Majchrzak, Tim A.; Kuchen, Herbert

Working Paper

Handlungsempfehlungen für erfolgreiches Testen von Software in Unternehmen

Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 127

Provided in Cooperation with:
University of Münster, Department of Information Systems

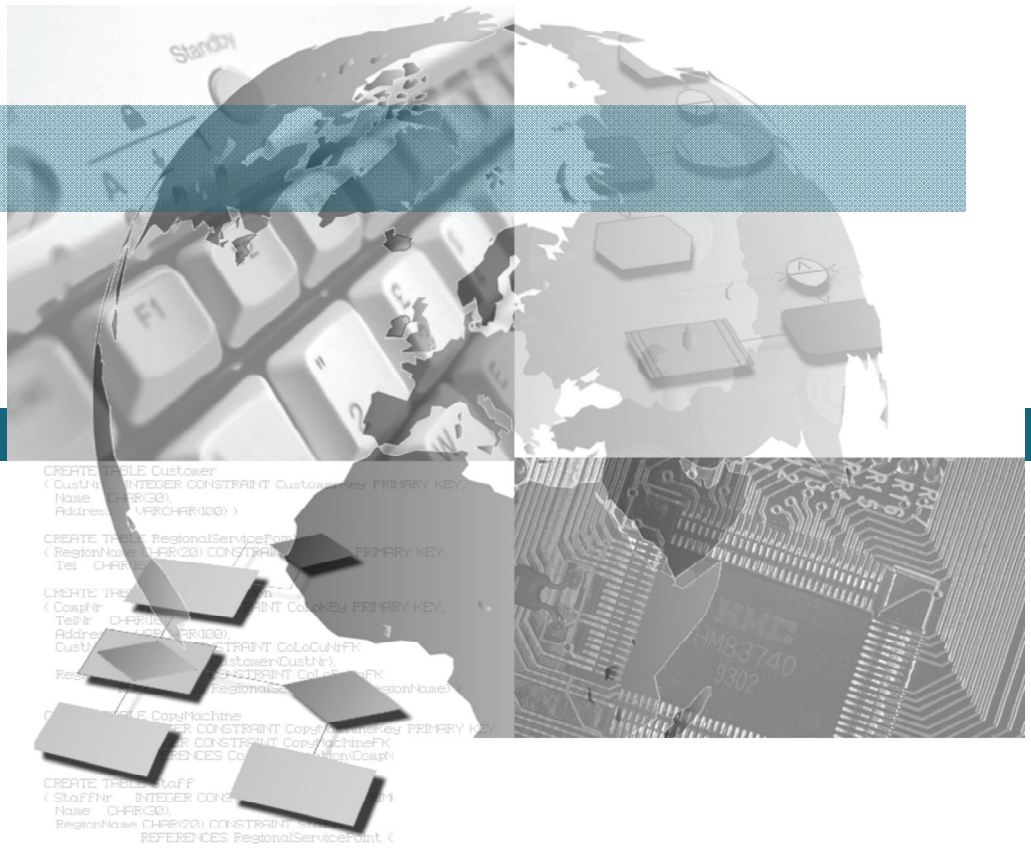
Suggested Citation: Majchrzak, Tim A.; Kuchen, Herbert (2010) : Handlungsempfehlungen für erfolgreiches Testen von Software in Unternehmen, Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 127, Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, Münster

This Version is available at:
https://hdl.handle.net/10419/59552

Westfälische
Wilhelms-Universität
Münster

› Arbeitsberichte des Instituts für Wirtschaftsinformatik

Handlungsempfehlungen für erfolgreiches Testen von Software in Unternehmen



Arbeitsbericht Nr. 127

wissen.leben
WWU Münster

WIRTSCHAFTS
INFORMATIK

Westfälische Wilhelms-Universität Münster

Arbeitsberichte des Instituts für Wirtschaftsinformatik
Herausgeber: Prof. Dr. J. Becker, Prof. em. Dr. H. L. Grob,
Prof. Dr.-Ing. B. Hellingrath, Prof. Dr. S. Klein, Prof. Dr. H. Kuchen,
Prof. Dr. U. Müller-Funk, Prof. Dr. G. Vossen

Arbeitsbericht Nr. 127

Handlungsempfehlungen für erfolgreiches Testen von Software in Unternehmen

Tim A. Majchrzak, Herbert Kuchen

ISSN 1438-3985

Inhaltsverzeichnis

Abbildungsverzeichnis

Abstract / Zusammenfassung

Abstract

Software testing as a main part of the development process is essential for the production of high quality software. We worked with regional enterprises in order to learn about their strengths and weaknesses with regard to testing. Based on the observations we derived recommendations that complement the existing literature. In this work we introduce the project and sketch our research method. We explain a framework for the categorization of the recommendations. Eventually, we exemplarily give four recommendations and offer hints for their implementation. This report supplements the articles on the project that so far have been published.

Zusammenfassung

Softwaretests sind als ein Hauptbestandteil des Entwicklungsprozesses ausschlaggebend für das Erstellen hochqualitativer Software. Wir haben mit regionalen Unternehmen zusammengearbeitet, um ihre Stärken und Schwächen bezüglich des Testens von Software kennenzulernen. Darauf aufbauend haben wir Handlungsempfehlungen erarbeitet, die über bisher in der Literatur zu findende Ratschläge hinausgehen. In diesem Bericht stellen wir das zugrundeliegende Projekt vor und skizzieren die gewählte Forschungsmethodik. Wir führen in einen Ordnungsrahmen für die Kategorisierung der Empfehlungen ein. Schließlich stellen wir exemplarisch vier besonders aussichtsreiche Handlungsempfehlungen vor und geben Implementierungshinweise. Dieser Bericht ergänzt die bisher zum Projekt erschienenen Artikel.

Kapitel 1

Einführung

Bereits seit Jahrzehnten wird an Methoden für die erfolgreiche Entwicklung qualitativer Software gearbeitet. Darunter fallen Ansätze zur organisatorischen Verankerung ebenso wie neue Techniken. Aber auch die Einführung des *Software Engineering* [1] vermochte nicht zu ändern, dass die *Softwarekrise* [2] andauert. Spektakuläre Desaster, die von Softwareproblemen verursacht wurden, aber durch *besseres* Testen wohl hätten verhindert werden können, finden sich immer wieder. Genau untersucht wurde etwa der Absturz der Ariane 5 [3] oder die fehlgeschlagene Mission des Mars Climate Orbiters [4]. Aktuelle Beispiele sind die (mittlerweile behobenen) Probleme des Mautsystems Toll Collect, oder die sich verzögernde Einführung der elektronischen Gesundheitskarte. Leider finden sich zahlreiche Hinweis auf scheiternde Softwareprojekte aller Größen und Arten [5]. In einigen besonders schlimmen Fällen sind Menschen leidtragend betroffen [6]; fast immer haben die Probleme ernste wirtschaftliche Konsequenzen.

Das Testen von Software kann helfen, Probleme zu vermeiden. Softwarequalität ergibt sich als Kombination von Softwareentwicklung und Testen [7]. Eine Studie von Pierre Audoin Consultants zufolge erkennen Unternehmen die hohe Bedeutung des Testens, gleichzeitig sei es aber häufig „schlecht organisiert“ [8]. Insbesondere quantitative Aussagen könnten die wenigsten Unternehmen treffen, 65% von ihnen waren etwa die beim Testen anfallenden Kosten unklar.

Forschung zur Verbesserung der Softwarequalität wird dringend benötigt. Dabei streben wir eine enge Zusammenarbeit mit der Industrie an. Wissen muss zwischen Forschung und Praxis ausgetauscht und in gebündelter Form in die Unternehmen getragen werden. Denn auch die Ausbildungssituation ist problematisch. In einer Studie zur Informatik-Ausbildung bescheinigen Praktiker nur knapp 10% der Absolventen ausreichende Kenntnisse des Testens; 58% verfügten über unzureichendes oder gar keine Wissen [9].

In einem Projekt mit regionalen Unternehmen haben wir versucht herauszufinden, *was* qualitative Softwareentwicklung erfolgreich macht. Dazu haben wir den Status quo erarbeitet und danach erfolgreiche Herangehensweisen und Methoden mit den Unternehmen diskutiert. Daraus entstanden schließlich Handlungsempfehlungen. Manche dieser Empfehlungen erscheinen auf den ersten Blick offensichtlich. Und doch werden sie nicht konsequent umgesetzt oder verständlich in der Literatur beschrieben. Es ist daher gerade im Bereich der Softwareentwicklung wichtig, auf *kleine* Dinge zu achten [10] und konsequent vorzugehen.

Dieser Bericht ist wie folgt aufgebaut. Kapitel 2 erläutert die Hintergründe des Projekts und skizziert unsere Forschungsmethodik. Den Ordnungsrahmen zur Kategorisierung der Handlungsempfehlungen erläutern wir in Kapitel 3. In Kapitel 4 stellen wir vier Empfehlungen vor. Schließlich ziehen wir in Kapitel 5 ein Fazit. Weitere Handlungsempfehlungen aus dem Projekt wurden bereits in [11] und [12] beschrieben.

Kapitel 2

Hintergrund und Forschungsmethodik

Im Münsterland haben sich viele Unternehmen angesiedelt, die Software entwickeln bzw. sie zur Ermöglichung ihrer Geschäftsprozesse benötigen. Bei letzteren handelt es sich um größere Finanzdienstleister. Die Unternehmen sind Mitglieder der Industrie- und Handelskammer, die das *Institut für Angewandte Informatik* (IAI) unterstützt. Es ist der WWU Münster angeschlossen und vereint die Arbeit von Ökonomen und Informatikern.

Regelmäßige Gespräche zwischen Forschern und Unternehmen führten zur Beobachtung, dass allgemeine Unzufriedenheit im Bezug auf Softwaretests herrschte. Während die Unternehmen sowohl eine Erhöhung der Softwarequalität als auch Kostensenkungen anstrebten, waren die Wege zu diesen Zielen unbekannt. Auch fehlten Kapazitäten, um Änderungen auszuprobieren. Nichts desto trotz wird im Münsterland sehr erfolgreich Software entwickelt. Dies lässt zwei Schlüsse zu [11]:

- Alle Unternehmen stehen vor Problemen und haben Prozesse, die Spielraum für Verbesserungen lassen.
- Jedes von ihnen hat individuelle Stärken entwickelt, die der Entwicklung qualitativer Software dienlich sind.

Als Konsequenz wurde das IAI-Projekt zum Testen von Software initiiert. Neben dem Status quo sollten die Stärken der Unternehmen identifiziert werden, um konkrete Handlungsempfehlungen (“best practices”) ableiten zu können. Wir erwarteten dabei, komplementäre Stärken zu finden, so dass sich für alle Unternehmen verwertbare Hinweise ergeben. Während die Heterogenität der Unternehmen überhaupt ermöglicht, eine Vielzahl hilfreicher Beobachtungen zu machen, führt sie auch zu Schwierigkeiten: Nicht jede Empfehlung ist für jedes Unternehmen geeignet. Folglich entwickelten wir einen Ordnungsrahmen (vgl. Kapitel 3), der Hilfestellung bei der Auswahl der Empfehlungen bietet.

Die Forschungsmethodik sollte akademischen Ansprüchen genügen, gleichzeitig aber sollten Ergebnisse für Unternehmen direkt nutzbar sein. Daher wurde ein gestaltungsorientierter Ansatz gewählt, der als *design science* bezeichnet wird [13] und auf praktische Fragestellungen zurückgreift [14]. Ungelöste Probleme werden auf neue Arten und Weisen angegangen, um sie effizienter oder effektiver zu lösen [15]. Unser Ziel war weder, den *idealen* Testprozess, noch ein Standardwerk zur Lösung von Testproblemen zu schaffen. Vielmehr sollen Empfehlungen für *typische* Probleme erarbeitet werden.

Mithilfe einer einfachen Studie, etwa einer Umfrage, ist es kaum möglich, erfolgreiche Vorgehensweisen zu identifizieren. Um die Probleme aus Sicht der Teilnehmer wahrnehmen zu können, haben wir

semi-strukturierte Experteninterviews anhand eines groben Interview-Leitfadens durchgeführt (siehe Anhang A.1). Dieser qualitative Ansatz ermöglichte uns, zu verstehen, *wie* in den Unternehmen getestet wird. In den Interviews wurden Stärken und Schwächen und schließlich erfolgreiche Vorgehensweisen mit den Teilnehmern erörtert.

Die von uns ermittelten Handlungsempfehlungen konkurrieren nicht mit theoretischer oder praktischer Literatur zu Softwarequalität. Wir geben daher zahlreiche Hinweise auf weiterführende Literatur und versuchen den vorhandenen Kenntnisstand zu ergänzen. Denn nicht alle in Unternehmen auftretenden Probleme werden in der Literatur behandelt. Dazu kommt, dass Wissen häufig nicht zugänglich bzw. problemadäquat aufbereitet ist.

Zu Beginn des Projekts wurden Unternehmen kontaktiert und Führungskräfte sowie technisch ausgebildete Angestellte für Interviews ausgewählt. Während in kleinen Unternehmen *ein* Termin für Interviews ausreichte, haben wir große Unternehmen mehrfach besucht und Gespräche mit verschiedenen Mitarbeitern geführt. Dabei wurde geklärt, *wer* für Tests verantwortlich ist, *wann* getestet wird, *was* dabei eingeschlossen ist (z. B. die Programmoberfläche), *welche* Methoden verwendet werden, *wie* Testen grundsätzlich angegangen wird und ob *Testwerkzeuge* zum Einsatz kommen.

Zunächst wurde der Status quo erarbeitet; er ist nicht Teil dieses Berichts. Danach haben wir mit den Teilnehmern diskutiert, auf welche generellen Probleme sie gestoßen sind und welche erfolgreichen Vorgehensweisen im Unternehmen entwickelt wurden. Dieser Schritt beinhaltete auch, wünschenswerte Verbesserungen zu identifizieren. Die Interviews waren sehr offen gestaltet, um Gedanken auszutauschen und eine große Zahl interessanter Ansätze zu besprechen. In der folgenden Analyse haben wir die Ergebnisse verdichtet, den Status quo gezeichnet und Handlungsempfehlungen abgeleitet. Dazu gehörte vor allem auch die Überlegung, unter welchen Bedingungen sie jeweils einsetzbar sind bzw. welcher Voraussetzungen sie bedürfen.

Kapitel 3

Ordnungsrahmen

Aufgrund der Komplexität des Testens und abweichender Gegebenheiten in Unternehmen ist eine Kategorisierung von Empfehlungen unabdingbar. Ihren vollen Nutzen erreichen sie nur, wenn Nutzungsmöglichkeiten und Voraussetzungen klar sind. Aus diesem Grund verwenden wir einen von uns entwickelten Ordnungsrahmen [11].

Jeder Empfehlung ist ein *Anspruch* zugeordnet, der den Umfang organisatorischer Veränderungen für die Umsetzung beschreibt. *Grundsätzliche* Empfehlungen sollten von jedem Unternehmen umgesetzt werden. Werden sie als *fortgeschritten* eingeordnet, ist mit deutlich erhöhtem Aufwand zu rechnen. Ein *Zielzustand* schließlich steht für Ideale, die nur mit größtem Aufwand erreicht werden können. In der Regel erfordern sie die Initiierung eines Prozesses kontinuierlicher Verbesserungen. Unternehmen müssen abwägen, ob sie dieses Ziel anstreben wollen, können aber langfristig erhebliche Erfolge erzielen.

Die *Projektgröße* ist in drei Kategorien unterteilbar. *Kleine* Projekte werden häufig von einem Team bearbeitet, das Entwicklung und Testen vereint. Für *mittlere* Projekten werden diese Aufgaben in der Regel auf mindestens zwei Teams aufgeteilt. *Große* Projekte sind anders organisiert und weisen häufig hunderte beteiligte Mitarbeiter auf. Oftmals sind zudem weitere Abteilungen involviert. Bei der Einordnung der Projektgröße sollte der Charakter des Projekts betrachtet werden, weniger die tatsächlichen Mitarbeiterzahlen.

Grob lassen sich zwei *Arten* von Softwareprodukten unterschieden. *Individualsoftware* wird für einzelne Kunden gemäß vertraglicher Absprachen implementiert. Software für den *Massenmarkt* wird für einen größeren Kundenkreis entwickelt und über einen längeren Zeitraum angeboten. Hierfür sind Regressionstests besonders wichtig.

Weiterhin wird die Zahl der *Releases* unterschieden. Möglich sind *ein*, *einige* und *regelmäßige* Releases. Letzteres bedeutet, dass ein Produkt über längere Zeit gepflegt wird.

Schließlich wird das Testen in *Stufen* unterteilt. Der Literatur folgend [16] werden *Komponenten-*, *Integrations-*, *System-* und *Abnahmetests* unterschieden.

Wie in Abb. 3.1 dargestellt, bilden Anspruch und Stufe des Testens eine Matrix. Ein Häkchen zeigt an, dass eine Handlungsempfehlung für die entsprechende Kombination Bedeutung hat. Wird es in Klammern dargestellt, ist die Implementierung für diese Kombination optional. Die anderen drei Determinanten sind als Balken dargestellt. Eine dunkelgraue Schraffur zeigt, dass die Empfehlung unter dieser Bedingung gilt. Ein Farbverlauf weist auf eine eingeschränkte Relevanz hin. In Abb. 3.1 ist eine fiktive Verwendung dargestellt (vgl. [11]):

- Die Empfehlung gilt für Integrations-, System- und Abnahmetests. Der Anspruch ist fortgeschritten. Die zusätzlichen Häkchen bei *grundsätzlich* bedeuten, dass die Umsetzung nicht direkt in vollem

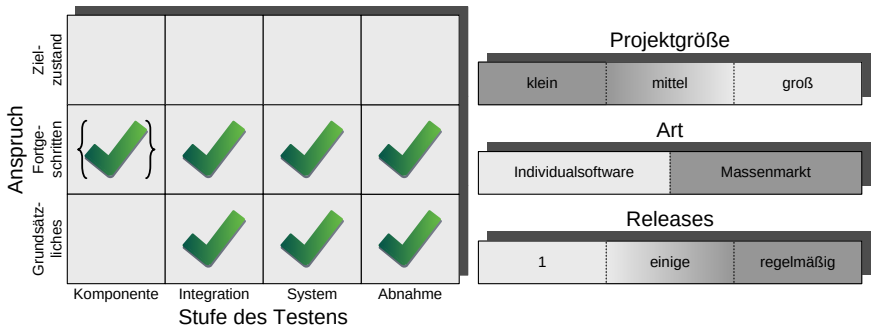


Abbildung 3.1: Beispielhafte Anwendung des Ordnungsrahmens

Umfang erfolgen muss. Das Häkchen in Klammern zeigt, dass für den Komponententest geringfügige Vorteile realisierbar sind.

- Die Handlungsempfehlung bezieht sich auf kleine und mittelgroße Projekte. Der Übergang der Färbung bedeutet, dass die Relevanz für solche Projekte noch gegeben ist, die kleinen Projekten ähnlich sind.
- Ferner solle es sich um in einigen Releases oder dauerhaft erscheinende Massenmarkt-Software handeln. Denkbar wäre auch ein Farbverlauf für die individuelle Entwicklung. Das hieße, sie würde ebenfalls profitieren, wenn auch nicht in so hohem Maße.

Für die Wahl umzusetzender Handlungsempfehlungen müssen Unternehmen weitere Prüfungen vornehmen. Der Ordnungsrahmen hilft dabei, einen Überblick zu erhalten und das Potential einzelner Empfehlungen einzuschätzen.

Kapitel 4

Handlungsempfehlungen

4.1 Projektbezug und Anreizsysteme

Auf den ersten Blick scheint es selbstverständlich, dass Software im Rahmen von Projekten entwickelt und getestet wird. Dies impliziert einen zugrundeliegenden Testprozess [17] und begünstigt ein Testmanagement [18]. Unserer Beobachtung nach werden viele Softwareprodukte nicht konsequent projektbezogen entwickelt. Wir empfehlen dies aber dringend, um die Qualität der Produkte zu erhöhen. Mit der notwendigen Strukturierung des Testens sollte die Einrichtung eines Anreizsystems einhergehen. Diese Empfehlung gilt für alle Unternehmen. Vorteile zeigen sich in allen Testphasen und sind mit geringem Aufwand zu erreichen. Ein sehr weitreichender Ausbau ist möglich (Abb. 4.1).

Für projektbezogenen Softwaretests sollte ein Testprozesses im Rahmen eines Qualitätsmanagements [19] definiert werden. Dabei sollten alle Schritte des Testens zumindest grob festgelegt sein. Diese Forderung widerspricht *agilen* Herangehensweisen wie der testgetriebenen Entwicklung [20] nicht. Festgelegt werden soll nicht im Detail, *wann* und *wie* zu testen ist, sondern *was* grundsätzlich zu erfolgen hat. Nur so kann unsystematisches Testen verhindert werden. Wir empfehlen, auch etwaige Abweichungen vom Entwicklungsplan im Voraus zu berücksichtigen. In der Praxis wird auf zusätzliche Anforderungen bei gleichbleibenden Fristen reagiert, indem Kapazitäten für das Testen gestrichen und der Entwicklung zugewiesen werden. Dies gilt es zu verhindern. Neben diesen Grundsätzen ist ein komplexer Ausbau der Entwicklungsprojekten zugrundeliegenden Testprozesse möglich. Anhaltspunkte hierzu liefert die Literatur zu Softwarequalität [21, 22], pragmatische Werke zum Testen [23] sowie aktuell diskutierte erfolgreiche Vorgehensweisen (vgl. [11]). Wesentlich ist, dass jedes Unternehmen im Laufe der Zeit *seinen* Prozess entwickelt und ihn nicht zu sehr generischen Empfehlungen von oberflächlichen Unternehmensberatungen oder an Vorgaben der Testwerkzeuge anpasst.

In Projekten sollten Entwicklung und Testen gleichberechtigt sein. Wenn dedizierte Tester zum Einsatz kommen, können sie von Entwicklern als störend empfunden werden. Ihre Aufgabe ist aber nicht, Entwickler auf ihr *Versagen* hinzuweisen. Vielmehr sollten Tester so in Projekte integriert werden, dass Entwickler sie als zuverlässige Hilfe für die Entwicklung qualitativer Software wahrnehmen. Die daraus resultierende Unterstützung durch die Entwickler fördert das Testen ungemein. Durch die Einbindung in das Projekt identifizieren sich Tester mit diesem. Als Zielzustand ist ein Zusammenspiel von Analysten, Entwicklern und Testern wünschenswert. Dies ist nur durch Auflösung potentieller Konflikte dieser Rollen möglich. Dazu ist ein professioneller (eventuell agiler [24]) Umgang mit Anforderungen nötig [25, 26], der mit Testern abgestimmt werden sollte [11]. Ein Vorschlag aus der Praxis ist, Testen und Anforderungsanalyse zu verknüpfen [27].

Konflikte drohen Entwicklern auch bei kurzen Projektlaufzeiten und kleinen Entwicklungsteams. Häufig kommen keine Tester zum Einsatz und Entwickler müssen diese Aufgabe übernehmen. Wenn Kosten und

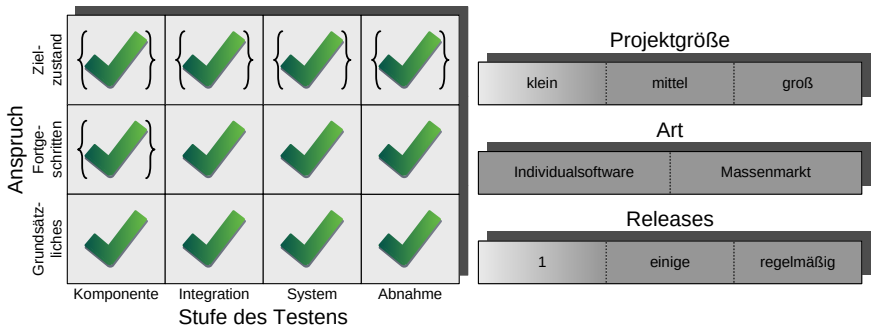


Abbildung 4.1: Einordnung des Projektbezugs und der Anreizsysteme

der Fertigstellungstermin vorgegeben sind, ist die zu erreichende Qualität allerdings ebenfalls fix. Entwickler müssen dann entscheiden, ob sie den Termin überschreiten, Funktionen weglassen, oder weniger Testen. An dieser Stelle versagt häufig das Anreizsystem: Ein nachlässiges Testen wird am wenigsten sanktioniert, schmälert aber nachhaltig die Qualität der entwickelten Software. Eine Lösung kann nur organisatorisch erfolgen. Eine Rollentrennung sollte verhindern, dass Entwickler überhaupt zwischen zwei Zielen entscheiden [11]. Ihr Zeitbudget als Tester sollte fix sein und darf nicht „unter der Hand“ geändert werden. Qualität sollte zur Maxime werden – ganz offenbar nötige Tests, die mehr Zeit als zunächst veranschlagt erfordern, dürfen nicht zu Konsequenzen für Entwickler und Testern führen, die gewissenhaft ihrer Aufgabe nachgehen. Ein wiederum projektbezogener Einsatz der Mitarbeiter und eine Planung ihrer Aufgaben und Arbeitszeit hilft, solche Dilemmata zu vermeiden.

Die Definition von Prozessen soll helfen, das Auftreten zahlreicher Probleme unwahrscheinlich werden zu lassen. Insbesondere müssen alle Anreize – auch informelle – auf den Prüfstand gestellt werden. Stehen sie mit der Ausrichtung auf Softwarequalität in Konflikt, sind Änderungen nötig. Proaktive Maßnahmen können ungünstige Situationen verhindern. Ein Beispiel hierfür ist die Auswertung von Kennzahlen im Rahmen eines Test-Controllings [11], mit dem sich drohende Probleme frühzeitig erkennen lassen. Maßnahmen sollten auf die Verankerung des Qualitätsbezugs in der Unternehmenskultur abzielen. Zu Not müssen organisatorische Veränderungen per Geschäftsanweisung eingeleitet werden. Richtlinien sollten so gestaltet werden, dass sie ein einheitliches Vorgehen unterstützen, aber Innovationen und Prozessoptimierungen nicht verhindern.

4.2 Abbau systematischer Komplexität

Die Komplexität der Softwareentwicklung kann insbesondere bei umfangreichen Projekten oder stetig weiterentwickelten Produkten sehr hoch werden. Probleme bereiten dann die Planung, komplexe Abhängigkeiten, die Anbindung weiterer Systeme sowie gewachsene Architekturen. Besteht ein Softwaresystem aus zahlreichen Komponenten, wird es zunehmend schwierig, den Überblick über alle Komponenten und ihre Schnittstellen zu behalten. Eventuell müssen Gegebenheiten beim Kundeneinsatz im Hinterkopf behalten werden. Software kann unter Umständen nicht ausgetauscht werden, wenn unklar ist, zu welchen Systemen es Schnittstellen gibt. Auch technische Hürden wie inkompatible Hard- oder Software sowie unterschiedliche Programmiersprachen bereiten Probleme. Zum Teil ist für alte Systeme kein Quelltext mehr vorhanden, so dass die Fehleranalyse äußerst schwierig ist. Auch fehlt häufig eine ausreichenden Dokumentation der Systeme.

Komplexität erschwert nicht nur die Weiterentwicklung; sie behindert auch das Testen. Über erhöhte Kosten hinaus wird schlimmstenfalls das Erreichen des gewünschten Qualitätsniveaus mit der Zeit unmöglich. Erfahrungen in der Praxis zeigen, dass die Komplexität in Projekten schneller steigt als ihr Umfang. Wir empfehlen allen Unternehmen, die in größeren Projekten Software für den Massenmarkt entwickeln, an der Verringerung systematischer Komplexität zu arbeiten. Vorteile zeigen sich vor allem in den späten Testphasen, sind allerdings erst mit einigem Aufwand zu erreichen (Abb. 4.2).

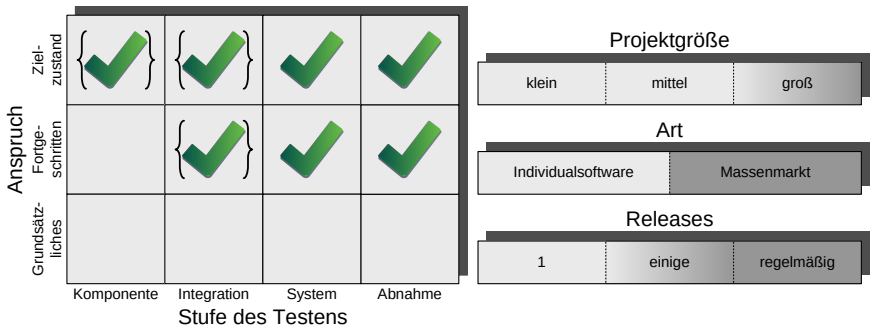


Abbildung 4.2: Einordnung des Abbaus systematischer Komplexität

Ein wesentlicher Beitrag zur Strukturierung und damit dem Komplexitätsabbau leistet eine geeignete Abstraktion. Die Entwicklung sollte in Einheiten zerlegt werden, die beherrschbar bleiben. Kontakt zu weiteren Einheiten sollte ausschließlich über definierte Schnittstellen erfolgen. Die Vorteile lassen sich an einem einfachen Beispiel illustrieren. Eine datengetriebene Software wird geplant, die Informationen für die Darstellung in Browsern aufbereitet. Ohne technische Details wie Entwurfsmuster zu diskutieren, lässt sich sagen, dass es sinnvoll ist, sie in drei Teilen (Datenhaltung, Geschäftslogik und Oberfläche) zu implementieren. Dazu werden auch Mitarbeiter benötigt, die ohne detaillierte Kenntnisse eines der Teile den Überblick bewahren und auf die Einhaltung der Schnittstellen achten. Auch können Tester dann aufgabenbezogen eingesetzt werden. Auf Oberflächentests spezialisierte Tester prüfen Benutzerschnittstellen z. B. deutlich effizienter und eventuell sogar effektiver als Tester, die ein monolithisches Gesamtsystem prüfen müssen.

Bei großen Systemlandschaften kann stärker abstrahiert werden, indem Verbünde von Systemen als Einheiten gesehen werden. Dies erhöht zwar den Verwaltungsaufwand, schafft aber die Grundlage, um überhaupt einen Überblick zu behalten. Denn einzelne Mitarbeiter müssen nur noch die Komplexität ihres Teilbereichs beherrschen. An Stelle einer umfassenden Kenntnis aller Systeme tritt ein grober Überblick sowie das Wissen um die jeweiligen Ansprechpartner. Auf dieselbe Art lassen sich auch größere Projekte beherrschen.

Besonders wichtig ist eine klare Dokumentation von Verantwortlichkeiten und beteiligten Mitarbeitern. In stetig weiterentwickelten Projekten kommen gewöhnlich in jeder Iterationsstufe andere Tester zum Einsatz. Viel sinnvoller wäre es, hierfür die Erfahrungen aus der Vergangenheit zu nutzen. Treten etwa Schnittstellenprobleme auf, sollten die oben angesprochenen Verwaltungsmitarbeiter direkt zuständige Tester und Entwickler benennen können. Denn Fehler müssen nicht auf geänderte Module zurückzuführen, sondern könnten nur bisher nicht in Erscheinung getreten sein. Wurden Abhängigkeiten nicht dokumentiert, können solche Probleme nur mit größtem Aufwand gelöst werden.

Die Dokumentation kann mit einfachen Tabellen beginnen (vgl. auch [18]), sollte aber in ein ganzheitliches Verwaltungssystem münden. Für umfangreiche Landschaften empfehlen sich *Systemlandkarten*, die Systeme und Abhängigkeiten abbilden. Diese sollten im Sinne der Abstraktionsstufen skalierbar und mit Mitarbeiterdaten annotiert sein. Für die Komplexitätsbeherrschung muss die Dokumentation unbedingt bei einem Projektstart begonnen und konsequent fortgeführt werden. Auch sollte darauf geachtet werden, nur nötige Informationen zu erfassen und Datenbestände aktuell zu halten.

Auf eine umfangreiche Dokumentation kann ein Risikomanagement aufgebaut werden; die Möglichkeiten sind gewaltig [28, 29, 30]. So könnte ein entsprechendes System vor potentiellem Wissensabflusses warnen, wenn nur noch wenige Mitarbeiter Kenntnisse einer Funktion haben. Die nächste Ausbaustufe stellt ein *Betriebliches Kontinuitätsmanagement* (Business Continuity Planning) dar [31]. Es soll Problemen vorbeugen und selbst im Falle erheblicher Schwierigkeiten den Betrieb aufrechterhalten.

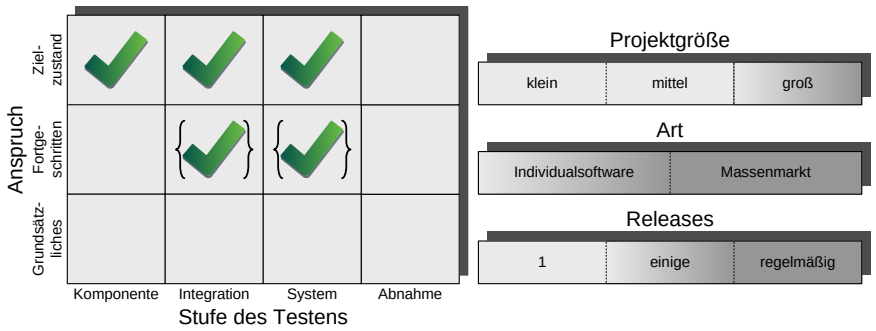


Abbildung 4.3: Einordnung der Anpassung von Werkzeugen an die Prozesse

4.3 Anpassen von Werkzeugen an die Prozesse

In den meisten Fällen arbeiten Testwerkzeuge technikgetrieben. Selbst wenn sie sich anpassen lassen, geben sie eine grundsätzliche Art und Weise vor, auf die sie genutzt werden. Dies impliziert, dass Prozesse an die Werkzeuge angepasst werden. In vielen Fällen lassen sie sich sonst nicht sinnvoll nutzen bzw. der Anpassungsaufwand wäre zu hoch. Diese Beobachtung steht im Widerspruch zu den ersten beiden Empfehlungen. Projektteilnehmer mit definiertem Testprozess wiesen uns daher darauf hin, dass dringend davon abzuraten sei, Prozesse an Werkzeuge anzupassen. Vielmehr müssten diese an die vorherrschenden Prozesse anpassbar sein. Wir empfehlen, Werkzeuge anhand ihrer Anpassbarkeit auszuwählen und ein besonderes Augenmerk darauf zu legen, gut funktionierende Prozesse nicht unnötigerweise zu verändern. Dies ist bei großen Projekten mit zumindest einigen Releases hilfreich. Vorteile lassen sich vor allem dann erzielen, wenn Unternehmen ihre Testprozesse kontinuierlich optimieren (Abb. 4.3).

Grundsätzlich ist immer zu erwarten, dass die Einführung neuer bzw. die Aktualisierung bestehender Werkzeuge zu Änderungen führt. Diese ergeben sich z. B. dadurch, dass neue Testphasen eingeführt werden oder umfangreiche Funktionen hinzukommen. Diese Anpassung ist normal und erstrebenswert. Es sollten allerdings *optimale* Prozesse angestrebt werden. Eine Adaptierung der durch Werkzeuge vorgegebenen Vorgehensweisen kann nur Ausgangspunkt der Überlegungen sein. Denn diese sind nur in wenigen Fällen auf das Unternehmen abgestimmt. Neben einer durchdachten Einpassung des Werkzeugs in den Testprozess sollten daher auch Evaluationen geplant werden. So können die nach einiger Nutzung gesammelten Erkenntnisse für weitere Optimierungen verwertet werden.

Prozessvorgaben liegen häufig in technischen Details begründet. Der Erfahrung aus dem Projekt nach lassen sich nur wenige Testwerkzeuge intuitiv nutzen. Diesem Problem lässt sich begegnen, indem bereits bei der Auswahl möglicher Werkzeuge auf die Anpassbarkeit geachtet wird. Schritte zum Einpflegen eines Testfalls etwa können dann so gestaltet werden, wie es in den Arbeitsablauf der Mitarbeiter passt. Finden sich in der Dokumentation, in Demonstrationsmaterialien oder bei Vorstellungen durch den Hersteller Hinweise auf starre Vorgaben, sollte genau geprüft werden. Besonders hinderlich ist die fehlende Unterbrechbarkeit von Prozessen sowie ein Mangel an Schnittstellen. Problematisch sind etwa Werkzeuge zur Testausführung, die sich nicht mit denen zur Dokumentation integrieren.

Die Anpassbarkeit eines Werkzeugs kann auf mehreren Stufen gegeben sein. Im technischen Sinne ist es etwa sinnvoll, wenn sich Tests während der Ausführung unterbrechen und die Zwischenergebnisse speichern lassen. Auch sind umfangreiche Im- und Exportschnittstellen bzw. die Möglichkeit der Echtzeitanbindung zu begrüßen. Bezüglich der Benutzbarkeit ist eine frei konfigurierbare Oberfläche sinnvoll. Die Anpassbarkeit sollte dabei der Komplexität des Werkzeugs Rechnung tragen; eine Anpassbarkeit auf rein technischer Ebene (etwa durch Skripting) ist nur für kleine Werkzeuge akzeptabel. Sehr geeignet sind Werkzeuge, die über Plug-ins erweitert werden können. Besonders hilfreich ist es ferner, wenn sich Werkzeuge selbst als Plug-in in integrierte Entwicklungsumgebungen (IDEs) einfügen lassen. Die Gestaltung durchgehender Prozesse wird so vereinfacht.

Die Erfahrungen im Projekt haben bestätigt, dass ein gut kalkulierter Aufwand für die Anpassung von Werkzeugen eher zu rechtfertigen ist, als die Anpassung der eigenen Prozesse. Zudem kann beim

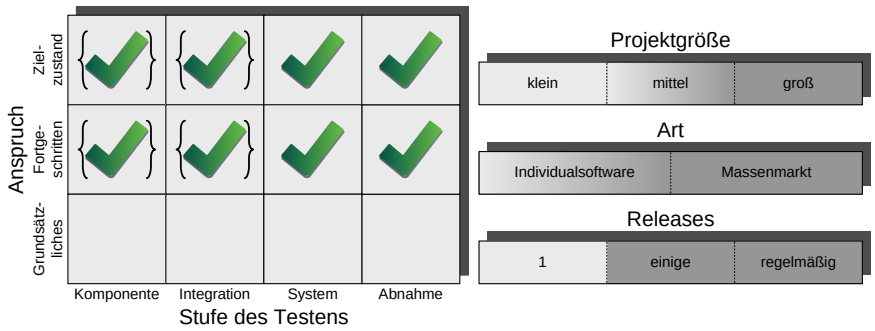


Abbildung 4.4: Einordnung des Aufbaus eines Gesamtkonzepts

Einpassen der Werkzeuge der eigene Testprozess reflektiert und verbessert werden. Langfristig lohnen sich selbst kleine Maßnahmen. Unregelmäßigkeiten durch kaum anpassbare Werkzeuge hingegen können die Produktivität senken. Zu bedenken ist immer auch der entgangene Nutzung durch die Möglichkeit, *besser* zu testen und die Qualität der entwickelten Software zu erhöhen.

4.4 Aufbau eines Gesamtkonzepts

Softwaretests lassen sich in verschiedene Phasen aufteilen [16]. Sie erfordern unterschiedliche am Prozess beteiligte Rollen. Zu den wichtigsten Erkenntnissen des Projektes gehört, dass Testen immer ganzheitlich gesehen werden sollte. Ein auf das Unternehmen zugeschnittenes Gesamtkonzept sollte ebenso selbstverständlich werden wie ein dazugehöriges Stufenkonzept. Die Vorteile zeigen sich bereits für Projekte mittlerer Größe und vor allem in den späteren Phasen des Testens. Der Schwerpunkt liegt bei stetig weiterentwickelter Software. Aber auch individuelle Projekte können profitieren (Abb. 4.4).

Das Stufenkonzept wird in der Praxis auch als *Staging-Konzept*, *Staging-Plan* oder schlicht *Staging* bezeichnet. Es untergliedert den Testprozess in Abschnitte, die den dem Ordnungsrahmen zugrunde gelegten Phasen entsprechen (siehe Kapitel 3). Wichtig ist, das Stufenkonzept an die Bedürfnisse des Unternehmens anzupassen und bei Bedarf zusätzliche Phasen für unterschiedliche Projekte vorzusehen. Denkbar sind etwa optionale Phasen für Last- und Stresstests, die während des Integrations- oder Systemtests angesiedelt werden. Auf diese Weise kann eine Leistungsüberprüfung fest vorgesehen werden.

Ferner ist eine feingliedrigere Unterteilung möglich. Ein Beispiel aus der Praxis ist die Aufteilung des Integrationstests. In der ersten Phase testet noch der Entwickler die Integration eines von ihm erstellten Moduls in das Gesamtsystem. In den zweiten Phase findet der eigentliche Integrationstests durch dedizierte Tester statt. Für jede einzelne Testphase sollten auf das Unternehmen, seine Kultur und die zur Verfügung stehenden Mitarbeiter zugeschnittene Vorgaben getroffen werden. Sinnvoll sind sowohl grundsätzliche Richtlinien als auch optionale Empfehlungen. Konkrete Vorschläge sprengen den Rahmen dieses Berichts, exemplarisch sei auf die Literatur zum Systemtest [32] verwiesen.

Die Phasen müssen unbedingt einander angeschlossen werden. Die Dokumentation ihrer Ergebnisse ermöglicht einen Lernprozess und unterstützt Mitarbeiter in der folgenden Phase. Mit einer formellen Freigabe kann nach jeder Phase bescheinigt werden, dass alle zu diesem Punkt realistischen Anforderungen erfüllt werden. Sie sichert somit eine bestimmte Qualität formell und nachprüfbar zu. Bei Mängeln wird eine Nachbesserung nötig.

Im Rahmen des Stufenkonzepts sollte festgelegt werden, *welche* Mitarbeiter bzw. *welche* Rollen in *welcher* Phase *welche* Aufgabe übernehmen, durch *wen* die Freigabe erteilt wird, *welche* Art von Informationen dokumentiert und weitergegeben wird, und *wie* bei etwaigen Probleme zu verfahren ist. So ist der Rückgriff auf Mitarbeiter früherer Phasen hilfreich. Wir empfehlen die genaue Konzeption der Stufen und strikte Einhaltung der Freigaben.

Es ergeben sich umfangreiche Ausbaumöglichkeiten. Eine doppelte Freigabe durch Techniker *und* Kundenbetreuer oder die Fachabteilung kann in späteren Phasen das Qualitätsniveau steigern. Des Weiteren bietet es sich an, zu verwendende Testsysteme zu spezifizieren und somit für kontrollierte Bedingungen zu sorgen. Aufgrund der innewohnenden Komplexität muss ein Stufenkonzept nach und nach aufgebaut und stetig optimiert werden. Idealerweise wird es integraler Bestandteil der Softwareentwicklung im Unternehmen.

Diesem Gedanken folgend sollte das Testen im Allgemeinen, bzw. sogar die gesamte Softwareentwicklung in ein Gesamtkonzept eingebettet werden. Ansätze hierzu finden sich in der neueren Literatur zur Gestaltung von Testprozessen [33]. Die Entwicklung von Insellösungen in getrennten Teams und eine stiefmütterliche Betrachtung des Testens ist abzulehnen. Testen als eine Verpflichtung zur Softwarequalität sollte fest verankert und selbstverständlich werden. Je selbstverständlicher es wird, desto höher wird der Beitrag zur Erreichung der Qualitätsziele. Das Gesamtkonzept sollte Entwickler und Tester nicht einschränken und Kreativität bremsen. Nichts desto trotz ist eine Formalisierung des Testen, eine Einigung auf gemeinsame Methoden, Werkzeuge und eine gemeinsame Sprache unerlässlich. Nur so kann Testen effektiv gestaltet werden. Eine Kultur, in der Veränderungen gescheut werden, weil der Status quo (noch) *akzeptabel* ist, gilt es zu vermeiden.

Kapitel 5

Fazit und zukünftige Arbeit

In diesem Bericht haben wir Ergebnisse eines Projektes mit regionalen Unternehmen vorgestellt. Dazu haben wir zunächst den Hintergrund beschrieben und die Forschungsmethodik skizziert. Danach haben wir einen Ordnungsrahmen zur Kategorisierung und vier Handlungsempfehlungen zur Erhöhung der Softwarequalität vorgestellt.

Projektbezug und geeignete Anreizsysteme steigern nicht nur die Produktivität, sondern dienen auch der Qualitätssicherung. Damit Komplexität dieser nicht entgegenwirkt, bedarf es systematischer Gegenmaßnahmen. Um Testwerkzeuge effektiv und effizient nutzen zu können, sollten diese an die eigenen Prozesse anpassbar sein, anstatt die eigenen Prozesse zu „verbiegen“. Schließlich empfehlen wir, dem Testen ein Stufenkonzept zugrunde zu legen und es in ein Gesamtkonzept einzubetten. Nur so ist eine *Kultur des Testens* erreichbar.

Das IAI-Projekt ist noch nicht beendet. Für die Zukunft ist geplant, die Ergebnisse mit den Teilnehmern zu diskutieren und Erfahrungen in der Implementierung der Empfehlungen zu sammeln. Gut anzuschließen wäre eine quantitative Studie zur Ergebnisvalidierung.

Literaturverzeichnis

[1] Peter Naur and Brian Randell. *Software Engineering: Report of a conf. spon. by the NATO Science Committee, Garmisch, Germany*. Scientific Affairs Division, NATO, 1969.

[2] E. Dijkstra. The humble programmer. *Communications of the ACM*, 15:859–866, 1972.

[3] G. Le Lann. An analysis of the ariane 5 flight 501 failure-a system engineering perspective. *Proc. Conf. Engineering of Computer-Based Systems*, pages 339–346, 1997.

[4] NASA. Mars climate orbiter mishap investigation board phase I report, 1999.

[5] Stephen Flowers. *Software failure: management failure: amazing stories and cautionary tales*. John Wiley & Sons, Inc., New York, 1996.

[6] Donald Gotterbarn and Keith W. Miller. The public is the priority: Making decisions using the software engineering code of ethics. *Computer*, 42(6):58–65, 2009.

[7] Capers Jones. *Software Quality: Analysis and Guidelines for Success*. Thomson Learning, 1997.

[8] Pierre Audoin Consultants (PAC). Pressemitteilung: Software-Testen ist bei vielen Unternehmen schlecht organisiert, April 2008.

[9] Jens Coldewey. Schlechte Noten für Informatik-Ausbildung. *OBJEKTspektrum*, 5:12–15, 2009.

[10] J. B. Rainsberger. Personal planning. *IEEE Softw.*, 24(1):16–17, 2007.

[11] Tim A. Majchrzak. Best practices for the organizational implementation of software testing. In *Proc. of the 43th Annual Hawaii Int. Conf. on System Sciences (HICSS-43)*. IEEE Computer Society, 2010.

[12] Tim A. Majchrzak. Technische Aspekte des erfolgreichen Testens von Software in Unternehmen. In Jens Knoop and Adrian Prantl, editors, *Schriftenreihe des Instituts für Computersprachen, Bericht 2009-X-1: 15. Kolloquium Programmiersprachen und Grundlagen der Programmierung*, pages 193–207, Maria Taferl, Austria, 2009. TU Wien.

[13] Vijay K. Vaishnavi and William Kuechler, Jr. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach Publications, Boston, 2007.

[14] Juhani Iivari. A paradigmatic analysis of information systems as a design science. *Scandinavian J. Inf. Systems*, 19(2), 2007.

[15] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1), 2004.

[16] John Watkins. *Testing IT: an off-the-shelf software testing process*. Cambridge University Press, New York, 2001.

[17] Rex Black. *Managing the Testing Process*. Wiley, Indianapolis, 3rd edition, 2009.

[18] Andreas Spillner, Thomas Roßner, Mario Winter, and Tilo Linz. *Praxiswissen Softwaretest – Testmanagement*. Dpunkt, Heidelberg, 2008.

[19] Ernest Wallmüller. *Software-Qualitätsmanagement in der Praxis*. Hanser, München, 2 edition, 2001.

[20] Kent Beck. *Test-Driven Development by Example*. Addison-Wesley, 2002.

[21] Dirk W. Hoffmann. *Software-Qualität*. Springer, Berlin, 2008.

[22] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum-Akademischer Verlag, Berlin, 2 edition, 2009.

[23] Rex Black. *Pragmatic Software Testing*. Wiley, Indianapolis, 2007.

[24] Peter Braun and Sabine Canditt. Wenn der Kunde nicht weiß, was er will: Tipps für den agilen Umgang mit Anforderungen. *OBJEKTspektrum*, 5:64–72, 2009.

[25] Christof Ebert. *Systematisches Requirements Engineering Management*. Dpunkt, Heidelberg, 2 edition, 2008.

[26] Klaus Pohl. *Requirements Engineering*. Dpunkt, Heidelberg, 2 edition, 2008.

[27] Gereon Tochtrop. Testexperte als anforderungsmanager – ein erfahrungsbericht. *Softwaretechnik-Trends*, 28(3):34–37, 2008.

[28] Fabian Ahrendts and Anita Marton. *IT-Risikomanagement leben!* Springer, Berlin, 2007.

[29] Martyn A. Ould. *Managing Software Quality and Business Risk*. Wiley, New York, 1999.

[30] Georg Erwin Thaller. *Drachentöter: Risikomanagement für Software-Projekte*. Heise, Hannover, 2003.

[31] Martin Wieczorek, Bob Bartlett, and Uwe Naujoks, editors. *Business Continuity: IT Risk Management for International Corporations*. Springer, Berlin, 2002.

[32] Harry M. Sneed, Manfred Baumgartner, and Richard Seidl. *Der Systemtest*. Hanser, München, 2 edition, 2009.

[33] Pierre Henry. *The Testing Network: An Integral Approach to Test Activities in Large Software Projects*. Springer, Heidelberg, 2008.

Anhang

A.1 Leitfaden für die Interviews

IAI Projekt „Testen“ Interview-Leitfaden
für semi-strukturierte Experteninterviews

Die Fragen sind offen; eine Diskussion ist erwünscht.

1 Begrüßung

1.1 Erläuterung des Vorgehens

1.2 Klärung der beteiligten Personen

1.3 Wünsche der Interviewpartner

2 Status-Quo des Testens im Unternehmen

2.1 Art der Softwareentwicklung im Unternehmen

2.2 Organisatorische Einbettung des Testens(**wer**?)

- Wie ist das Testen in die Unternehmenskultur eingebettet?
- Wer ist für Tests verantwortlich?
- Gibt es eine Abteilung bzw. Organisationseinheit für das Testen?
- Werden Tests von Entwicklern vorgenommen oder gibt es spezialisierte Tester?

2.3 Ablauf des Testens, Zeitpunkt, Abfolge, Integration in den Softwareentwicklungsprozess (**wann**?)

- Existiert ein definierter Testprozess?
- Wann wird getestet?
- Welche Schritte gibt es dabei?
- Wie ist das Testen in den Softwareentwicklungsprozess integriert?

2.4 Was wird getestet? Algorithmen, Schnittstellen, Oberfläche, etc. (**was**?)

- Systemschnittstellen, Algorithmen, grafische Benutzerschnittstellen etc.

2.5 Art/Technik der Tests (**welche**? **wie**?)

2.5.1 Manuell und/oder automatisiert?

- falls automatisiert: aufgenommen oder „echte“ Automation?
- Black Box und/oder Glass Box?

2.5.2 Funktionale Tests

- Komponenten-, Integrations- und Systemtests

2.5.3 Regressionstests

2.5.4 Stresstests / Performancemessungen

2.5.5 Tests der Datenhaltung /-bank

2.5.6 Betatests, Abnahmetests, Akzeptanztests

2.6 Eingesetzte Software (**womit**?)

2.6.1 Unterscheidung: Software für das eigentliche Testen und Software zur Testunterstützung (Testfallverwaltung etc.)

2.6.2 IBM Rational

2.6.3 HP Mercury

2.6.4 Kleine Tools z.B. Quickcheck, AgitarOne

2.6.5 Eigenentwicklungen

2.7 Genaues Vorgehen, erfolgreiche Strategien etc. (**wie**?)

- Die Diskussion dieser Frage kann ausgedehnt werden

3 Besonderheiten, die im Unternehmen angetroffen wurden (vertiefte Diskussion)

3.1 Probleme

- Häufige, prinzipielle oder offene Probleme
- Verzicht auf bestimmte Testarten

3.2 Besonders erfolgreiche Strategien („Best practises“)

3.3 Besondere Erfahrungen mit gekaufter Software bzw. mit Eigenentwicklungen

4 Planungen für die Zukunft (im Bezug auf das Testen)

5 Abschluss

- Erläuterung des weiteren Vorgehens
- Ankündigung der Broschüre

A.2 Leitfaden für die Interviews (Englische Version)

IAI Project “Testing” Interview Guideline
for semi-structured expert-interviews

Questions are open. Discussion is encouraged.

1 Salutation

1.1 Explanation of the interview procedure

1.2 Introduction of participating company staff and researchers

1.3 Discussion of expectations of company staff

2 Status quo of testing in the enterprise

2.1 Kind of software developed

2.2 Organizational background of testing (**who**?)

- How is testing embedded into the company culture?
- Who is responsible for testing?
- Is there a department for testing?
- Is testing done by developers or are there distinguished testers?

2.3 Testing procedure (**when**?)

- Is there a process for testing?
- When is testing done?
- What steps are taken?
- How is testing integrated into the software development process?

2.4 What is being tested? (**what**?)

- System interfaces, algorithms, graphical user interface etc.

2.5 Kinds of tests and methodology used (**which**? **how**?)

2.5.1 Manual or automatized?

- if automatized: recorded or "truly" automated?
- Black box and/or glass box?

2.5.2 Functional tests

- Component, integration and system tests

2.5.3 Regression tests

2.5.4 Stress tests and performance measurement

2.5.5 Tests of the data abstraction layer and the database

2.5.6 Beta tests and acceptance tests

2.6 Software support (**whereby**?)

2.6.1 Distinction: Testing software and supportive software (e.g. test case management)

2.6.2 IBM Rational

2.6.3 HP Mercury

2.6.4 Tools such as QuickCheck or AgitarOne

2.6.5 Individually developed tools

2.7 Detailed process and successful procedures (**how**?)

- Discussion of this question can be expanded

3 Particularities met in the company (in-depth discussion)

3.1 Problems

- Common, general and open problems
- Are some testing techniques not used?

3.2 Successful strategies (best practices)

3.3 Notable experiences with commercial or individually developed testing software

4 Plans for the future (regarding testing)

5 Closing

- Explanation of further procedure
- Announcement of the booklet

Arbeitsberichte des Instituts für Wirtschaftsinformatik

Nr. 1 Bolte, Ch.; Kurbel, K.; Moazzami, M.; Pietsch, W.: Erfahrungen bei der Entwicklung eines Informationssystems auf RDBMS- und 4GL-Basis. Februar 1991.

Nr. 2 Kurbel, K.: Das technologische Umfeld der Informationsverarbeitung - Ein subjektiver 'State of the Art'-Report über Hardware, Software und Paradigmen. März 1991.

Nr. 3 Kurbel, K.: CA-Techniken und CIM. Mai 1991.

Nr. 4 Nietsch, M.; Nietsch, T.; Rautenstrauch, C.; Rinschede, M.; Siedentopf, J.: Anforderungen mittelständischer Industriebetriebe an einen elektronischen Leitstand - Ergebnisse einer Untersuchung bei zwölf Unternehmen. Juli 1991.

Nr. 5 Becker, J.; Prischmann, M.: Konnektionistische Modelle - Grundlagen und Konzepte. September 1991.

Nr. 6 Grob, H. L.: Ein produktivitätsorientierter Ansatz zur Evaluierung von Beratungserfolgen. September 1991.

Nr. 7 Becker, J.: CIM und Logistik. Oktober 1991.

Nr. 8 Burgholz, M.; Kurbel, K.; Nietsch, Th.; Rautenstrauch, C.: Erfahrungen bei der Entwicklung und Portierung eines elektronischen Leitstands. Januar 1992.

Nr. 9 Becker, J.; Prischmann, M.: Anwendung konnektionistischer Systeme. Februar 1992.

Nr. 10 Becker, J.: Computer Integrated Manufacturing aus Sicht der Betriebswirtschaftslehre und der Wirtschaftsinformatik. April 1992.

Nr. 11 Kurbel, K.; Dornhoff, P.: A System for Case-Based Effort Estimation for Software-Development Projects. Juli 1992.

Nr. 12 Dornhoff, P.: Aufwandsplanung zur Unterstützung des Managements von Softwareentwicklungsprojekten. August 1992.

Nr. 13 Eicker, S.; Schnieder, T.: Reengineering. August 1992.

Nr. 14 Erkelenz, F.: KVD2 - Ein integriertes wissensbasiertes Modul zur Bemessung von Krankenhausverweildauern - Problemstellung, Konzeption und Realisierung. Dezember 1992.

Nr. 15 Horster, B.; Schneider, B.; Siedentopf, J.: Kriterien zur Auswahl konnektionistischer Verfahren für betriebliche Probleme. März 1993.

Nr. 16 Jung, R.: Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht. Juli 1993.

Nr. 17 Miller, C.; Weiland, R.: Der Übergang von proprietären zu offenen Systemen aus Sicht der Transaktionskostentheorie. Juli 1993.

Nr. 18 Becker, J.; Rosemann, M.: Design for Logistics - Ein Beispiel für die logistikgerechte Gestaltung des Computer Integrated Manufacturing. Juli 1993.

Nr. 19 Becker, J.; Rosemann, M.: Informationswirtschaftliche Integrationsschwerpunkte innerhalb der logistischen Subsysteme - Ein Beitrag zu einem produktionsübergreifenden Verständnis von CIM. Juli 1993.

Nr. 20 Becker, J.: Neue Verfahren der entwurfs- und konstruktionsbegleitenden Kalkulation und ihre Grenzen in der praktischen Anwendung. Juli 1993.

Nr. 21 Becker, K.; Prischmann, M.: VESKONN - Prototypische Umsetzung eines modularen Konzepts zur Konstruktionsunterstützung mit konnektionistischen Methoden. November 1993.

Nr. 22 Schneider, B.: Neuronale Netze für betriebliche Anwendungen: Anwendungspotentiale und existierende Systeme. November 1993.

Nr. 23 Nietsch, T.; Rautenstrauch, C.; Rehfeldt, M.; Rosemann, M.; Turowski, K.: Ansätze für die Verbesserung von PPS-Systemen durch Fuzzy-Logik. Dezember 1993.

Nr. 24 Nietsch, M.; Rinschede, M.; Rautenstrauch, C.: Werkzeuggestützte Individualisierung des objektorientierten Leitstands ooL. Dezember 1993.

Nr. 25 Meckenstock, A.; Unland, R.; Zimmer, D.: Flexible Unterstützung kooperativer Entwurfsumgebungen durch einen Transaktions-Baukasten. Dezember 1993.

Nr. 26 Grob, H. L.: Computer Assisted Learning (CAL) durch Berechnungsexperimente. Januar 1994.

Nr. 27 Kirn, St.; Unland, R. (Hrsg.): Tagungsband zum Workshop “Unterstützung Organisatorischer Prozesse durch CSCW”. In Kooperation mit GI-Fachausschuß 5.5 “Betriebliche Kommunikations- und Informationssysteme” und Arbeitskreis 5.5.1 “Computer Supported Cooperative Work”, Westfälische Wilhelms-Universität Münster, 4.-5. November 1993. November 1993.

Nr. 28 Kirn, St.; Unland, R.: Zur Verbundintelligenz integrierter Mensch-Computer-Teams: Ein organisationstheoretischer Ansatz. März 1994.

Nr. 29 Kirn, St.; Unland, R.: Workflow Management mit kooperativen Softwaresystemen: State of the Art und Problemabriß. März 1994.

Nr. 30 Unland, R.: Optimistic Concurrency Control Revisited. März 1994.

Nr. 31 Unland, R.: Semantics-Based Locking: From Isolation to Cooperation. März 1994.

Nr. 32 Meckenstock, A.; Unland, R.; Zimmer, D.: Controlling Cooperation and Recovery in Nested Transactions. März 1994.

Nr. 33 Kurbel, K.; Schnieder, T.: Integration Issues of Information Engineering Based I-CASE Tools. September 1994.

Nr. 34 Unland, R.: TOPAZ: A Tool Kit for the Construction of Application Specific Transaction. November 1994.

Nr. 35 Unland, R.: Organizational Intelligence and Negotiation Based DAI Systems - Theoretical Foundations and Experimental Results. November 1994.

Nr. 36 Unland, R.; Kirn, St.; Wanka, U.; O’Hare, G. M. P.; Abbas, S.: AEGIS: AGENT ORIENTED ORGANISATIONS. Februar 1995.

Nr. 37 Jung, R.; Rimpler, A.; Schnieder, T.; Teubner, A.: Eine empirische Untersuchung von Kosteneinflußfaktoren bei integrationsorientierten Reengineering-Projekten. März 1995.

Nr. 38 Kirn, St.: Organisatorische Flexibilität durch Workflow-Management-Systeme?. Juli 1995.

Nr. 39 Kirn, St.: Cooperative Knowledge Processing: The Key Technology for Future Organizations. Juli 1995.

Nr. 40 Kirn, St.: Organisational Intelligence and Distributed AI. Juli 1995.

Nr. 41 Fischer, K.; Kirn, St.; Weinhard, Ch. (Hrsg.): Organisationsaspekte in Multiagentensytemen. September 1995.

Nr. 42 Grob, H. L.; Lange, W.: Zum Wandel des Berufsbildes bei Wirtschaftsinformatikern, Eine empirische Analyse auf der Basis von Stellenanzeigen. Oktober 1995.

Nr. 43 Abu-Alwan, I.; Schlagheck, B.; Unland, R.: Evaluierung des objektorientierten Datebankmanagementsystems ObjectStore. Dezember 1995.

Nr. 44 Winter, R.: Using Formalized Invariant Properties of an Extended Conceptual Model to Generate Reusable Consistency Control for Information Systems. Dezember 1995.

Nr. 45 Winter, R.: Design and Implementation of Derivation Rules in Information Systems. Februar 1996.

Nr. 46 Becker, J.: Eine Architektur für Handelsinformationssysteme. März 1996.

Nr. 47 Becker, J.; Rosemann, M. (Hrsg.): Workflowmanagement - State-of-the-Art aus Sicht von Theorie und Praxis, Proceedings zum Workshop vom 10. April 1996. April 1996.

Nr. 48 Rosemann, M.; zur Mühlen, M.: Der Lösungsbeitrag von Metadatenmodellen beim Vergleich von Workflowmanagementsystemen. Juni 1996.

Nr. 49 Rosemann, M.; Denecke, Th.; Püttmann, M.: Konzeption und prototypische Realisierung eines Informationssystems für das Prozeßmonitoring und –controlling. September 1996.

Nr. 50 v. Uthmann, C.; Turowski, K. unter Mitarbeit von Rehfeldt, M.; Skall, M.: Workflowbasierte Geschäftsprozeßregelung als Konzept für das Management von Produktentwicklungsprozessen. November 1996.

Nr. 51 Eicker, S.; Jung, R.; Nietsch, M.; Winter, R.: Entwicklung eines Data Warehouse für das Produktionscontrolling: Konzepte und Erfahrungen. November 1996.

Nr. 52 Becker, J.; Rosemann, M.; Schütte, R. (Hrsg.): Entwicklungsstand und Entwicklungsperspektiven der Referenzmodellierung, Proceedings zur Veranstaltung vom 10. März 1997. März 1997.

Nr. 53 Loos, P.: Capture More Data Semantic Through The Expanded Entity-Relationship Model (PERM). Februar 1997.

Nr. 54 Becker, J.; Rosemann, M. (Hrsg.): Organisatorische und technische Aspekte beim Einsatz von Workflowmanagementsystemen. Proceedings zur Veranstaltung vom 10. April 1997. April 1997.

Nr. 55 Holten, R.; Knackstedt, R.: Führungsinformationssysteme - Historische Entwicklung und Konzeption. April 1997.

Nr. 56 Holten, R.: Die drei Dimensionen des Inhaltsaspektes von Führungsinformationssystemen. April 1997.

Nr. 57 Holten, R.; Striemer, R.; Weske, M.: Ansätze zur Entwicklung von Workflow-basierten Anwendungssystemen - Eine vergleichende Darstellung. April 1997.

Nr. 58 Kuchen, H.: Arbeitstagung Programmiersprachen, Tagungsband. Juli 1997.

Nr. 59 Vering, O.: Berücksichtigung von Unschärfe in betrieblichen Informationssystemen – Einsatzfelder und Nutzenpotentiale am Beispiel der PPS. September 1997.

Nr. 60 Schwegmann, A.; Schlagheck, B.: Integration der Prozeßorientierung in das objektorientierte Paradigma: Klassenzuordnungsansatz vs. Prozeßklassenansatz. Dezember 1997.

Nr. 61 Speck, M.: In Vorbereitung.

Nr. 62 Wiese, J.: Ein Entscheidungsmodell für die Auswahl von Standardanwendungssoftware am Beispiel von Warenwirtschaftssystemen. März 1998.

Nr. 63 Kuchen, H.: Workshop on Functional and Logic Programming, Proceedings. Juni 1998.

Nr. 64 v. Uthmann, C.; Becker, J.; Brödner, P.; Maucher, I.; Rosemann, M.: PPS meets Workflow. Proceedings zum Workshop vom 9. Juni 1998. Juni 1998.

Nr. 65 Scheer, A.-W.; Rosemann, M.; Schütte, R. (Hrsg.): Integrationsmanagement. Januar 1999.

Nr. 66 zur Mühlen, M.; Ehlers, L.: Internet - Technologie und Historie. Juni 1999.

Nr. 67 Holten R.: A Framework for Information Warehouse Development Processes. Mai 1999.

Nr. 68 Holten R.; Knackstedt, R.: Fachkonzeption von Führungsinformationssystemen – Instanziierung eines FIS-Metamodells am Beispiel eines Einzelhandelsunternehmens. Mai 1999.

Nr. 69 Holten, R.: Semantische Spezifikation Dispositiver Informationssysteme. Juli 1999.

Nr. 70 zur Mühlen, M.: In Vorbereitung.

Nr. 71 Klein, S.; Schneider, B.; Vossen, G.; Weske, M.; Projektgruppe PESS: Eine XMLbasierte Systemarchitektur zur Realisierung flexibler Web-Applikationen. Juli 2000.

Nr. 72 Klein, S.; Schneider, B. (Hrsg): Negotiations and Interactions in Electronic Markets, Proceedings of the Sixth Research Symposium on Emerging Electronic Markets, Muenster, Germany, September 19 - 21, 1999. August 2000.

Nr. 73 Becker, J.; Bergerfurth, J.; Hansmann, H.; Neumann, S.; Serries, T.: Methoden zur Einführung Workflow-gestützter Architekturen von PPS-Systemen. November 2000.

Nr. 74 Terveer, I.: Die asymptotische Verteilung der Spannweite bei Zufallsgrößen mit paarweise identischer Korrelation. Februar 2002.

Nr. 75 Becker, J. (Ed.): Research Reports, Proceedings of the University Alliance Executive Directors Workshop – ECIS 2001. Juni 2001.

Nr. 76 Klein, St.; u. a. (Eds.): MOVE: Eine flexible Architektur zur Unterstützung des Außendienstes mit mobile devices.

Nr. 77 Knackstedt, R.; Holten, R.; Hansmann, H.; Neumann, St.: Konstruktion von Methodiken: Vorschläge für eine begriffliche Grundlegung und domänenspezifische Anwendungsbeispiele. Juli 2001.

Nr. 78 Holten, R.: Konstruktion domänenspezifischer Modellierungstechniken für die Modellierung von Fachkonzepten. August 2001.

Nr. 79 Vossen, G.; Hüsemann, B.; Lechtenbörger, J.: XLX – Eine Lernplattform für den universitären Übungsbetrieb. August 2001.

Nr. 80 Knackstedt, R.; Serries, Th.: Gestaltung von Führungsinformationssystemen mittels Informationsportalen; Ansätze zur Integration von Data-Warehouse- und Content-Management-Systemen. November 2001.

Nr. 81 Holten, R.: Conceptual Models as Basis for the Integrated Information Warehouse Development. Oktober 2001.

Nr. 82 Teubner, A.: Informationsmanagement: Historie, disziplinärer Kontext und Stand der Wissenschaft. Februar 2002.

Nr. 83 Vossen, G.: Vernetzte Hausinformationssysteme – Stand und Perspektive. Oktober 2001.

Nr. 84 Holten, R.: The MetaMIS Approach for the Specification of Management Views on Business Processes. November 2001.

Nr. 85 Becker, J.; Neumann, S.; Hansmann, H.: (Titel in Vorbereitung). Januar 2002.

Nr. 86 Teubner, R. A.; Klein, S.: Bestandsaufnahme aktueller deutschsprachiger Lehrbücher zum Informationsmanagement. März 2002.

Nr. 87 Holten, R.: Specification of Management Views in Information Warehouse Projects. April 2002.

Nr. 88 Holten, R.; Dreiling, A.: Specification of Fact Calculations within the MetaMIS Approach. Juni 2002.

Nr. 89 Holten, R.: Metainformationssysteme – Backbone der Anwendungssystemkopplung. Juli 2002.

Nr. 90 Becker, J.; Knackstedt, R. (Hrsg.): Referenzmodellierung 2002. Methoden – Modelle – Erfahrungen. August 2002.

Nr. 91 Teubner, R. A.: Grundlegung Informationsmanagement. Februar 2003.

Nr. 92 Vossen, G.; Westerkamp, P.: E-Learning as a Web Service. Februar 2003.

Nr. 93 Becker, J.; Holten, R.; Knackstedt, R.; Niehaves, B.: Forschungsmethodische Positionierung in der Wirtschaftsinformatik - epistemologische, ontologische und linguistische Leitfragen. Mai 2003.

Nr. 94 Algermissen, L.; Niehaves, B.: E-Government – State of the art and development perspectives. April 2003.

Nr. 95 Teubner, R. A.; Hübsch, T.: Is Information Management a Global Discipline? Assessing Anglo-American Teaching and Literature through Web Content Analysis. November 2003.

Nr. 96 Teubner, R. A.: Information Resource Management. Dezember 2003.

Nr. 97 Köhne, F.; Klein, S.: Prosuming in der Telekommunikationsbranche: Konzeptionelle Grundlagen und Ergebnisse einer Delphi-Studie. Dezember 2003.

Nr. 98 Vossen, G.; Pankratius, V.: Towards E-Learning Grids. 2003.

Nr. 99 Vossen, G.; Paul, H.: Tagungsband EMISA 2003: Auf dem Weg in die E-Gesellschaft. 2003.

Nr. 100 Vossen, G.; Vidyasankar, K.: A Multi-Level Model for Web Service Composition. 2003.

Nr. 101 Becker, J.; Serries, T.; Dreiling, A.; Ribbert, M.: Datenschutz als Rahmen für das Customer Relationship Management – Einfluss des geltenden Rechts auf die Spezifikation von Führungsinformationssystemen. November 2003.

Nr. 102 Müller, R.A.; Lembeck, C.; Kuchen, H.: GlassTT – A Symbolic Java Virtual Machine using Constraint Solving Techniques for Glass-Box Test Case Generation. November 2003.

Nr. 103 Becker, J; Brelage C.; Crisandt J.; Dreiling A.; Holten R.; Ribbert M.; Seidel S.: Methodische und technische Integration von Daten- und Prozessmodellierungstechniken für Zwecke der Informationsbedarfsanalyse. März 2004.

Nr. 104 Teubner, R. A.: Information Technology Management. April 2004.

Nr. 105 Teubner, R. A.: Information Systems Management. August 2004.

Nr. 106 Becker, J.; Brelage, C.; Gebhardt, Hj.; Recker, J.; Müller-Wienbergen, F.: Fachkonzeptionelle Modellierung und Analyse web-basierter Informationssysteme mit der MWKiD Modellierungstechnik am Beispiel von ASInfo. Mai 2004.

Nr. 107 Hagemann, S.; Rodewald, G.; Vossen, G.; Westerkamp, P.; Albers, F.; Voigt, H.: BoGSy – ein Informationssystem für Botanische Gärten. September 2004.

Nr. 108 Schneider, B.; Totz, C.: Web-gestützte Konfiguration komplexer Produkte und Dienstleistungen. September 2004.

Nr. 109 Algermissen, L; Büchel, N.; Delfmann, P.; Dümmer, S.; Drawe, S.; Falk, T.; Hinzen, M.; Meesters, S.; Müller, T.; Niehaves, B.; Niemeyer, G.; Pepping, M.; Robert, S.; Rosenkranz, C.; Stichnote, M.; Wienefoet, T.: Anforderungen an Virtuelle Rathäuser – Ein Leitfaden für die herstellerunabhängige Softwareauswahl. Oktober 2004.

Nr. 110 Algermissen, L; Büchel, N.; Delfmann, P.; Dümmer, S.; Drawe, S.; Falk, T.; Hinzen, M.; Meesters, S.; Müller, T.; Niehaves, B.; Niemeyer, G.; Pepping, M.; Robert, S.; Rosenkranz, C.; Stichnote, M.; Wienefoet, T.: Fachkonzeptionelle Spezifikation von Virtuellen Rathäusern – Ein Konzept zur Unterstützung der Implementierung. Oktober 2004.

Nr. 111 Becker, J.; Janiesch, C.; Pfeiffer, D.; Rieke, T.; Winkelmann, A.: Studie: Verteilte Publikationserstellung mit Microsoft Word und den Microsoft SharePoint Services. Dezember 2004.

Nr. 112 Teubner, R. A.; Terwey, J.: Informations-Risiko-Management: Der Beitrag internationaler Normen und Standards. April 2005.

Nr. 113 Teubner, R. A.: Methodische Integration von Organisations- und Informationssystemgestaltung: Historie, Stand und zukünftige Herausforderungen an die Wirtschaftsinformatik-Forschung. Mai 2006.

Nr. 114 Becker, J.; Janiesch, C.; Knackstedt, R.; Kramer, S.; Seidel, S.: Konfigurative Referenzmodellierung mit dem H2-Toolset. November 2006.

Nr. 115 Becker, J.; Fleischer, S.; Janiesch, C.; Knackstedt, R; Müller-Wienbergen, F.; Seidel, S.: H2 for Reporting – Analyse, Konzeption und kontinuierliches Metadatenmanagement von Management-Informationssystemen. Februar 2007.

Nr. 116 Becker, J.; Kramer, S.; Janiesch, C.: Modellierung und Konfiguration elektronischer Geschäftsdokumente mit dem H2-Toolset. November 2007.

Nr. 117 Becker, J., Winkelmann, A., Philipp, M.: Entwicklung eines Referenzvorgehensmodells zur Auswahl und Einführung von Office Suiten. Dezember 2007.

Nr. 118 Teubner, A.: IT-Service Management in Wissenschaft und Praxis.

Nr. 119 Becker, J.; Knackstedt, R.; Beverungen, D. et al.: Ein Plädoyer für die Entwicklung eines multidimensionalen Ordnungsrahmens zur hybriden Wertschöpfung. Januar 2008.

Nr. 120 Becker, J.; Krcmar, H.; Niehaves, B. (Hrsg.): Wissenschaftstheorie und gestaltungsorientierte Wirtschaftsinformatik. Februar 2008.

Nr. 121 Becker, J.; Richter, O.; Winkelmann, A.: Analyse von Plattformen und Marktübersichten für die Auswahl von ERP- und Warenwirtschaftssysteme. Februar 2008.

Nr. 122 Vossen, G.: DaaS-Workshop und das Studi-Programm. Februar 2009.

Nr. 123 Knackstedt, R.; Pöppelbuß, J.: Dokumentationsqualität von Reifegradmodellentwicklungen. April 2009.

Nr. 124 Winkelmann, A.; Kässens, S.: Fachkonzeptionelle Spezifikation einer Betriebsdatenerfassungskomponente für ERP-Systeme. Juli 2009.

Nr. 125 Becker, J.; Knackstedt, R.; Beverungen, D.; Bräuer, S.; Bruning, D.; Christoph, D.; Greving, S.; Jorch, D.; Joßbächer, F.; Jostmeier, H.; Wiethoff, S.; Yeboah, A.: Modellierung der hybriden Wertschöpfung: Eine Vergleichsstudie zu Modellierungstechniken. November 2009.

Nr. 126 Becker, J.; Beverungen, D.; Knackstedt, R.; Behrens, H.; Glauner, C.; Wakke, P.: Stand der Normung und Standardisierung der hybriden Wertschöpfung. Januar 2010.

Arbeitsberichte des Instituts für Wirtschaftsinformatik

Kontakt
Institut für Wirtschaftsinformatik
Praktische Informatik in der Wirtschaft
✉ Leonardo-Campus 3, 48149 Münster
✆ +49 (251) 8338264
@ tima@wi.uni-muenster.de
🖳 http://www.wi.uni-muenster.de/pi/

WIRTSCHAFTS INFORMATIK

Westfälische Wilhelms-Universität Münster

ISSN 1438-3985