005), S. 4 ff.
[211] Vgl. Felderer/Ramler (2014), S. 552; Veenendaal (2011), S. 13
[212] Vgl. Felderer/Ramler (2014), S. 553
[213] Vgl. Amland (2000), S. 288
[214] Vgl. Redmill (2005), S. 4
[215] Vgl. Redmill (2004), S. 11
[216] Vgl. Amland (2000), S. 290

Die ***Eintrittswahrscheinlichkeit*** wird durch technische Kriterien bestimmt.[217] Dazu gehören vor allem die Artefakte der Softwareentwicklung, im Speziellen die kompilierte Software und deren Quellcode. Die Bewertung selbst kann entweder durch entsprechendes Personal wie Softwarearchitekten, -entwickler und/oder -tester erfolgen, oder über technische Verfahren wie Fehlerbaumanalysen und/oder Code-Metriken.[218] Amland kommt unter anderem zum Ergebnis, dass im Durchschnitt die zehn Funktionen mit der höchsten Fehlerrate, eine doppelt so hohe zyklomatische Komplexität (Metrik nach McCabe[219]) aufweisen.[220] In diesem Bezug weist Rosenberg darauf hin, dass für die Erkennung von möglichen Problemen stets mehr als eine Code-Metrik eingesetzt werden müsse.[221]

2.5.5 Testen

Hinsichtlich des Testens sind in der Literatur ***kaum Ansätze vorhanden***. So wird in der Literatur selbst bemängelt, dass sich das risikobasierte Testen in vielen Fällen nur auf die Risikopriorisierung beschränkt. In anderen Fällen wird zwar eine Test- bzw. Testfall-Priorisierung aufgrund der Risikobewertung propagiert, allerdings entweder nur grobe Richtlinien für die Ableitung der Tests geliefert oder bereits von existierenden Tests ausgegangen.[222]

Lediglich zwei grundlegende Ideen zum risikobasierten Testen sind in der Literatur hinreichend beschrieben: Die „Testfall-Priorisierung mittels Aktivitätsdiagrammen“ und die „Teststrategie aufgrund der Risikoklassifizierung“. Jedoch wurden beide Ansätze von verschiedenen Autoren aufgegriffen und weiterentwickelt. Aus diesem Grund werden diese Konzepte in den Ergebnissen unter Abschnitt 4.5 „*Phase: Risikobasiertes Testen*“ vorgestellt.

---

[217] Vgl. Felderer/Ramler (2014), S. 552; Veenendaal (2011), S. 13

[218] Vgl. Amland (2000), S. 289; Felderer/Ramler (2014), S. 553; Kloos et al. (2011), S. 27; Rosenberg et al. (1999), S. 5

[219] McCabe (1976)

[220] Vgl. Amland (2000), S. 294

[221] Vgl. Rosenberg et al. (1999), S. 4

[222] Vgl. Bauer et al. (2008), S. 101; Felderer et al. (2012), S. 163; Felderer/Ramler (2014), S. 551; Stallbaum et al. (2008), S. 67

3 Methode

Die im ersten Schritt vollzogene Literaturrecherche mit Stichwörtern bzw. Stichwortkombinationen aus

- „Risiko“ (engl. „risk“)
- „risikobasiert“ (engl. „risk-based“)
- „Testen“ (engl. „test“)
- „Ansatz“ (engl. „approach“)
- „Software“
- „Entwicklung“ (engl. „development“)

brachten nur wenige Publikationen zum Thema „risikobasiertes Testen“ zutage. Das Problem lag vor allem in den Suchbegriffen. Für sich allein sind diese zu generell, um entsprechend gezielte Ergebnisse zu liefern. In Kombinationen und je nach durchsuchter Datenbank wurden entweder generell sehr wenige Ergebnisse geliefert oder es wurden Treffer aus dem Bereich der Medizin, Versicherungen und Banken bzw. zu Software aus diesen Bereichen gelistet.

Daher wurde in einem zweiten Schritt gezielt nach den in der Literatur verwiesenen Publikationen sowie deren Autoren gesucht. Besonders die Suche nach den Autoren stellte sicher, dass sich die Suche nicht nur in die Vergangenheit richtete.

Zuletzt wurden Quellen (insb. Bücher, Normen und Qualitätsstandards) zum Thema Software, Softwareentwicklung und Softwarequalität gesucht, um die gefundenen Ergebnisse im Kontext des risikobasierten Testens einzuordnen.

Besonders Normen stellen den Stand der Technik dar und helfen durch entsprechende Definitionen bei ambivalenten oder mehrdeutigen Ausdrücken für Klarheit zu sorgen.

4 Ergebnisse

4.1 Heterogenität in der Literatur

Bei der Betrachtung der wissenschaftlichen Publikationen zu risikobasiertem Testen wird ersichtlich, wie viel Spielraum der Denkansatz in der Anwendung zulässt. Noch deutlicher werden die Unterschiede beim Vergleich der zugehörigen Herangehensweisen.[223] Die meisten Ansätze sind sehr generell gehalten und wurden als kaum direkt im praktischen Gebrauch umsetzbar bewertet.[224] Ein Hinweis, der diese Aussage noch weiter bestärkt, ist der Mangel an durchgeführten Fallstudien.[225] Der folgende Überblick der unterschiedlichen Herangehensweisen der einzelnen Bereiche des risikobasierten Testens dient als Grundlage für in dieser Arbeit herausgearbeitete Gemeinsamkeiten und die phasenweise Beschreibung der Maßnahmen im Verlauf des gesamten Entwicklungsmodells.

Die ***Risikoidentifikation*** kann sehr technisch und strukturiert erfolgen, indem das System in einzelne Teile zerlegt wird und anschließend diese hinsichtlich Risiken überprüft werden.[226] Allerdings sind auch generische Ansätze möglich, die an die verschiedenen Stakeholder angepasst werden. Das Spektrum reicht von der Erstellung bzw. dem Prüfen von Risiko-Checklisten, über Workshops bis hin zu Experteninterviews (Branche, Domäne, Technik, …).[227] Zusätzlich können Risiken aus Anforderungs- oder Projektmanagement-Tools abgeleitet werden. Bei ähnlichen, bereits im Einsatz befindlichen Softwareprodukten, kann zudem die Fehlerverwaltung und/oder das Ticketsystem des Service-Teams als Informationsquelle dienen.[228]

Bei der ***Risikobewertung*** unterscheiden sich die Ansätze in der Literatur ebenfalls erheblich. Eine Meta-Studie[229] aus dem Jahr 2015 über 17 Ansätze des risikobasierten Testens zeigt, dass ein breites Spektrum an Risikobewertungen verwendet wird. Die Ergebnisse wurden in Tabelle 1 zusammengefasst. Darin ist die unterschiedliche Klassifizierung bei der Bewertung sowie die Verbreitung der einzelnen Merkmale ersichtlich. Anzumerken ist, dass einige Ansätze mehrere Merkmale einer Klasse anwenden (in der Spalte „Merkmalskombination“ angeführt), sowie einige Ansätze bei einzelnen Klassen keine Ausprägung verwenden bzw. anführen.

Zusammengefasst schätzen die meisten in der Studie untersuchten Ansätze das Risiko auf Systemebene für funktionale Artefakte, berücksichtigen explizit Wahrscheinlichkeit und Auswirkung, verwenden eine quantitative Skala und basieren auf manueller Messung. Expertenurteile sind fast ebenso häufig erwähnt wie ein formales Schätzungsmodell. Dies gilt ebenfalls für initiale und iterative Messungen. Von einer Tool-Unterstützung für Risikoabschätzungen wird jedoch kaum berichtet.[230]

---

[223] Vgl. Großmann et al. (2020), S. 47
[224] Vgl. Erdogan et al. (2014), S. 640
[225] Vgl. Ramler/Felderer (2015), S. 368
[226] Vgl. Amland (2000), S. 289
[227] Vgl. Bach (1999), S. 3 ff.; Dahiya et al. (2020), S. 197; Redmill (2004), S. 11
[228] Vgl. Ramler/Felderer (2015), S. 358
[229] Felderer et al. (2015)
[230] Vgl. Felderer et al. (2015), S. 40

Bewertete Klasse	***Summe der Ansätze***	
	jeweiliges Merkmal	Merkmals-kombination
Typ des Risikoobjekts		
Entwicklungsergebnis (z.B. Anforderungen, Komponenten, Quellcode)	9	4
Generisches Risiko (z.B. Sicherheitsrisiko)	3	
Testfall	1	
Art der Risikobewertung		
Direkte Risikoeinschätzung	3	0
Kombination von Eintrittswahrscheinlichkeit mit Schadensausmaß	14	
Art der Kriterien		
Wirtschaftliche Kriterien (z.B. monetärer Verlust)	1	6
Technische Kriterien (z.B. Komplexität des Quellcodes)	8	
Keine Angaben	2	
Bewertungsmethode		
Expertenschätzung	6	3
Formale Methode	8	
Risikoeinteilung		
Quantitativ	11	0
Qualitativ (z.B. niedrig, mittel, hoch)	6	
Zeitpunkt der Risikobewertung		
Initial	9	0
Iterativ	8	
Automatisierung der Messung		
Manuelle Messung	12	3
Automatische Messung	2	
Unterstützung durch Tools		
Tabellenkalkulation	3	0
Spezielles Tool zur Risikobewertung	3	
Tool zur Unterstützung des Testmanagement	0	
Keine Angaben	11	

Tabelle 1: Merkmale zur Risikobewertung[231]

[231]Quelle: Zusammenfassung der Ergebnisse aus Felderer et al. (2015), S. 38

Eine weitere Meta-Studie[232] aus dem Jahr 2014, die sich generell mit Studien hinsichtlich Risikoanalyse und Softwaretests befasst, zeigt die ***Heterogenität der Ansätze*** noch deutlicher. Von den 24 untersuchten Ansätzen befassen sich[233]

- 22 ausschließlich mit verschiedenen Aspekten des risikobasierten Testens (nur 2 mit testbasierter Risikoanalyse (= TRB)),
- 4 mit risikobasiertem Testen auf allgemeiner Ebene,
- 2 explizit mit modellbasierter Risikoabschätzung,
- 5 besonders mit der Testfallgenerierung,
- 3 hauptsächlich mit der Testfallanalyse,
- 1 RBT- und 1 TBR-Ansatz mit Quellcodeanalyse,
- 2 mit Programmierparadigmen,
- 4 mit spezifischen Anwendungen und
- jeweils 1 RBT- und 1 TBR-Ansatz mit der Messung.

Zusammengefasst lässt sich bezüglich der grundlegenden Ideen ein Konsens in der Literatur erkennen. Bei Umfang und Umsetzung zeigen sich jedoch erhebliche Unterschiede. Wenn sich diese auch nicht widersprechen, werden im Detail doch eine Vielzahl an Möglichkeiten der konkreten Ausprägung aufgezeigt.

[232] Erdogan et al. (2014)
[233] Vgl. Erdogan et al. (2014), S. 640

4.2 Allgemeine Erkenntnisse (vom Dissens zum Konsens)

Bei der Literaturrecherche hat sich ebenfalls gezeigt, dass es sich bei risikobasiertem Testen um kein Verfahren im eigentlichen Sinne handelt. Vielmehr handelt es sich um eine Bandbreite von Ansätzen und Möglichkeiten das Testen von Software aufgrund von Überlegungen hinsichtlich Risikoaspekten zu gestalten bzw. zu optimieren.

Wie im vorherigen Abschnitt erklärt, gibt es hinsichtlich der einzelnen Tätigkeiten und deren Reihenfolge Überschneidungen, doch lässt sich kein einheitlicher Konsens erkennen. Für die Aufarbeitung der Ergebnisse wurden daher die wesentlichen Phasen bei der Anwendung des risikobasierten Testens herausgearbeitet (siehe unten). An diesen Phasen wird sich die weitere Struktur dieses Kapitels orientieren.

Es sei an dieser Stelle darauf hingewiesen, dass eine Phase nicht auf einen zeitlichen Rahmen begrenzt ist. Vielmehr handelt es sich um ein Tätigkeitsfeld, in welchem gewisse Aufgaben angesiedelt sind.[234] Konkret bedeutet das, dass sich in bestimmten Phasen gewisse Mitarbeiter um die mit der Phase verbundenen Tätigkeiten kümmern. Da agile Softwareentwicklung üblicherweise in iterativ-inkrementellen Zyklen abläuft, werden Phasen mehrfach und teilweise auch parallel durchlaufen.

Um die zu Beginn der Arbeit gestellten wissenschaftlichen Fragen

- „Wie kann risikobasiertes Testen in allen Phasen der Softwareentwicklung eingesetzt werden?“ und
- „Wie kann dabei der administrative Aufwand des risikobasierten Testens geringgehalten werden?“

zu beantworten, werden in diesem Kapitel verschiedene Denkansätze, Möglichkeiten und Strategien vorgestellt und kombiniert, mit welchen dies bewältigt werden kann. Dabei orientieren sich die Ergebnisse an den im Anschluss beschriebenen grundlegenden Phasen des risikobasierten Testens. Allgemeine und übergreifende Erkenntnisse werden am Ende dieses Kapitels angeführt. Zudem sei erwähnt, dass einiges des hier Vorgestelltem auch abseits des risikobasierten Testens angewendet werden kann. Da es allerdings explizit in der Literatur erwähnt wird, wird es als Teil der entsprechenden Ansätze behandelt.

Grundlegend erfolgt das risikobasierte Testen in folgenden Schritten. Wobei je nach Ansatz einzelne Schritt nicht vorkommen, aufgespalten oder zusammengefasst sind:[235]

- Zuerst erfolgt die ***Vorbereitung***. Es werden Pläne für das Testen (Strategien, Umgebung, Dokumentation, …) festgelegt, sowie die notwendigen Daten erhoben.
- Die erhobenen Daten dienen als Grundlage für die ***Risikoidentifizierung***.
- Durch die Ermittlung der jeweiligen Risikoindikatoren kann die ***Risikobewertung*** erfolgen. Je nach Ansatz kann dieser Schritt mehr oder weniger aufwendig ausfallen.

---

[234] Vgl. Broy/Kuhrmann (2013), S. 87 f.

[235] Vgl. Alam/Khan (2013), S. 34; Amland (2000), S. 289 ff.; Bach (1999), S. 2; Felderer et al. (2012), S. 163 ff.; ISO/IEC/IEEE 29119-2:2021 (2021), S. 17 ff.; Ottevanger (1999), S. 4 ff.; Veenendaal (2011), S. 11 ff.

- Die Risikobewertung bildet die Basis für die ***Maßnahmen des risikobasierten Testens***. Diese können zum Beispiel die Priorisierung bereits vorhandener Testfälle oder die Ausarbeitung eigener Teststrategien für einzelne Teile (Module, Komponenten, …) sein.
- Aufgrund der festgelegten Maßnahmen findet das ***optimierte Testen*** statt.
- Abschließend sind die ***Erkenntnisse auszuwerten***. Mit Hilfe dieser können die Risikoeinschätzung und die Testdurchführung geprüft und entsprechende Maßnahmen zur weiteren Verbesserung abgeleitet werden.

4.3 Phase: Vorbereitung und Planung

Grundlegende Erfolgsfaktoren für das risikobasierte Testen

Laut Bauer et al. sind zwei Dinge für die erfolgreiche Etablierung des risikobasierten Testens besonders wichtig:[236]

- Eine genaue und nachvollziehbare Risikoeinschätzung
- Ein Testplan, der die erhobenen Risiken adäquat abdeckt

Für beide Dinge müssen bereits zu Beginn des Projekts die entsprechenden Weichen gestellt werden. Wie Bach bereits 1999 anmerkte, ist das Testen an sich bereits eine Maßnahme zur Reduktion des Produktrisikos.[237] Beim Testen speziell auf Risiken zu achten, resultiert Erdogan et al. zufolge aus zwei Motivationen:[238]

- Verringerung des Ressourceneinsatzes
- Verbesserung der Qualität

Der Unterschied dieser zwei Motivationen liegt darin, dass im ersten Fall versucht wird, den Testaufwand zu verringern und dabei die Qualität möglichst beizubehalten. Im zweiten Fall wird versucht die Qualität zu steigern und dabei den Testaufwand möglichst beizubehalten. Diese Motivationen entsprechen einer Anpassung der Stellgrößen im magischen Dreieck unter der Prämisse das die gegebenen Ressourcen unverändert bleiben. In der Anwendung kann die Motivation allerdings vernachlässigt werden, da die zugrundeliegenden Überlegungen für beide Ziele dienlich sind. Trotzdem ist es laut Redmill notwendig, das Ziel bzw. die Erwartung durch die Einbeziehung von Risiken beim Testen zu definieren. Nach ihm könnten diese Ziele zum Beispiel sein, das Fehlerrisiko auf ein gewisses (zu spezifizierendes) Niveau zu reduzieren, das Ausfallrisiko zu minimieren oder Zeit- und Budgetvorgaben für einzelne Teile der Software einzuhalten.[239] Dies schafft ein einheitliches Verständnis innerhalb des Projekts.

Generell ist zu erwähnen, dass sich Risikoanalyse und Testplanung wechselseitig ergänzen. Nicht nur, dass aufgrund von erkannten Risiken das Testen optimiert werden kann. Aufgrund der gefundenen Fehler während des Testens, können auch Rückschlüsse auf erhöhte Risiken gezogen werden.[240]

Die allgemeinen Erfolgsfaktoren für die Durchführung von Softwareprojekten werden an dieser Stelle ausgespart. Ein Überblick wurde im Abschnitt 2.1 „*Die moderne Softwareentwicklung*“ gegeben. Allerdings soll hier trotzdem noch einmal darauf hingewiesen werden, dass besonders Softwareprojekte von den involvierten Stakeholdern und den erhobenen Anforderungen abhängig sind und somit besonderem Fokus bei der Planung bedürfen.

---

[236] Vgl. Bauer et al. (2008), S. 100
[237] Vgl. Bach (1999), S. 2
[238] Vgl. Erdogan et al. (2014), S. 640
[239] Vgl. Redmill (2004), S. 9
[240] Vgl. Erdogan et al. (2014), S. 627

Risikoplanung

Die Norm ISO/IEC/IEEE 29119-2[241] liefert hinsichtlich der Aufgaben der Risikoplanung bereits einen ersten Überblick.[242] Darin gefordert ist ein Punkt zur Risikoidentifikation und -analyse (org. „Identify and analyse risks (TP3)“)[243], welcher die folgenden Aufgaben umfasst:

- Bewertung von bekannten Risiken bzgl. Softwaretests
- Ergänzung neuer Risiken bzgl. Softwaretests
- Klassifizierung der Risiken
- Bewertung der Risiken (Auswirkung und Eintrittswahrscheinlichkeit)
- Zustimmung der Stakeholder zur Risikobewertung einholen
- Dokumentation der Ergebnisse der Risikobewertung

Sowie ein Punkt zur Identifikation der Risikobehandlung (org. „Identify risk treatment approaches (TP4)“)[244], in welchem die Tätigkeiten

- ermitteln der Risikobehandlung aufgrund von Art, Klassifizierung und Bewertung der Risiken und
- dokumentieren der eruierten Mittel zur Risikobehandlung

gefordert werden.

Testplanung

Großmann et al. merken an, dass der Risikoaspekt zur Optimierung von bereits bestehenden Testaktivitäten eingesetzt werden kann. So kann dies u.a. in die Bereiche Priorisierung, Auswahl und Ressourcenplanung einfließen.[245] Dazu ergänzt Amland, dass risikobasiertes Testen kein Problem löst, wenn das Testen ohne Prozess und komplett unstrukturiert erfolgt.[246]

Die Grundlage für eine Teststrategie bildet die Klarheit des gesamten Projektteams hinsichtlich der angestrebten Qualitätsziele und Vorgehensweise. Darauf aufbauend werden die Maßnahmen für das eigentliche Testen geplant. Die Herausforderung liegt zu Beginn des Projekts am begrenzten Wissen der Testplaner. Diese sind von den Anforderungen, Systemspezifikationen sowie Informationen der Softwareentwickler abhängig. Auch wenn diese Informationen grundsätzlich die Benutzersicht widerspiegeln, besteht die Gefahr des Übersehens von Qualitätsmerkmalen.[247] Es empfiehlt sich folglich während der Testplanung noch einmal explizit auf die Qualitätsmerkmal nach ISO/IEC 25010 bzw. ISO/IEC 25012 zu achten bzw. gegen diese zu prüfen.

An dieser Stelle kann bereits eine erste Qualitätssicherung erfolgen, da die Anforderungen die Grundlage für Testfälle bilden.[248] Es werden in einem ersten Schritt die Anforderungen an Entwicklungsobjekte gebunden, für welche entsprechende Qualitätskriterien abgeleitet wurden. Können

---

[241] ISO/IEC/IEEE 29119-2:2021 (2021)
[242] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 17 ff.
[243] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 20
[244] Vgl. ISO/IEC/IEEE 29119-2:2021 (2021), S. 21
[245] Vgl. Großmann et al. (2020), S. 42
[246] Vgl. Amland (2000), S. 293
[247] Vgl. Redmill (2004), S. 7
[248] Vgl. Droste/Merz (2019), S. 74; Witte (2019), S. 61

diese Testfälle nicht spezifiziert werden, weil Informationen fehlen oder widersprüchlich sind, weist dies auf Mängel in den Anforderungen hin.[249] Diese Mängel können somit frühzeitig behoben werden.

Hinsichtlich des Testens wird von Bach empfohlen, sich nicht nur auf risikobasiertes Testen zu stützen, sondern mindestens ein Viertel des Testaufwands auch in andere Ansätze zu investieren, da keine Heuristik immer funktioniere.[250] Ähnliches empfiehlt auch Veenendaal, der explizit darauf hinweist, nicht nur die wichtigsten Teile zu testen, sondern auch jene mit den meisten Fehlern.[251] Im Subtext beider Aussagen findet sich der Hinweis, immer auf die gesamte Software zu achten, statt zu stark auf einzelne Bereiche zu fokussieren.

Zudem muss stets bedacht werden, dass Tests nur die Spezifikation eines Programmes überprüft, und damit lediglich eine Verifizierung darstellt.[252] Daraus ergibt sich, dass auch Maßnahmen zur Validierung eingeplant werden müssen (z.B. Usability-Tests, siehe *2.3.4* „*Methoden zur Qualitätssicherung*“ Absatz „*Usability-Tests*“).

[249] Vgl. Broy/Kuhrmann (2021), S. 471; Droste/Merz (2019), S. 5
[250] Vgl. Bach (1999), S. 9
[251] Vgl. Veenendaal (2011), S. 2
[252] Vgl. Witte (2019), S. 12

4.4 Phase: Risikoidentifikation und -bewertung

Wie bereits in Abschnitt 4.1 „*Heterogenität in der Literatur*“ aufgezeigt, ist hinsichtlich Risikoidentifikation und -bewertung ein breites Spektrum an Möglichkeiten vorhanden. Jedoch lassen sich aufgrund der Literatur keine Präferenzen für ein bestimmtes Vorgehen bestimmen. Im Folgenden werden daher Punkte zur allgemeinen Beachtung bei der Risikoabschätzung aufgeführt.

Fundierte Informationen

Wichtig sind hinsichtlich der Risikoanalyse fundierte Informationen. Es bedarf korrekter und vollständiger Daten sowie einem guten Verständnis für Risiken, um diese zu interpretieren und die Schlüsse für das entsprechende Projekt zu ziehen.[253] Ramler et al. haben bei ihrer Fallstudie zum Einsatz von risikobasiertem Testen in fünf Unternehmen festgestellt, dass die Art und Weise, wie die Schätzungen durchgeführt wurden, in einem hohen Maße von den verfügbaren Informationen abhängig waren.[254]

Bedeutung der Stakeholder

Die ***verschiedenen Sichtweisen*** und Interessen der unterschiedlichen Stakeholder beim Identifizieren und darauffolgenden Bewerten der Risiken sind für eine umfassende Betrachtung essenziell. So ergeben sich Risiken durch den Einsatz einer Software nicht nur für den Anwender. Andere Gruppen wie Vertrieb, Service/Support, Management oder Entwicklung, um nur einige zu nennen, können durch einen Mangel an der Software ebenfalls Schaden erleiden.[255] Neben den funktionalen Risiken sind vor allem jene des Unternehmens zu berücksichtigen.[256] Veenendaal merkt dazu an, dass das Vergessen eines Stakeholders bedeutet, alle entsprechenden Risiken nicht zu erfassen.[257]

Ebenfalls zu beachten ist, dass Fehler bzw. nicht erfüllte Anforderungen auf verschiedene Stakeholder ***unterschiedliche Auswirkungen*** haben und somit eine unterschiedliche Schadenshöhe bedeuten.[258] Erschwerend kommt oft hinzu, dass keine verlässlichen und/oder objektiven Kriterien für die Schadenshöhe zur Verfügung stehen. Die Bewertungen obliegen in solchen Fällen den subjektiven Einschätzungen der befragten Personen. Es sei an dieser Stelle noch einmal auf den Hinweis von Redmill bzgl. der kognitiven Verzerrungen bei subjektiven Einschätzungen durch Menschen hingewiesen.[259] Um Einzelmeinungen zu vermeiden ist es daher anzuraten, mit mehreren Vertretern einer jeden Interessensgruppe zusammenzuarbeiten.

Insgesamt ist somit ein wesentlicher Erfolgsfaktor für eine zuverlässige Risikobewertung, bereits zu Beginn für die Zusammenarbeit mit ***qualifizierten Personen*** zu sorgen.[260] Erdogan gibt als Ziel

---

[253] Vgl. Redmill (2004), S. 12
[254] Vgl. Ramler/Felderer (2015), S. 359
[255] Vgl. Amland (2000), S. 289; Bach (1999), S. 9; Redmill (2004), S. 11 f.
[256] Vgl. Alam/Khan (2013), S. 35
[257] Vgl. Veenendaal (2011), S. 13
[258] Vgl. Redmill (2004), S. 6
[259] Vgl. Redmill (2004), S. 4 f.
[260] Vgl. Alam/Khan (2013), S. 35; Redmill (2004), S. 8

dieser umfänglichen Betrachtungsweise die Vermeidung von flüchtigem und/oder partiellem Vorgehen bei der Risikobestimmung an.[261]

Ergänzende Randbedingungen zu den Anforderungen

Obwohl den funktionalen Anforderungen gefolgt von den Qualitätsanforderungen die meiste Aufmerksamkeit geschenkt wird, sind mitunter zusätzliche Randbedingungen zu beachten. Daher sollten auch weitere Bereiche wie beispielsweise

- Rechtliche Rahmenbedingungen und regulatorische Stellen,
- IT-Sicherheit und
- die Softwareentwicklung an sich

einer umfassenden Analyse unterzogen werden, um implizite Anforderungen und Risiken zu identifizieren.

So können unberücksichtigte ***gesetzliche Anforderungen*** (z.B. Medizinproduktegesetz, Datenschutz-Grundverordnung) erhebliche Strafen bedeuten oder die Inverkehrbringung von Software gänzlich verhindern. Die Sicherstellung, dass die zu entwickelnde Software im Zweifelsfall diesen Anforderungen entspricht, ist für den Projekterfolg unumgänglich.

Im Bereich der IT-Sicherheit ist es empfehlenswert die ***Schutzanforderungen*** für das eigene System und dessen Daten zu ermitteln. Insbesondere auf die vier Dimensionen Verfügbarkeit, Datenexistenz, Integrität sowie Vertraulichkeit sollte geprüft werden.[262]

Ebenfalls wird laut Redmill der Bereich der ***Softwareentwicklung*** hinsichtlich Risiken meist zu stark vernachlässigt. Schlechte Softwarearchitektur, lockere Programmierung und unerfahrene Softwareentwickler sind nur einige beispielhafte Punkte, die erhebliche Risiken darstellen können.[263]

Risikoklassen

Aus den in der Literatur beschriebenen Möglichkeiten empfiehlt sich eine Vorgehensweise bei der Bestimmung des Risikos besonders. Ramler et al. empfehlen die Eintrittswahrscheinlichkeit und die Schadenshöhe als getrennte Dimensionen zu behandeln. Dies verhindert einen Informationsverlust durch die Aggregation der Werte.[264] Um diesen Informationsverlust bei der Berechnung des Risikowerts (R) durch die Multiplikation von Eintrittswahrscheinlichkeit (P) mit Schadenshöhe (I) ($R = P \times I$) aufzuzeigen, wurde das Risikoobjekt „D“ in Abbildung 9 hervorgehoben. Der in Tabelle (a) berechnete Risikowert für „D“ entspricht demnach 450 und ist der zweitniedrigste Wert aller angeführten Risikoobjekte. Bei der Betrachtung als Dupel ($R = [P; I]$) bleiben die Werte für die Eintrittswahrscheinlichkeit und die Schadenshöhe bestehen ($R_D = [5; 90]$). Dadurch bleibt im gegebenen Beispiel auch der hohe Wert der Schadenshöhe direkt ersichtlich. Eingetragen in das Risikodiagramm (b) fällt zudem die Lage am oberen Rand des Diagramms auf. Würde diesem Risikoobjekt aufgrund des geringen Risikowerts wenig Aufmerksamkeit geschenkt, könnte dies fatale

[261] Vgl. Erdogan et al. (2014), S. 641
[262] Vgl. Gadatsch/Mangiapane (2017), S. 17 ff.
[263] Vgl. Redmill (2004), S. 12 f.
[264] Vgl. Ramler/Felderer (2015), S. 359

Folgen haben. So fällt die Einschätzung angesichts der Sichtbarkeit des hohen Schadenspotenzials wohl anders aus.

Die Behandlung der Risikowerte als Dupel birgt den zusätzlichen Vorteil, dass Risiken hinsichtlich ihrer Klassifizierung nachjustiert werden können. So merken Ramler et al. an, dass sie in ihrer Fallstudie Risikoobjekte, die nahe an der Grenze zu anderen Risikoklassen lagen, aufgrund von Diskussionen umstuften.[265] Das in der Abbildung 9 angeführt Risikoobjekt „B“ wäre ein Beispiel für eine solche Diskussion. Allerdings wird der Mehrwert dieses Vorgehens durch Ramler et al. nicht belegt.[266]

(a)

Risikoobjekte

	P	I	R	Klasse
A	10	20	200	I
B	30	45	1350	I
C	75	30	2250	II
D	5	90	**450**	III
E	25	75	1875	III
F	70	80	5600	IV

(b)

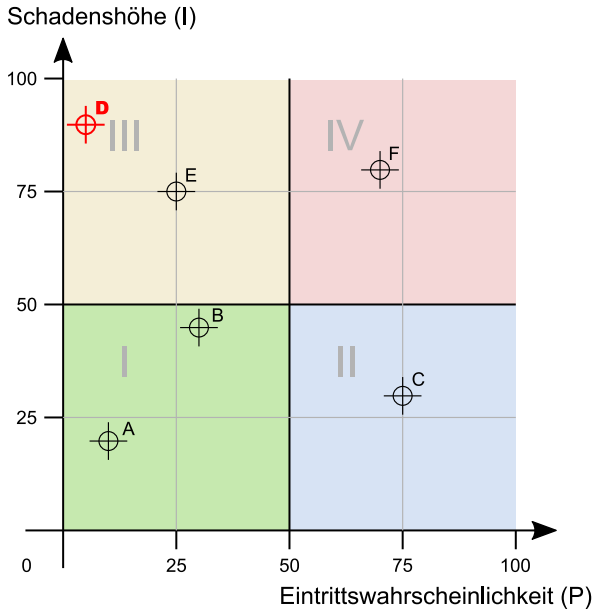

Abbildung 9: Unterschied Risikowert berechnet und klassifiziert[267]

Empfehlungen zur Risikobewertung

Die Risikobewertung bestimmt beim risikobasierten Testen maßgeblich den Fokus des Testprozesses und spielt somit für die gesamte Qualitätssicherung eine zentrale Rolle.[268] Dadurch könnte der Eindruck entstehen, dass die Risikobewertung möglichst genau und feingranular erfolgen muss. Droste et al. warnen jedoch vor einer ***Scheingenauigkeit*** bei quantitativen Schätzungen. Ihrer Auffassung nach stiftet es wenig Mehrwert, nach Genauigkeit zu streben, weshalb sie ein grobes Raster für die Einteilung empfehlen.[269] Auch Amland weist darauf hin, dass es zielführender

[265] Vgl. Ramler/Felderer (2015), S. 361
[266] Vgl. Ramler/Felderer (2015), S. 361
[267] Quelle: Verfasser
[268] Vgl. Felderer et al. (2015), S. 32
[269] Vgl. Droste/Merz (2019), S. 47

ist, Risiken zu gruppieren. Seiner Argumentation zu Folge, wird in anderen Branchen das Risiko oft als Verteilung gesehen, anstatt als berechneter Wert oder Dupel.[270]

Um die Einschätzungen für Menschen einfacher und intuitiver zu gestalten, wird empfohlen die ***Klassifizierung verbal*** zu gestalten. So schlägt Amland für die Unterteilung der Schadenshöhe die Werte „gering", „mittel" und „hoch" vor.[271] Felderer et al. geben als Beispiel für die Eintrittswahrscheinlichkeit eine Skala von „selten", „gelegentlich" „mittel", „häufig" und „hoch" an.[272] Die Skalen müssen für die Berechnung bzw. die Darstellung in einem Risikodiagramm mit Werten hinterlegt werden. Veenendaal gibt an, dass in der Praxis meist aufsteigende Zahlen wie 1 bis 3 oder 1 bis 5 verwendet werden. Jedoch sind auch nicht lineare Skalen wie 0, 1, 3, 5, 9 möglich. Letztere bieten den Vorteil, dass sich hohe Beurteilungen durch ihre entsprechend höheren Werte abheben und daher besser ersichtlich sind.[273]

Alternativ kann mit den Skalen auch eine ***Risikomatrix*** gebildet werden, in welche die Risiken direkt eingetragen werden.[274] Die Abbildung 10 zeigte eine Risikomatrix, in der die Schadenshöhe nach oben und die Eintrittswahrscheinlichkeit nach rechts aufgetragen ist. Die Unterteilung erfolgte dabei mit den oben genannten textuellen Werten. Jedes Feld wird einer Risikoklasse zugeordnet. Im Beispiel wurden hierfür die drei Risikoklassen „I", „II" und „III" bestimmt.

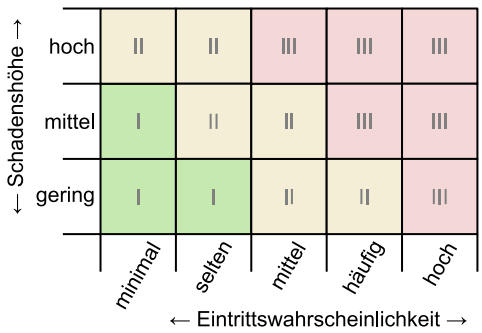

Abbildung 10: Risikomatrix[275]

In neuen Softwareprojekten, insbesondere wenn Erfahrungswerte aufgrund von neuen Teams, Technologien und/oder Themengebieten fehlen, kann mitunter keine Einschätzung der Fehlerwahrscheinlichkeit erfolgen. Redmill schlägt in einem solchen Fall eine „***Single-Factor Analysis***" (dt. „Einzelfaktorenanalyse") aufgrund der Schadenshöhe vor. Die ersten Testdurchläufe sollen sich seiner Meinung nach zuerst auf die Teile mit der höchsten Schadenshöhe konzentrieren. Aufgrund der damit erlangten Erfahrungswerte sollen dann die Wahrscheinlichkeitswerte ermittelt und in die Risikobewertung einbezogen werden.[276]

---

[270] Amland (2000), S. 290
[271] Vgl. Amland (2000), S. 290
[272] Vgl. Felderer et al. (2012), S. 177
[273] Vgl. Veenendaal (2011), S. 13
[274] Vgl. Ramler/Felderer (2016), S. 360
[275] Quelle: Verfasser
[276] Vgl. Redmill (2005), S. 4 ff.

Ebenfalls sei erwähnt, dass Felderer et al. für eine ***automatisierte Risikobewertung*** plädieren. Nach ihrer Auffassung ist die Risikobewertung mit einem hohen Zeit- und Kostenaufwand verbunden, was die Ressourceneinsparung durch risikobasiertes Testen mindert.[277] Sie gehen ebenfalls davon aus, dass eine automatische Auswertung von Metriken zur Eintrittswahrscheinlichkeit einfacher durchzuführen und aussagekräftiger ist als eine Auswertung hinsichtlich Schadenshöhe.[278]

[277] Vgl. Felderer et al. (2012), S. 175
[278] Vgl. Felderer et al. (2012), S. 175

4.5 Phase: Risikobasiertes Testen

In der Literatur werden bezüglich des risikobasierten Testens und der zugehörigen Maßnahmen nur zwei grundlegende Ideen aufgezeigt:

1) Was zu testen ist → Testfall-Priorisierung mittels Aktivitätsdiagrammen[279]
2) Wie zu testen ist → Teststrategie aufgrund der Risikoklassifizierung[280]

Testfall-Priorisierung mittels Aktivitätsdiagramm

Bei der ersten Idee werden Tests aufgrund der Gewichtung von Anwendungsfällen in der Software priorisiert. Ein Vertreter der Testfall-Priorisierung mittels Aktivitätsdiagramm ist der „***ranTEST-Ansatz***“. Hierfür werden in einem ersten Schritt die erhobenen Anforderungen mittels Anwendungsfalldiagrammen[281] (engl. „use case diagram“) dokumentiert. Die so erstellten Anwendungsfälle werden mittels nicht näher beschriebener Risikoanalyse bewertet. Es folgt die Verfeinerung mittels Verhaltensmodellierung (u.a. Aktivitätsdiagrammen[282]), bei der die unterschiedlichen Abläufe eines Anwendungsfalls beschrieben werden. Daraus ergeben sich verschiedene Sichten, die für eine Testfall-Priorisierung herangezogen werden können. So können durch die ***Risikobewertungen der einzelnen Aktionen*** die Gesamtrisiken für verschiedene Szenarien ermittelt werden. Dadurch werden die besonders risikoreichen Pfade sichtbar. Ebenfalls ist es möglich zu prüfen, bei welchen Abläufen besonders risikoreiche Aktionen involviert sind. Und es werden die Vorgänger von Aktionen sichtbar, wodurch erhoben werden kann über welche Transitionen (Vorbedingungen, Nachrichten) Aktionen aufgerufen werden. Aufgrund der Risikobewertung der einzelnen Aktionen lässt sich somit das Gesamtrisiko für jedes Szenario ermitteln.[283] Der Fokus der Testaktivität liegt somit nicht mehr alleine auf einzelnen Aktionen, sondern erfasst diese in einem größeren Konstrukt.

Zur Veranschaulichung zeigt Abbildung 11 ein einfaches Aktivitätsdiagramm mit fünf risikobewerteten Aktionen und zwei Entscheidungsknoten. Für die Szenarien werden jeweils die Aktionen vom Start- bis zum Zielpunkt dokumentiert und auf Basis der Einzelrisikowerte wird ein Gesamtrisiko bestimmt. Ein Auszug aus den daraus resultierenden Szenarien ist in der Tabelle 2 aufgelistet.

---

[279] Beispiele: Bauer et al. (2008); Chen et al. (2002); Stallbaum et al. (2008)
[280] Beispiele: Ottevanger (1999); Ramler/Felderer (2015)
[281] Diagrammtyp der Unified Modeling Language (UML), siehe Kecher et al. (2018), S. 209 ff.
[282] Diagrammtyp der Unified Modeling Language (UML), siehe Kecher et al. (2018), S. 225 ff.
[283] Vgl. Bauer et al. (2008), S. 102 ff.

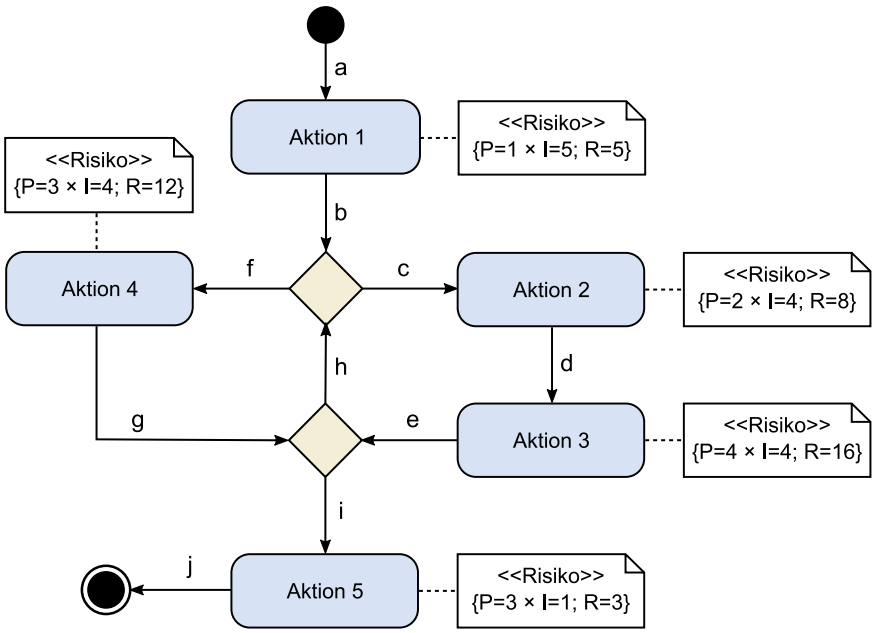

Abbildung 11: Risikobewertetes Aktivitätsdiagramm[284]

Szenario	***Pfad***	***∑ Risiko***
S1	a, b, c, d, e, i, j	32
S2	a, b, c, d, e, h, c, d, e, i, j	56
S3	a, b, c, d, e, h, c, d, e, h, c, d, e, i, j	80
S4	a, b, f, g, i, j	20
S5	a, b, f, g, h, f, g, i, j	32
S6	a, b, f, g, h, f, g, h, f, g, i, j	44
S7	a, b, c, d, e, h, f, g, i, j	46
S8	a, b, f, g, h, c, d, e, i, j	46
…	…	…

Tabelle 2: Risikobewertete Szenarien[285]

Ein Problem, dass sich bei diesem Ansatz ergibt, ist das mehrfache Durchlaufen von Aktionen. Diese im Fachjargon bezeichneten „***Schleifen***“ (engl. „loops“) können im Extremfall zu endlosen Wiederholungen führen. Dadurch würden sich unendlich lange Pfade und letztlich unendlich hohe Risiken ergeben. Im angeführten Beispiel kann der Teilpfad „f, g, h“ theoretisch unendlich oft durchlaufen werden. Die Szenarien S4 bis S6 in Tabelle 2 unterscheiden sich nur dahingehend, dass der Teilpfad „f, g, h“ unterschiedlich oft durchlaufen wird (S4 0x, S5 1x u. S6 2x). Bei jedem Durchlauf wird das Ergebnis um das Risiko der Aktion 4 ($R_4 = 12$) erhöht werden. Um dem entgegenzuwirken, beschränken Stallbaum et al. in ihrem ***vergleichbaren Ansatz*** „***RiteDAP***“ die maximale

[284] Quelle: Verfasser
[285] Quelle: Verfasser

Wiederholung einer Schleife auf zwei Durchläufe. Weiters geben sie an, dass Szenarien gebaut werden, in welchen Schleifen nach Möglichkeit nicht durchlaufen werden.[286]

Alternativ zur Berechnung des Gesamtrisikos durch die Summierung der Einzelrisiken, führen Stallbaum et al. noch eine zweite Möglichkeit an. Dabei werden nur jene ***Risiken*** bewertet, ***die nicht*** bereits ***bei einem vorherigen Szenario*** bewertet wurden. Als Ausgangspunkt wird das Szenario mit dem höchsten Gesamtrisiko herangezogen. Im angeführten Beispiel ist das S3 mit einer Bewertung von $R_{S3} = 80$. In diesem Szenario werden die Aktionen 1, 2, 3 und 5 abgedeckt. In der Priorisierung ist das nächste Szenario S6. Durch den maximalen Schleifendurchlauf der noch nicht abgedeckten Aktion 4 ergibt sich für S6 eine Bewertung von $R_{S6} = 36$.[287] Nachfolgend gibt es keine Aktionen mehr, die nicht abgedeckt sind, weshalb die Bewertungen aller anderen Szenarien auf 0 sinkt.[288]

Teststrategie aufgrund der Risikoklassifizierung

Das zweite Konzept ist die Festlegung der Teststrategie aufgrund der Risikoklassifizierung. Dabei werden zuerst Risikoklassen definiert. Für ***jede Risikoklasse*** wird anschließend eine ***eigene Teststrategie*** festgelegt. Diese umfasst neben den anzuwendenden Testmethoden auch die Beschreibung wie und mit welcher Genauigkeit die Tests durchzuführen sind. Aufgrund der Risikobewertung werden die Risikoobjekte den Risikoklassen zugeordnet. Dadurch gibt die Risikoklasse durch ihre hinterlegte Teststrategie vor, wie ein Risikoobjekt zu testen ist.[289]

Da die Teststrategien in der Regel sehr allgemein gehalten sind, muss in einem letzten Schritt die Testspezifikation für jedes Risikoobjekt entsprechend seinen Eigenschaften und der zugeordneten Teststrategie festgelegt bzw. angepasst werden.[290]

Zur Veranschaulichung werden in Abbildung 12 solche Testmethoden aufgeführt, die den Risikoklassen „I“ bis „IV“ zugeordnet wurden. Ein Haken bedeutet, dass diese Testmethode für die entsprechende Risikoklasse angewendet wird. So ist für die Risikoklasse „I“ nur die Testmethode des explorativen Testens vorgesehen. Um die Zuordnung zu verdeutlichen, wurden im Anschluss die in Abbildung 8 bewerteten Risikoobjekte entsprechend ihrer Klassifizierung den Testmethoden zugeordnet.

---

[286] Vgl. Stallbaum et al. (2008), S. 68

[287] Berechnung: $3 \times R_4$ ohne erneute Bewertung von R_1 und $R_5 = 0 + 3 \times 12 + 0 = 36$

[288] Vgl. Stallbaum et al. (2008), S. 68

[289] Vgl. Ramler/Felderer (2015), S. 361

[290] Vgl. Ramler/Felderer (2015), S. 362

Testmethode	Risikoklasse				Risikoobjekt / Risikoklasse	A	B	C	D	E	F
	I	II	III	IV		I	I	II	III	III	IV
Exploratives Testen	✓			✓		✓	✓				✓
Manuelles Testen (Standardabläufe)		✓	✓	✓				✓	✓	✓	✓
Manuelles Testen (Alternative Wege)			✓	✓					✓	✓	✓
Unit Tests / Regressionstests			✓	✓					✓	✓	✓
Code Review				✓							✓

Abbildung 12: Zuordnung von Testmethoden zu Risikoklassen[291]

Für die Festlegung der Testmethoden ist lt. Ramler et al. eine Liste an im Unternehmen bzw. im Projekt anwendbaren ***Qualitätssicherungs- und Testtechniken erforderlich***. Das sind Techniken, für die ausreichende Kenntnisse, Erfahrungen und entsprechende Werkzeugunterstützung vorhanden sind.[292] Wie diese Liste erstellt wird bzw. nach welchen Kriterien die anwendbaren Techniken ausgewählt werden, wurde dabei nicht erläutert.

In einem analogen Konzept stützt sich Ottevanger ***zusätzlich auf Qualitätsmerkmale*** (ähnlich denen der ISO/IEC 25010-2011[293]). Statt ausschließlich die Risikoobjekte zu bewerten, wird mit den Stakeholdern die relative Wichtigkeit der vorgegebenen Qualitätsmerkmale bestimmt. Anschließend wird jedes Risikoobjekt hinsichtlich der Relevanz der als wichtig angesehenen Qualitätsmerkmale zusätzlich gewichtet. Aufgrund der sich so ergebenden Prioritäten werden dann die Testspezifikationen für jedes Risikoobjekt festgelegt.[294]

[291] Quelle: In Anlehnung an Ramler/Felderer (2015), S. 361
[292] Vgl. Ramler/Felderer (2015), S. 361
[293] ISO/IEC 25010:2011 (2011)
[294] Vgl. Ottevanger (1999), S. 7 ff.

4.6 Phase: Optimieren des Testens

Neben den Maßnahmen des risikobasierten Testens werden in der Literatur auch andere Prinzipien und Handlungsempfehlungen aufgeführt, die nicht explizit dem risikobasierten Testen zugeordnet werden können. Da sie allerdings ebenfalls einen wesentlichen Beitrag bei der Durchführung von Softwaretests leisten, werden sie hier gesondert erwähnt.

Programmierer dürfen nicht testen

Softwareentwickler sollen die eigene Software nicht selbst testen. Zum einen werden die konzeptionellen Fehler des Implementierens beim Testen höchstwahrscheinlich wiederholt. Denn es ist davon auszugehen, dass eine bei der Implementierung getroffene ***fehlerhafte Annahme***, beim Erstellen von Testfällen ***wiederholt*** wird. Weiters hat sich gezeigt, dass Softwareentwickler wesentlich genauer arbeiten, wenn ihnen bewusst ist, dass jemand anderer ihre Arbeit überprüft. Und zuletzt wurde festgestellt, dass durch Softwareentwickler gefundene Fehler (insb. schwerwiegende) zwar behoben, die Korrektur jedoch weder exakt und umfänglich dokumentiert noch die dafür benötigte Zeit richtig angegeben wurde.[295]

Testen soll die Qualität steigern

Der tiefere Sinn des Testens besteht darin, für eine hohe Fehlererkennungsrate und in Folge für eine ***effiziente Fehlerbehebung*** zu sorgen. Das Testen ist somit mit der Fehleridentifikation (eig. Analyse von Anomalien; siehe 2.3.3 „*Grundlagen und Begriffe*“ Absatz „*Anomalie*“) und der Fehlerbehebung durch die Programmierer stark verschränkt.[296] Im Idealfall wird für jeden Fehler ein Testfall gefunden, der sowohl den Fehler wie auch dessen Konsequenzen klar und reproduzierbar aufzeigt. Im Allgemeinen weisen gute Testfälle folgende Eigenschaften auf:[297]

- Sie decken mit hoher Wahrscheinlichkeit Fehler auf.
- Sie sind weder zu einfach noch zu komplex.
- Sie machen Programmierfehler offensichtlich.

Optimalerweise erfolgt die Erstellung der Testfälle in Abstimmung mit den Stakeholdern der jeweiligen Anforderungen, den Fachbereichen und den Softwareentwicklern. Dies stellt zum einen sicher, dass die Testaktivitäten die jeweiligen Erwartungen erfüllen. Zudem wird sichergestellt, dass durch die Testfälle die Bedürfnisse der Stakeholder geprüft werden.[298] Außerdem merken Alam et al. an, dass die Zusammenarbeit mit den Stakeholdern hinsichtlich Anforderungen und Testmöglichkeiten das Verständnis und die Wertigkeit des Testens erhöht.[299]

Testfälle müssen angepasst und gewartet werden

Weiters ist zu beachten, dass Testfälle angepasst werden müssen. Dies kann zum einen durch sich ergebende Änderungen am System erfolgen, welche eine Adaption bestehender oder die

---

[295] Vgl. Broy/Kuhrmann (2021), S. 473; Mayr (2005), S. 258; Witte (2019), S. 14
[296] Vgl. Broy/Kuhrmann (2021), S. 473
[297] Vgl. Mayr (2005), S. 262
[298] Vgl. Droste/Merz (2019), S. 150
[299] Vgl. Alam/Khan (2013), S. 35

Generierung zusätzlicher Testfälle erfordert.[300] Es kann allerdings auch aufgrund von entdeckten Anomalien sinnvoll sein, zusätzliche Testschritte zu integrieren. So weisen sowohl Witte wie auch Veenendaal darauf hin, dass Fehler oft gehäuft in einzelnen Bereichen auftreten.[301] Veenendaal erklärt dies mit Problemen bei der Entwicklung. Dies kann entweder lokal einen Bereich der Software betreffen (z.B., weil dieser von einem unerfahrenen Programmierer implementiert wurde). Oder es kann sein, dass eine Art von Fehlern systematisch wiederholt wurde (z.B., weil ein grundlegender Zusammenhang nicht ausreichend geklärt wurde).[302]

Entwicklung einbinden

Frühes Feedback durch Testen ermöglicht es den Softwareentwicklern früh aus Fehlern zu lernen und entsprechende Verbesserungsmaßnahmen zu etablieren.[303] Zudem ist eine enge Einbindung der Softwareentwicklung in das Testen empfehlenswert, da nur durch eine geeignete Systemarchitektur eine Testautomatisierung möglich ist.[304] Außerdem kann durch eine Modularisierung der Software sowohl die Fehleranalyse wie auch die -behebung erheblich vereinfacht werden.

In die andere Richtung kann die Kommunikation der Risiken die Entwicklung unterstützen. So ist es möglich, bereits während der Entwicklung auf besonders risikobehaftete Teile Rücksicht zu nehmen. Sei es, dass diese Bereiche frühestmöglich entwickelt werden und so ein frühes und intensives Testen ermöglichen. Oder dass für diese Bereiche besonders erfahrene Softwareentwickler eingesetzt werden.[305]

Qualitätssicherung der Qualitätssicherung

Droste et al. merken an, dass niemand – auch das Testmanagement nicht – vor Betriebsblindheit gefeit ist. Weshalb auch das Test-Team Maßnahmen zur Qualitätssteigerung durchzuführen hat. Es ist ihrer Auffassung nach daher ebenfalls nötig, die durch die vom Test-Team erstellten Dokumente durch die betreffenden Projektmitglieder und Stakeholder prüfen zu lassen.[306]

Sukzessives Testen

Weiters empfiehlt es sich kleine, testbare Bereiche möglichst früh zu testen. Damit ist deren Funktionalität für die folgenden, meist komplexeren Tests bereits geprüft. So besteht ein System aus Komponenten und diese wiederum aus Modulen. Wird am Ende das gesamte System in einem einzigen Test geprüft, sind Fehler oft wesentlich schwerer zu lokalisieren. Zudem können sich Änderungen auf andere Teile der Software auswirken, was mitunter zu aufwendigen Änderungen und Anpassung führen kann.[307] Diese hätte damit einen deutlich höheren Ressourcenverbrauch zur Folge.

---

[300] Vgl. Witte (2019), S. 11 ff.
[301] Vgl. Veenendaal (2011), S. 2; Witte (2019), S. 13
[302] Vgl. Veenendaal (2011), S. 2
[303] Vgl. Witte (2019), S. 17
[304] Vgl. Witte (2019), S. 234
[305] Vgl. Amland (2000), S. 294; Redmill (2004), S. 13; Rosenberg et al. (1999), S. 5
[306] Vgl. Droste/Merz (2019), S. 77 f.
[307] Vgl. Witte (2019), S. 87

4.7 Phase: Erkenntnisse auswerten

Besonders in Projekten mit neuem Umfeld stellen die Identifizierung und Bewertung von Risiken eine besondere Herausforderung dar. Denn zu Beginn sind meist nur Aussagen einzelner Stakeholder vorhanden. Im Verlauf des Projekts und mit zunehmendem Erkenntnis- und Informationsgewinn, können Risikoeinschätzungen und folglich auch entsprechende Tests bzw. Teststrategien angepasst werden.[308] Für den Erfolg sind somit die ***Erfahrungswerte*** von Domänenexperten, Softwarearchitekten und -entwicklern sowie Testern äußerst förderlich.[309]

Wie bereits mehrfach erwähnt, ist einer der maßgeblichen Vorteile des risikobasierten Testens die ***Optimierung des Ressourceneinsatz***. Diese muss ebenfalls die Erkenntnisse aus den verschiedenen Phasen berücksichtigen. Allerdings ist laut Amland besonders zu Beginn eines Projekts der tatsächliche Aufwand für verschiedene Testaktivitäten schwer einzuschätzen. Daher ist es seiner Auffassung nach wichtig, die Informationen besonders bei den ersten Tests zu erfassen und für die Ermittlung besonders ressourcenintensiver Tests zu nützen.[310] Auf diese Weise lassen sich realistischere Aufwandsschätzungen erstellen bzw. ableiten. Zudem kann der Ressourceneinsatz noch einmal optimiert werden.

So ermöglicht das risikobasierte Testen eine ***Kürzung der Softwaretests***, indem jene Testfälle mit geringem Risiko zuerst gestrichen werden können. Dies führt dazu, dass zumindest die Auswirkungen auf das Qualitätsrisiko insgesamt möglichst geringgehalten werden.[311] Zudem können die ermittelten Testergebnisse in Kombination mit deren Risiko als Grundlage für die ***Freigabe von Software*** dienen.[312]

Ebenfalls wird durch das frühe Befassen mit dem Testen die Testphase an den Beginn des Projekts verlagert. Grundsätzlich (auch abseits von risikobasiertem Testen) kann direkt nach Abschluss der Anforderungserhebung mit der Testplanung begonnen werden.[313] Dies birgt die Vorteile, dass der Aufwand für ***Testaktivitäten früher und genauer abgeschätzt*** werden kann, und dass frühzeitig Diskrepanzen zwischen vorgesehenem und benötigtem Testbudget erkannt werden.[314]

Generell profitiert die Projektleitung, da schwierige ***Entscheidungen leichter*** getroffen werden können, wenn die damit verbundenen Risiken zuvor bekannt sind.[315]

---

[308] Vgl. Bach (1999), S. 9
[309] Vgl. Ottevanger (1999), S. 4 f.; Redmill (2004), S. 10
[310] Vgl. Amland (2000), S. 289
[311] Vgl. Alam/Khan (2013), S. 35
[312] Vgl. Alam/Khan (2013), S. 35
[313] Vgl. Amland (2000), S. 294
[314] Vgl. Brandes/Heller (2017), S. 8; Ramler/Felderer (2015), S. 362; Ramler/Felderer (2015), S. 370
[315] Vgl. Ramler/Felderer (2015), S. 370

4.8 Übergreifende Erkenntnisse

Einer der entscheidenden Erfolgsfaktoren ist lt. Amland, dass das risikobasierte Testen von der ***Organisation unterstützt*** wird. So müssen sich Softwareentwickler beispielsweise darauf konzentrieren, zuerst die kritischen Fehler zu beheben statt die einfachen. Testmanager und/oder Risikomanager müssen vorab festlegen, welche Bereiche vorrangig und mit wieviel Aufwand getestet werden, welche Fehler zuerst korrigiert werden müssen und wieviel Aufwand in die Testdokumentation zu fließen hat.[316]

Ebenfalls weist Amland darauf hin, dass risikobasiertes Testen keine Probleme lösen wird, wenn kein Testprozess etabliert ist und das Testen komplett unstrukturiert erfolgt.[317] In der Literatur wird das Thema Definition und Qualitätssicherung von Prozessen allerdings nur selten explizit behandelt. Ohne klar definierte Prozesse und/oder Vorgehensweisen unterliegen allerdings auch Arbeitsschritte der subjektiven Interpretation der im Projekt beteiligten Personen.

[316] Vgl. Amland (2000), S. 293
[317] Vgl. Amland (2000), S. 293

5 Diskussion

Mangelnde Evidenz

Redmill forderte bereits 2004 in seiner Arbeit, dass unbedingt ***mehr Forschung*** auf dem Gebiet des risikobasierten Testens nötig wäre, um eine einheitliche Wissensbasis zu schaffen. Nach ihm muss zuerst ein einheitliches Verständnis hinsichtlich des Risikos, der Risikoanalyse und dem Risikomanagement geschaffen werden. Im Anschluss ist dann die Anwendung des risikobasierten Testens für die Testplanung (und auch für die Projektplanung) auszuarbeiten und zu beschreiben.[318]

Ein Jahrzehnt später bemängeln Erdogan et al. in ihrer Arbeit, dass die meisten von ihnen untersuchten Beiträge nur wenige Hinweise auf den Nutzen des jeweils vorgeschlagenen Ansatzes bieten.[319] Und auch Ramler et al. weisen darauf hin, dass es, obwohl es doch einige Ansätze zu risikobasiertem Testen gibt, ***nur wenige empirische Studien*** in diesem Bereich gibt.[320]

Letztlich schreiben Großmann et al. 2020, dass risikobasiertes Testen inzwischen recht populär ist und es eine wachsende Nachfrage seitens der Industrie gibt. Allerdings weisen sie ebenfalls darauf hin, dass Normen[321] hinsichtlich der konkreten Umsetzung meist abstrakt sind und keine Anleitungen für die Definition, Anpassung oder Bewertung von risikobasierten Testansätzen bieten. Besonders im Hinblick auf die zunehmende Anzahl an risikobasierten Testansätzen ist ihrer Ansicht nach eine methodische Unterstützung zur Definition, zur Anpassung, zur Kategorisierung, zur Bewertung, zum Vergleich und zur Auswahl risikobasierter Testansätze erforderlich.[322] Erdogan et al. vermuten in diesem Bezug, dass dies auf einen offensichtlichen ***Mangel an Grundlagen*** und das Fehlen von allgemein anerkannten Best-Practices zurückzuführen ist.[323]

Fehlende Verfahrensanweisungen

Die aktuell vorhandene Literatur befasst sich überwiegend nur mit Teilbereichen des risikobasierten Testens, primär mit der Risikobewertung. Dabei sind die vorhandenen Konzepte überwiegend von theoretischer Natur und geben nur ***selten konkrete Handlungsempfehlungen*** für deren Umsetzung. Weiters lassen sich durch die gegebene Heterogenität in der Literatur die vorhandenen Konzepte nicht oder nur schwer vereinen. Daraus ergeben sich mehrere Probleme.

Wie bereits angemerkt wurde, ist keine direkte Anwendung des risikobasierten Testens möglich. Es bedarf vorab einer ***entsprechenden Aufbereitung und Adaption*** durch die im Projekt Verantwortlichen (Projektmanagement, Testmanagement, …). Neben dem Umgang mit Risiken (Identifikation, Bewertung, Beherrschung, …) muss ebenfalls festgelegt werden, wie die daraus gewonnenen Erkenntnisse sinnvoll und nutzbringend auf das Testen angewendet werden können. Auf letzteren Bereich wird in der Literatur nur selten explizit eingegangen. Bei der Anwendung einer reinen

---

[318] Vgl. Redmill (2004), S. 14
[319] Vgl. Erdogan et al. (2014), S. 641
[320] Vgl. Ramler/Felderer (2015), S. 368
[321] Z.B. ISO/IEC/IEEE 29119-2:2021 (2021)
[322] Vgl. Großmann et al. (2020), S. 40
[323] Vgl. Erdogan et al. (2014), S. 641

Testfallpriorisierung wird lt. Felderer et al. das Potential des risikobasierten Testens nicht ausgeschöpft, da keine Optimierung der Testressourcen stattfindet.[324] Ebenfalls werden in keinem der betrachteten Ansätze die unterschiedlichen Teststufen berücksichtigt. Es scheinen alle Testfälle unabhängig voneinander zu existieren. Jedoch sind beispielsweise Integrationstests erst sinnvoll, wenn entsprechende Modultests erfolgreich absolviert wurden.

Weiters finden sich in den Ansätzen kaum Hinweise, wie sich die unterschiedlichen Tätigkeiten des risikobasierten Testens in die verschiedenen ***Phasen der Softwareentwicklung integrieren*** lassen. Zwar wird das risikobasierte Testen selbst in Tätigkeiten und teilweise in Phasen unterteilt, jedoch fehlt grundsätzlich eine Koppelung zum Softwareentwicklungsprozess[325].

Zuletzt sprechen nur Chen et al. das Thema ***Rückverfolgung von Anforderungen*** an. Wie bereits mehrfach erwähnt, sind Anforderungen – ob funktionale oder andere – ein maßgeblicher Faktor für die Softwareentwicklung. Daher ist es nach Chen et al. unerlässlich, festzustellen, welche Anforderung durch welchen Testfall bzw. Testfälle verifiziert wird.[326] Nur auf diese Weise kann sichergestellt werden, dass auch jene Teile der Software, die elementare Anforderungen erfüllen, trotz minimaler Eintrittswahrscheinlichkeit, adäquat getestet werden.

Im Hinblick auf den in Softwareprojekten bewerten Einsatz von Agilität stellt sich in der Anwendung des risikobasierten Testens in Projekten aufgrund deren Heterogenität und die sich daraus ergebende Vielzahl an Vorgehensmodellen die Frage, inwieweit ein vorgegebener Prozess überhaupt eingehalten werden könnte. Infolgedessen könnte es daher sinnvoll und nötig sein, dass die Ansätze zu risikobasiertem Testen den Anwendern ***ein Set an Methoden*** an die Hand geben. Diese können im Anschluss für das jeweilige Entwicklungsprojekt bzw. den jeweiligen Entwicklungsprozess adaptiert und verwendet werden. Wichtig ist allerdings für Unternehmen, dass hinsichtlich der gewählten Methoden und ihrer Adaption eindeutige Verfahrensanweisungen erstellt werden. Nur so kann sichergestellt werden, dass eine gewisse Homogenität in die Projekte gebracht wird. Dies ist die Grundlage, um Prozessrisiken und allgemeine Projektrisiken zu senken und Projekte vergleichbar zu machen.

Methodenvielfalt

Sowohl Erdogan et al. wie auch Felderer et al. zeigen in ihren Metastudien auf, dass die Bandbreite an ***Möglichkeiten zur Risikoidentifikation und -bewertung*** in der Literatur sehr hoch ist. So reichen die vorgeschlagenen Ansätze von reinem Bauchgefühl bis hin zu formalen mathematischen und statistischen Methoden.[327] Das Problem im ersten Extrem – dem Bauchgefühl – liegt lt. Redmill im täglichen Umgang mit dem Thema Risiko. Intuitiv wird es von jedem verstanden, da ein jeder in seinem Leben Entscheidungen zur Vermeidung, Verringerung, Aufteilung oder Akzeptanz von Risiken trifft. Es wird daher implizit angenommen, dass diese bewährte Kompetenz, die in einfachen Situationen funktioniert, auch für komplexe Situationen angewendet werden kann.[328] Für das andere Extrem – der formalen Berechnung – fehlt es laut Redmill vor allem zu Beginn meist

[324] Vgl. Felderer et al. (2012), S. 163; Felderer/Ramler (2014), S. 551; Stallbaum et al. (2008), S. 67
[325] Siehe Abschnitt 2.4 „*Vorgehensmodelle in der Softwareentwicklung*“
[326] Vgl. Chen et al. (2002), S. 3
[327] Vgl. Erdogan et al. (2014), S. 640 f.; Felderer et al. (2015), S. 37
[328] Vgl. Redmill (2004), S. 4 f.

an einer entsprechenden Datenbasis.[329] Weiters ergibt sich das Problem, dass hinsichtlich der formalen Risikobewertung unterschiedliche Methoden in der Literatur aufgezeigt werden. So zeigen Felderer et al. in ihrer Metastudie elf Ansätze[330] auf, die teilweise unterschiedliche formale Methoden verwenden.[331]

Bei dieser Bandbreite an Methoden kommt erschwerend der Umstand hinzu, dass es offenbar ***keine Fallstudien*** gibt, die ***die unterschiedlichen Ansätze vergleichen*** bzw. hinsichtlich ihrer Effizienz untersuchen. Wird der Einsatz von risikobasiertem Testen in einem Projekt gewünscht, obliegt es dem Gutdünken der Verantwortlichen, welches Verfahren angewendet wird.

In diesem Zusammenhang ist zudem interessant, dass keiner der untersuchten Ansätze Verfahren propagiert, welche ***für andere softwareentwickelnden Branchen mitunter verpflichtend*** sind. So unterliegen zum Beispiel Medizinprodukte innerhalb der europäischen Union (EU) einem strengen Regulatorium[332], welches unter anderem auch Vorgaben[333] zur Risikobewertung dieser Produkte vorschreibt. Da viel der hier vorgestellten Literatur von Autoren in der EU verfasst wurde,[334] wäre ein Blick dahingehend naheliegend gewesen.

Ein weiterer Bereich, der ebenfalls in der Literatur nicht behandelt wird, ist die ***Qualitätssicherung der Risikobewertung*** für das risikobasierte Testen. Obwohl erwähnt wird, dass die Risikobewertung ein zentrales Element des risikobasierten Testens ist,[335] werden keine Hinweise gegeben, wie die Qualität der Risikobewertung überprüft werden kann. Lediglich Veenendaal gibt einige Indikatoren, die auf eine suboptimale Bewertung schließen lassen.[336]

Überdies herrscht in der Literatur Uneinigkeit bezüglich der ersten Tests und den damit verbundenen Problemen der ***fehlenden Erfahrungswerte***. So empfiehlt Redmill mit der Single-Factor-Analyse sich zuerst auf die Teile mit der höchsten Schadenshöhe zu konzentrieren. Durch gefundene Fehler lassen sich anschließend die Eintrittswahrscheinlichkeiten für Fehler in den entsprechenden Risikoobjekten erheben.[337] Veenendaal hingegen empfiehlt in einem solchen Fall erst mittels explorativer Tests (siehe 2.3.4 „*Methoden zur Qualitätssicherung*“ Absatz „*Exploratives Testen*“) die fehleranfälligen Bereiche ausfindig zu machen. Auch wenn diese Tests seiner Auffassung nach oberflächlich sein können, ist es jedoch sehr wichtig, dass sie das gesamte System abdecken.[338] Bach geht sogar noch einen Schritt weiter und empfiehlt grundsätzlich mindestens ein Viertel des Testaufwands auf nicht-risikobasierte Testmethoden zu verwenden.[339]

In diesem Zusammenhang ist es ebenfalls auffällig, dass keiner der Ansätze die ***Softwareentwickler in die Risikoanalyse einbezieht***. So weist Veenendaal explizit darauf hin, dass Fehler generell

---

[329] Vgl. Redmill (2004), S. 6
[330] Insg. wurden 17 Ansätze untersucht
[331] Vgl. Felderer et al. (2015), S. 38
[332] Siehe Verordnung (EU) 2017/745
[333] Siehe DIN EN ISO 13485:2016-08 (2016); DIN EN ISO 14971:2020-07 (2020)
[334] Z.B. Amland (NOR); Bauer (GER); Felderer (AUT); Kloos (GER); Ottevanger (NLD); Ramler (AUT); Redmill (GBR); Stallbaum (GER); Veenendaal (GBR)
[335] Vgl. Felderer et al. (2015), S. 32
[336] Vgl. Veenendaal (2011), S. 13 f.
[337] Vgl. Redmill (2005), S. 4 ff.
[338] Vgl. Veenendaal (2011), S. 11
[339] Vgl. Bach (1999), S. 9

ein Symptom dafür sind, dass Softwareentwickler Probleme bei der Entwicklung eines Bereichs der Software hatten.[340] Dass diese Probleme den Softwareentwicklern mitunter selbst während der Implementierung auffallen, wird jedoch nicht berücksichtigt. Ferner wird nicht berücksichtigt, dass niemand den Quellcode besser kennt als die Softwareentwickler selbst. Sie sind in der Lage sowohl die komplexen wie auch komplizierten Bereiche zu benennen, wenn Analysetools aufgrund der eingesetzten Technologien[341] bereits an ihre Grenzen stoßen.

Es sei noch erwähnt, dass die in Abschnitt 4.5 „*Phase: Risikobasiertes Testen*“ Absatz „*Testfall-Priorisierung mittels Aktivitätsdiagramm*“ erwähnten Ansätze „ranTest-Ansatz“ und „RiteDAP“ aufgrund der Verwendung von Aktivitätsdiagrammen starke Analogien zu ***modellbasiertem Testen*** aufzuweisen scheinen. Dies gilt ebenso für andere Ansätze, die im Zuge dieser Arbeiten betrachtet wurden und ähnliche Methoden propagieren. Da das modellbasierte Testen jedoch für diese Arbeit ausgeschlossen wurde, lassen sich diesbezüglich keine näheren Aussagen machen.

Fallstudien

Wie erwähnt besteht ein deutliches Problem in der Verfügbarkeit von Fallstudien, um so das RBT im praktischen Einsatz bewerten zu können. Die Publikationen gehen teilweise auf Schwierigkeiten ein, welche mit dem Design solcher Fallstudien einhergehen. Beim Erstellen von Fallstudien kommt erschwerend hinzu, dass eine gewisse Heterogenität zur Natur von Projekten gehört, was die Vergleichbarkeit vermindert. Diese scheint im Bereich der Softwareentwicklung umso schwerer, da eine ***Vielzahl von Parametern*** die Entwicklung beeinflussen können. Beispiele dazu sind das Fachwissen der Stakeholder, die eingesetzte(n) Technologie(n), die Erfahrungswerte der Projektmitglieder, der Zeit- und Ressourcendruck, das gesamte Projektumfeld und viele mehr.[342]

Hinsichtlich der Erstellung von Fallstudien weist Redmill auf einige Umstände hin, die es zu beachten gilt. Zum einen führt er an, dass für die Untersuchungen mitunter zusätzliche Testprogramme erstellt werden müssen (z.B. um Eintrittswahrscheinlichkeiten zu verifizieren). Es gilt zu beachten, dass die Aufwände für solche Tätigkeiten nicht dem tatsächlichen Testaufwand angelastet werden. Weiters merkt er an, dass auch eine sehr gute Risikoanalyse aufgrund einer schlechten Teststrategie zu schlechten und damit irreführenden Ergebnissen führen kann. Er plädiert dafür die Untersuchungen längerfristig anzusetzen. So sollen Aufwendungen für Testprogramme in Verbindung mit der Anzahl der Ausfälle und den Kosten für die Wartung im ersten Jahr oder länger betrachtet werden, um aussagekräftige Vergleiche zu erhalten. Ebenfalls empfiehlt er eine zusätzliche betriebswirtschaftliche Perspektive, falls vor allem die Effektivität im Vordergrund steht.[343]

Wirtschaftlichkeit

Wie mehrfach erwähnt, finden sich nur spärlich Fallstudien über den Einsatz des risikobasierten Testens in praktischen Projekten. Die untersuchten Fallstudien[344] beschäftigen sich vorrangig mit den Erfahrungen durch den Einsatz des risikobasierten Testens. ***Keine der Fallstudien weist***

---

[340] Vgl. Veenendaal (2011), S. 2
[341] Insb. Technologien zur losen Kopplung, z.B. „Dependency Injection“
[342] Vgl. Broy/Kuhrmann (2021), S. 7 ff.
[343] Vgl. Redmill (2005), S. 20
[344] Z.B. Amland (2000); Ramler/Felderer (2015)

einen wirtschaftlichen Mehrwert nach. Weder wird eine Steigerung der Qualität bei gleichzeitig konstantem Ressourcenverbrauch noch die Senkung des Ressourcenverbrauchs bei gleichzeitigem Beibehalten der Qualität aufgezeigt. Ebenfalls wurden in der Literatur keine Hinweise auf Indikationen ausgemacht, wie in einem Projekt die ***Wirtschaftlichkeit geprüft bzw. nachgewiesen*** werden kann. Dieser Umstand impliziert ebenfalls, dass die Effizienz der unterschiedlichen Ansätze zum risikobasierten Testen nicht erhoben werden kann. In Folge bedeutet dies, dass die Ansätze hinsichtlich ***Wirtschaftlichkeit nicht vergleichbar*** sind. Projektverantwortliche müssen sich somit auch in diesem Punkt bei der Auswahl eines Ansatzes auf ihr subjektives Empfinden bzw. ihre Erfahrung stützen.

Zu bedenken ist zudem, wie in der Einleitung (siehe 1.2 „*Problemstellung*“) erwähnt, dass für das risikobasierte Testen ein gewisser Mehraufwand nötig ist. So muss zumindest eine grundlegende Risikoanalyse durchgeführt werden. Inwieweit eine „herkömmliche“ Testplanung mehr oder weniger Aufwand gegenüber einer Testplanung für risikobasiertes Testen bedeutet, kann aufgrund der Heterogenität in der Literatur nicht beurteilt werden. Allerdings müssen für eine wirtschaftlich positive Bilanz alle diese ***Mehraufwände durch ein effektiveres Testen ausgeglichen*** werden.

Ein weiterer Aspekt, der in der Literatur keine Berücksichtigung findet, ist der Faktor Zeit. Wie eingangs in Abschnitt 2.1.4 „*Stellgrößen in Softwareprojekten*“ erwähnt wurde, spielt die Entwicklungszeit bzw. „Time-to-Market“ in Softwareprojekten eine immer wichtigere Rolle. Vor diesem Hintergrund ist zu bedenken, dass die ***Verfügbarkeit von Stakeholdern den Entwicklungsprozess verzögern*** kann. So ist das Wissen der Stakeholder für eine fundierte Risikobewertung unabdingbar, was deren stärkere Einbindung in den Testprozess zur Folge hat. Da Stakeholder aber üblicherweise in den Unternehmensprozess eingegliedert sind, stehen sie meist nur partiell zur Verfügung. Wenn zudem noch mehrere Vertreter einer Interessensgruppe für eine Diskussion erforderlich sind, werden mögliche Verzögerungen durch eine aufwendige Terminfindung offensichtlich. Ob und inwieweit solche Verzögerungen ebenfalls durch effektiveres Testen ausgeglichen werden können, ist fraglich.

Ein weiterer wirtschaftlicher Aspekt, der sich durch die Erhebung der Schadenshöhe und der Eintrittswahrscheinlichkeit eines Risikoobjekts ergibt, ist die ***Ermittlung der Kosten für einen*** darin enthaltenen ***Fehler***.[345] Diese ermittelten Kosten können nun unter anderem dazu verwendet werden, die maximalen Testressourcen für dieses Testobjekt zu bestimmen. Übersteigen die eingesetzten Testressourcen die Kosten für Fehler, ist es wirtschaftlich fraglich, ob dieses Objekt überhaupt getestet werden sollte. Diesbezüglich gilt es unbedingt die Kosten durch indirekte Schäden wie Reputationsverluste oder entstehende Schadensersatzansprüche zu beachten.

[345] Vgl. Yoon/Choi (2011), S. 194

6 Schlussfolgerung

Im Hinblick auf die zu Beginn der Arbeit gestellten Fragen, lassen sich diese nur unbefriedigend beantworten. Da das risikobasierte Testen selbst nicht so weit ausgereift ist, dass dezidierte Prozessschritte mit klaren Vorgehensweisen ausgearbeitet werden können, lassen sich diese auch nicht in die Vorgehensmodelle der Softwareentwicklung integrieren. Die erste Frage („Wie kann risikobasiertes Testen in allen Phasen der Softwareentwicklung eingesetzt werden?“) kann somit lediglich mit den allgemeinen Empfehlungen zum Thema „risikobasiertes Testen“ beantwortet werden. Dies umfasst vorrangig eine genaue und nachvollziehbare Risikoeinschätzung sowie ein darauf aufbauender adäquater Testplan. Dabei sind die Schlüsselfaktoren hinsichtlich Risikoeinschätzung zum einen die Stakeholder für die Bestimmung der Schadenshöhe und zum anderen technische Faktoren (insb. Quellcode) für die Ermittlung der Eintrittswahrscheinlichkeit. Sowohl bei der Erhebung dieser Schlüsselfaktoren wie auch deren Anwendung fehlen jedoch meist genaue Anleitungen. Infolgedessen und in Anbetracht der Tatsache, dass die Wirtschaftlichkeit in der Literatur höchstens marginal untersucht wurde, lässt sich die zweite Frage („Wie kann dabei der administrative Aufwand des risikobasierten Testens geringgehalten werden?“), überhaupt nicht beantworten. Es fehlen die empirischen Daten aus Fallstudien, um diesbezüglich Auswertungen zu erstellen und Aussagen tätigen zu können.

Allgemein scheint dem Thema „risikobasiertes Testen“ sehr wenig Aufmerksamkeit geschenkt zu werden. Obwohl es bereits seit über 20 Jahren existiert, hält sich die Anzahl der publizierten Arbeiten diesbezüglich in Grenzen. Was verwundert, da Agilität und damit verbunden die Aspekte Zeit und Ressourcen jeher ein präsentes Thema in der Softwareentwicklung sind. Durch die vorhandene Literatur entsteht der Eindruck, als würde versucht, unvereinbare Themen miteinander zu verknüpfen. Im Spannungsfeld des Teufelsquadrats[346] – Funktionsumfang in adäquater Qualität termingerecht mit gegebenen Ressourcen – scheint vor allem an den Ecken Ressourcen (Sprachen, Frameworks und Tools, …), Zeit/Termin (Agilität) und Qualität (effektivere Testverfahren) gezogen zu werden. Besonders mit Blick auf die Aussage, dass branchenübergreifend zirka die Hälfte aller entwickelten Funktionen nicht benötigt werden,[347] stellt sich die Frage, ob hinsichtlich des Funktionsumfanges nicht Potential besteht. So könnten die für die Risikobewertung ermittelten Schadenshöhen dazu dienen, die erhobenen Anforderungen zusätzlich zu priorisieren. Haben Fehler in geforderten Teilen der Software kaum oder keine Auswirkungen, stellt sich grundsätzlich die Frage nach deren Notwendigkeit bzw. Sinnhaftigkeit. Auf diese Weise wäre es mitunter möglich, den generellen Aufwand durch das Wegfallen unnötiger Anforderungen zu verringern.

[346] Siehe Abschnitt 2.1.4 „*Stellgrößen in Softwareprojekten*“
[347] Vgl. Ebert (2019), S. 10

7 Literaturverzeichnis

Aichele, C., Schönberger, M. (2014). *IT-Projektmanagement*. Springer Vieweg. https://doi.org/10.1007/978-3-658-08389-2

Alam, M., Khan, A. I. (2013). Risk-based Testing Techniques: A Perspective Study. *International Journal of Computer Applications*, *65*(1), 33–41.

Alt, R., Auth, G., Kögler, C. (2017). *Innovationsorientiertes IT-Management mit DevOps: IT im Zeitalter von Digitalisierung und Software-defined Business*. Springer Gabler. https://doi.org/10.1007/978-3-658-18704-0

Amland, S. (2000). Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software*, *53*(3), 287–295. https://doi.org/10.1016/s0164-1212(00)00019-4

Bach, J. (1999). Heuristic risk based testing. *Software Testing and Quality Engineering Magazine*(11), 1–10.

Bauer, T., Stallbaum, H., Metzger, A., Eschbach, R. (2008). Risikobasierte Ableitung und Priorisierung von Testfällen für den modellbasierten Systemtest. In H. Korbinian, B. Brügge (Hrsg.), *Software Engineering 2008* (S. 99–111). Gesellschaft für Informatik e. V. http://dl.gi.de/handle/20.500.12116/22220.

Beck, K. (2005). *Extreme Programming: Die revolutionäre Methode für Softwareentwicklung in kleinen Teams* (Studienausg., 2. Nachdruck). Addison-Wesley.

Beck, K. (2012). *Extreme programming explained: Embrace change* (2. Aufl.). *The XP Series*. Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). *Manifesto for Agile Software Development*. Verfügbar unter: https://agilemanifesto.org/ (Zugriff am 09.03.2022).

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, *21*(5), 61–72. https://doi.org/10.1109/2.59

Böhm, J. (2019). *Erfolgsfaktor Agilität: Warum Scrum und Kanban zu zufriedenen Mitarbeitern und erfolgreichen Kunden führen*. Springer Vieweg. https://doi.org/10.1007/978-3-658-25085-0

Bourque, P., Fairley, R. E. (2014). *SWEBOK Version 3.0: Guide to the Software Engineering Body of Knowledge*. IEEE.

Brandes, C., Heller, M. (2017). *Qualitätsmanagement in agilen IT-Projekten – quo vadis?* Springer Vieweg. https://doi.org/10.1007/978-3-658-18085-0

Brandt-Pook, H., Kollmeier, R. (2020). *Softwareentwicklung kompakt und verständlich*. Springer Vieweg. https://doi.org/10.1007/978-3-658-30631-1

Broy, M., Kuhrmann, M. (2013). *Projektorganisation und Management im Software Engineering*. Springer Vieweg. https://doi.org/10.1007/978-3-642-29290-3

Broy, M., Kuhrmann, M. (2021). *Einführung in die Softwaretechnik*. Springer Vieweg. https://doi.org/10.1007/978-3-662-50263-1

Chen, Y., Probert, R. L., Sims, D. P. (2002). Specification-Based Regression Test Selection with Risk Analysis. In *CASCON '02, Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research* (S. 1). IBM Press.

Dahiya, O., Solanki, K., Dhankhar, A. (2020). Risk-based testing: Identifying, Assessing, Mitigating & Managing Risks Efficiently in Software Testing. *International Journal of Advanced Research in Engineering and Technology (IJARET)*, *11*(3), 192–203.

Dijkstra, E. W. (1976). *A discipline of programming*. Prentice-Hall, Inc.

DIN EN ISO 13485:2016-08 (2016). *Medizinprodukte – Qualitätsmanagementsysteme – Anforderungen für regulatorische Zwecke*. Berlin. Beuth.

DIN EN ISO 14971:2020-07 (2020). *Medizinprodukte – Anwendung des Risikomanagements auf Medizinprodukte*. Berlin. Beuth.

Dowie, U. (2009). *Testaufwandsschätzung in der Softwareentwicklung: Modell der Einflussfaktoren und Methode zur organisationsspezifischen Aufwandsschätzung*. o.A.

Droste, O., Merz, C. (2019). *Testmanagement in der Praxis*. Springer Vieweg. https://doi.org/10.1007/978-3-662-49653-4

Ebert, C. (2019). *Systematisches Requirements Engineering: Anforderungen ermitteln, dokumentieren, analysieren und verwalten* (6. Aufl.). dpunkt.verlag.

Erdogan, G., Li, Y., Runde, R. K., Seehusen, F., Stølen, K. (2014). Approaches for the combined use of risk analysis and testing: a systematic literature review. *International Journal on Software Tools for Technology Transfer*, *16*(5), 627–642. https://doi.org/10.1007/s10009-014-0330-5

Erner, M. (2019). *Management 4.0 – Unternehmensführung im digitalen Zeitalter*. Springer Gabler. https://doi.org/10.1007/978-3-662-57963-3

Felderer, M., Haisjackl, C., Breu, R., Motz, J. (2012). Integrating Manual and Automatic Risk Assessment for Risk-Based Testing. In S. Biffl, D. Winkler, J. Bergsmann (Hrsg.), *Lecture Notes in Business Information Processing: Bd. 94. Software Quality. Process Automation in Software Development* (S. 159–180). Springer International Publishing. https://doi.org/10.1007/978-3-642-27213-4_11

Felderer, M., Haisjackl, C., Pekar, V., Breu, R. (2015). An Exploratory Study on Risk Estimation in Risk-Based Testing Approaches. In D. Winkler, S. Biffl, J. Bergsmann (Hrsg.), *Lecture Notes in Business Information Processing: Bd. 200. Software Quality. Software and Systems Quality in Distributed and Mobile Environments* (S. 32–43). Springer International Publishing. https://doi.org/10.1007/978-3-319-13251-8_3

Felderer, M., Ramler, R. (2014). Integrating risk-based testing in industrial test processes. *Software Quality Journal*, *22*(3), 543–575. https://doi.org/10.1007/s11219-013-9226-y

Foidl, H., Felderer, M. (2018). Integrating software quality models into risk-based testing. *Software Quality Journal*, *26*(2), 809–847. https://doi.org/10.1007/s11219-016-9345-3

Fowler, M., Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28–35. http://users.jyu.fi/~mieijala/kandimateriaali/Agile-Manifesto.pdf.

Gadatsch, A., Mangiapane, M. (2017). *IT-Sicherheit*. Springer Vieweg. https://doi.org/10.1007/978-3-658-17713-3

Geis, T., Polkehn, K. (2018). *Praxiswissen User Requirements* (1. Aufl.). dpunkt.verlag.

Geis, T., Tesch, G. (2019). *Basiswissen Usability und User Experience* (1. Auflage). dpunkt.verlag. https://dpunkt.de/produkt/basiswissen-usability-und-user-experience/.

Glaser, C. (2019). *Risiko im Management*. Springer Gabler. https://doi.org/10.1007/978-3-658-25835-1

Großmann, J., Felderer, M., Viehmann, J., Schieferdecker, I. (2020). A Taxonomy to Assess and Tailor Risk-Based Testing in Recent Testing Standards. *IEEE Software*, *37*(1), 40–49. https://doi.org/10.1109/MS.2019.2915297

Gualo, F., Rodriguez, M., Verdugo, J., Caballero, I., Piattini, M. (2021). Data quality certification using ISO/IEC 25012: Industrial experiences. *Journal of Systems and Software*, *176.* https://doi.org/10.1016/j.jss.2021.110938

IEEE 829-2008 (2008). *Software and System Test Documentation*. New York. Institute of Electrical and Electronics Engineers.

ISO/IEC 25012:2008 (2008). *Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model*. Switzerland. ISO.

ISO/IEC 25010:2011 (2011). *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Switzerland. ISO.

ISO/IEC/IEEE 24765:2010 (2010). *Systems and software engineering – Vocabulary*. Switzerland. ISO.

ISO/IEC/IEEE 29119-1:2013 (2013). *Software and systems engineering – Software testing – Part 1: Concepts and definitions*. Switzerland. ISO.

ISO/IEC/IEEE 15288:2015 (2015). *Systems and software engineering – Systems life cycle processes*. Switzerland. ISO.

ISO/IEC/IEEE 12207:2017 (2017). *Systems and software engineering – Software life cycle processes*. Piscataway. Institute of Electrical and Electronics Engineers.

ISO/IEC/IEEE 29148:2018 (2018). *Systems and software engineering – Life cycle processes – Requirements engineering*. Switzerland. ISO.

ISO/IEC/IEEE 29119-2:2021 (2021). *Software and systems engineering - Software testing - Part 2: Test processes*. Switzerland. ISO.

ISTQB. (2018). *Certified Tester Foundation Level Syllabus* (Version 3.1).

Kahneman, D. (2016). *Schnelles Denken, langsames Denken* (19. Auflage). Penguin Verlag.

Kahneman, D., Sibony, O., Sunstein, C. R. (2021). *Noise: Was unsere Entscheidungen verzerrt - und wie wir sie verbessern können* (1. Auflage). Siedler.

Kecher, C., Salvanos, A., Hoffmann-Elbert, R. (2018). *UML 2.5: Das umfassende Handbuch* (6. Aufl.). Rheinwerk.

Keller, H. B. (2019). *Entwicklung von Echtzeitsystemen*. Springer Vieweg.

Kloos, J., Hussain, T., Eschbach, R. (2011). Risk-Based Testing of Safety-Critical Embedded Systems Driven by Fault Tree Analysis. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops* (S. 26–33). IEEE. https://doi.org/10.1109/ICSTW.2011.90

Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Linssen, O., Hanser, E., Prause, C. R. (2017). Hybrid software and system development in practice: waterfall, scrum, and beyond. In R. Bendraou, D. Raffo, H. LiGuo, F. M. Maggi (Hrsg.), *Proceedings of the 2017 International Conference on Software and System Process* (S. 30–39). ACM. https://doi.org/10.1145/3084100.3084104

Leimeister, J. M. (2015). *Einführung in die Wirtschaftsinformatik* (12. Aufl.). Springer Gabler. https://doi.org/10.1007/978-3-540-77847-9

Madauss, B.-J. (2020). *Projektmanagement: Theorie und Praxis aus einer Hand* (8. Aufl.). Springer Vieweg. https://doi.org/10.1007/978-3-662-59384-4

Mayr, H. (2005). *Projekt Engineering: Ingenieurmäßige Softwareentwicklung in Projektgruppen* (2. Aufl.). Fachbuchverl. Leipzig im Carl-Hanser-Verl.

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Mödritscher, G. (2020). Strategieumsetzung und Projektmanagement. In W. Mussnig, G. Mödritscher, U. Liebhart (Hrsg.), *Strategien entwickeln und umsetzen: Speziell für kleine und mittelständische Unternehmen* (3. Aufl., S. 419–435). Linde.

Moskaliuk, J. (2019). *Beratung für gelingende Leadership 4.0*. Springer. https://doi.org/10.1007/978-3-658-23708-0

Ottevanger, I. B. (1999). *A Risk-Based Test Strategy*. IQUIP Informatica B.V.

Ramler, R., Felderer, M. (2015). A Process for Risk-Based Test Strategy Development and Its Industrial Evaluation. In P. Abrahamsson, L. Corral, M. Oivo, B. Russo (Hrsg.), *Lecture Notes in Computer Science. Product-Focused Software Process Improvement* (Bd. 9459, S. 355–371). Springer International Publishing. https://doi.org/10.1007/978-3-319-26844-6_26

Ramler, R., Felderer, M. (2016). Requirements for Integrating Defect Prediction and Risk-based Testing. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* Limassol, Cyprus.

Rau, K.-H., Schuster, T. (2021). *Agile objektorientierte Software-Entwicklung*. Springer Vieweg. https://doi.org/10.1007/978-3-658-33395-9

Redmill, F. (2004). Exploring risk-based testing and its implications. *Software Testing, Verification and Reliability*, *14*, 3–15. https://doi.org/10.1002/stvr.288

Redmill, F. (2005). Theory and practice of risk-based testing. *Software Testing, Verification and Reliability*, *15*(1), 3–20. https://doi.org/10.1002/stvr.310

Rosenberg, L. H., Stapko, R., Gallo, A. (1999). Risk-based Object Oriented Testing. In *Proceedings of the 24th annual Software Engineering Workshop* (S. 1–6). NASA, Software Engineering Laboratory.

Schwaber, K., Sutherland, J. (2013). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Verfügbar unter: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf (Zugriff am 09.03.2022).

Sneed, H. M. (2005). *Software-Projektkalkulation*. Hanser.

Stallbaum, H., Metzger, A., Pohl, K. (2008). An automated technique for risk-based test case generation and prioritization. In *Proceedings of the International Conference on Software Engineering & co-located workshops* (S. 67–70). ACM, 2008. https://doi.org/10.1145/1370042.1370057

Starker, V., Peschke, T. (2021). *Hypnosystemische Perspektiven im Change Management*. Springer Gabler. https://doi.org/10.1007/978-3-662-64359-4

Sternad, D., Mödritscher, G. (2018). *Qualitatives Wachstum*. Springer Gabler. https://doi.org/10.1007/978-3-658-18880-1

Tell, P., Klunder, J., Kupper, S., Raffo, D., MacDonell, S. G., Munch, J., Pfahl, D., Linssen, O., Kuhrmann, M. (2019). What are Hybrid Development Methods Made Of? An Evidence-Based Characterization. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)* (S. 105–114). IEEE. https://doi.org/10.1109/ICSSP.2019.00022

Veenendaal, E. v. (2011). *Practical Risk-Based Testing – Product Risk Management: The PRISMA Method 20112011* (Bd. 2011). EuroSTAR 2011.

Verordnung (EU) 2017/745. *Verordnung des Europäischen Parlaments und des Rates über Medizinprodukte.*

West, D., Gilpin, M., Grant, T., Anderson, A. (2011). Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today. *Forrester Research*, *26*, 1–17.

Witte, F. (2018). *Metriken für das Testreporting*. Springer Vieweg. https://doi.org/10.1007/978-3-658-19845-9

Witte, F. (2019). *Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung* (2. Aufl.). Springer Vieweg. https://doi.org/10.1007/978-3-658-25087-4

Yoon, H., Choi, B. (2011). A test case prioritization based on degree of risk exposure and its empirical study. *International Journal of Software Engineering and Knowledge Engineering*, *21*(02), 191–209. https://doi.org/10.1142/S0218194011005220