



ulm university universität  
**ulm**

Universität Ulm

Fakultät für Ingenieurwissenschaften, Informatik und  
Psychologie

Institut für Datenbanken und Informationssysteme

# Kombinatorische Testmethoden zur Analyse aktuarieller Software

Bachelorarbeit

in Informatik

vorgelegt von  
Johannes Gabriel Sindlinger  
am 23.10.2021

**Gutachter**

Prof. Dr. Manfred Reichert



# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>iii</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Ziel der Arbeit . . . . .	1
1.3. Verwandte Arbeiten . . . . .	2
1.4. Struktur der Arbeit . . . . .	3
<b>2. Theoretische Grundlagen</b>	<b>4</b>
2.1. Einführung ins Softwaretesten . . . . .	4
2.1.1. Testfall-Design . . . . .	7
2.1.2. Beispiele für Teststrategien . . . . .	14
2.2. Combinatorial Testing . . . . .	17
2.2.1. Einführung in Combinatorial Testing . . . . .	17
2.2.2. Maße für Testabdeckung . . . . .	23
2.2.3. Algorithmen zur Testfallerzeugung . . . . .	27
2.2.4. Tools zur Testfallerzeugung . . . . .	37
<b>3. Anwendungsfall</b>	<b>42</b>
3.1. Vorstellung des Testobjekts . . . . .	42
3.1.1. Versicherungstechnische Grundlagen . . . . .	43
3.1.2. Easy Web . . . . .	46
3.2. Implementierung . . . . .	53
3.2.1. Basis-System . . . . .	55
3.2.2. Vergleich Combinatorial Testing / Random Testing . . . . .	59
3.2.3. Erweitertes System . . . . .	60
3.2.4. $t$ -fache Kombinatorik für jeden Durchführungsweg . . . . .	62

## *Inhaltsverzeichnis*

3.2.5. Vergleich verschiedener Combinatorial Testing Algorithmen . . . . .	64
3.3. Ergebnisse . . . . .	65
3.3.1. Basis-System . . . . .	65
3.3.2. Vergleich Combinatorial Testing / Random Testing . . . . .	66
3.3.3. Erweitertes System . . . . .	67
3.3.4. $t$ -fache Kombinatorik für jeden Durchführungsweg . . . . .	70
3.3.5. Vergleich verschiedener Combinatorial Testing Algorithmen . . . . .	71
<b>4. Diskussion</b>	<b>76</b>
<b>5. Fazit</b>	<b>83</b>
<b>A. Anhang</b>	<b>85</b>
A.1. Bedingungen Basis-System . . . . .	85
A.2. Bedingungen Erweitertes System . . . . .	86
A.3. Mögliche Werte Erweitertes System . . . . .	89
<b>Literaturverzeichnis</b>	<b>90</b>

# Abbildungsverzeichnis

2.1.	Die verschiedenen Komponenten der Qualitätssicherung in der Software-Entwicklung in der Übersicht [LL10, S. 271]. . . . .	5
2.2.	Ein Denkfehler bei der Implementierung führt zu einem Codefehler, welcher wiederum ein Fehlverhalten der Software auslöst. . . . .	6
2.3.	Das V-Modell dient als Grundlage für das stufenbasierte Testen. Verschiedene Phasen im Entwicklungsprozess einer Software werden mit unmittelbaren Testfällen verknüpft [CJ02, S. 101]. . . . .	10
2.4.	Fehlerverhalten verschiedener Software-Systeme nach Untersuchungen von Kuhn et al. [KWG04]: Die Interaktion von sechs oder weniger Parameter ist demnach für 100% der Fehler der vier untersuchten Software-Systeme verantwortlich. . . . .	18
2.5.	Beispiel für die Interaktionsproblematik beim Testen verschiedener Parameter: Bei den Kombinationen 'Height' < 10 / 'Width' < 10 und 'Length' ≥ 10 / 'Width' ≥ 10 / 'Height' ≥ 10 kommt es zu einem Fehlverhalten [KKL <sup>+</sup> 10]. . . . .	19
2.6.	Übersicht über die verschiedenen Tools und Algorithmen zur Erstellung von Testmengen nach dem Prinzip des Combinatorial Testing.: Die farbliche Markierung repräsentiert die Klassifizierung der Algorithmen in Bezug auf das Vorgehen bei der Erstellung der Testmengen. Kursiv gedruckte Algorithmen werden im Rahmen dieser Arbeit nicht berücksichtigt. . . . .	38
2.7.	Screenshot der grafischen Benutzeroberfläche von ACTS nach Erstellung eines Covering Arrays. Screenshot erstellt am 15.12.2021. . . . .	39
3.1.	Die Startseite von Easy Web [ALHb]: Es können Kundenprofile zur Kalkulation verschiedener Altersvorsorgeprodukte angelegt und gespeichert werden. Zudem besteht die Möglichkeit ein Kund*innenprofil im Rahmen der IDD-Versicherungsvertriebsrichtlinie anzulegen. Screenshot erstellt am 15.12.2021.	47

## *Abbildungsverzeichnis*

3.2. Das Auswahlmenü von Easy Web bei einer Kund*innenberatung: Es wird zwischen privater und betrieblicher Altersvorsorge unterschieden, zudem können Tarife verschiedener Durchführungswege ausgewählt werden. Screenshot erstellt am 15.12.2021. . . . . .	48
3.3. Die Eingabemaske für den Tarif 'Klassische Rente mit Rentengarantie (AR10)' des Durchführungswegs Direktversicherung. Im oberen Bereich werden Angaben zur versicherten Person verlangt, darunter folgt die Spezifizierung der Parameter für den Tarif. Screenshot erstellt am 15.12.2021. . . . .	52
3.4. Fehlermeldung von Easy Web bei einer Eingabe des Beitrags von 0 Euro im Tarif AR10 des Durchführungswegs Direktversicherung. Screenshot erstellt am 15.12.2021. . . . .	53
3.5. Gegenüberstellung des Basis-Systems und des Erweiterten Systems in Bezug auf die Anzahl der Testfälle (linke Seite) und die Ausführungszeit (rechte Seite) bei der Erstellung von Testmengen via ACTS und allgemeinem IPOG-Algorithmus . . . . .	69
3.6. Grafische Visualisierung der Anzahl erzeugter Testfälle der verschiedenen Algorithmen . . . . .	74

# Tabellenverzeichnis

2.1.	Ein Covering Array für die Kombination aller Dreierkombinationen bei der Wahl von 10 binären EingabevARIABLEN A-J. Die verschiedenen Rotfärbungen zeigen die vollständigen Abdeckungen der Variablenkombinationen A-B-C, D-E-G und H-I-J [KKL <sup>+</sup> 10] . . . . .	22
2.2.	Testbestand $T$ . . . . .	24
2.3.	Struktur des IPOG-Algorithmus anhand des Beispiels aus Abschnitt 2.2.2) und $t = 2$ : Zunächst wird für $t$ Parameter eine Tabelle mit einer vollständigen Kombinationsabdeckung erstellt (linke Seite). Anschließend wird diese Tabelle horizontal (rechte Seite) und vertikal erweitert. Im Beispiel entfällt die vertikale Erweiterung, da bereits nach der horizontalen Erweiterung alle Variablenpaare vollständig abgedeckt sind. . . . .	33
3.1.	Die Parameter und möglichen Werte des Basis-Systems . . . . .	55
3.2.	Grenzen (in Euro) der möglichen Eingabewerte der Beitragsparameter in Easy Web in Abhängigkeit des Durchführungswegs, der Rückdeckung und der jeweiligen Tarife. Rot sind untere Schranken markiert, grün obere Schranken . . . . .	57
3.3.	Mögliche Werte für die verschiedenen Beitragsparameter für das Basis-System in Abhängigkeit des Durchführungswegs, der Art der Rückdeckung und des Tarifs. . . . .	59
3.4.	Die Parameter und möglichen Werte des Erweiterten Systems: Das Erweiterte System umfasst zusätzliche Parameter bezüglich der Finanzierungsart und der Unterscheidung in Arbeitgeber- und Arbeitnehmerbeitrag . . . . .	61
3.5.	Entstehende Problematik bei Einbezug des Durchführungswegs und der Art der Rückdeckung in die $t$ -fach-Kombinatorik bei der Erstellung von Testmengen: Für $t = 2$ wird durch den ersten Testfall die Kombination Tarif = 'AREven' / Beitrag = 600 bereits abgedeckt und somit möglicherweise nicht mehr beim Durchführungsweg Direktzusage verwendet. . . . .	62

## Tabellenverzeichnis

3.6. Modellhafte Umsetzung des Ansatzes der separaten Erstellung von Testmengen mit $t$ -fach-Abdeckung für jeden Durchführungsweg und jede Art der Rückdeckung. . . . .	63
3.7. Anzahl der Variablen-Wert-Kombinationen des Basis-Systems . . . . .	65
3.8. Beispielhafte Menge an Testfälle für das Basis-System . . . . .	66
3.9. Ergebnisse Random Testing . . . . .	67
3.10. Beispielhafte Menge an Testfälle für das Erweiterte System . . . . .	68
3.11. Vergleich der Ergebnisse des Basis-Systems und des Erweiterten Systems . .	68
3.12. Anzahl der Testfälle bei der Erstellung von separaten Testmengen für jeden Durchführungsweg und jede Art der Rückdeckung: Als Vergleichsgröße wird zudem die Anzahl der Testfälle des Vorgehens des Erweiterten Systems aufgeführt. . . . .	70
3.13. Ergebnisse IPOG-Algorithmus . . . . .	71
3.14. Ergebnisse IPOG-F-Algorithmus . . . . .	72
3.15. Ergebnisse PICT-Algorithmus . . . . .	72
3.16. Ergebnisse CASA-Algorithmus . . . . .	72
3.17. Vergleich aller Algorithmen / Tools in Bezug auf die minimale Anzahl an Testfälle . . . . .	73
A.1. Mögliche Werte für die verschiedenen Beitragsparameter für das Erweiterte System in Abhängigkeit des Durchführungswegs, der Art der Rückdeckung und des Tarifs. . . . .	89

# **1. Einleitung**

## **1.1. Motivation**

Software-Systeme steigen zunehmend in ihrer Komplexität und damit auch die möglichen Fehler bei der Nutzung – mit enormen Auswirkungen wie eine Studie des Consortium for Information & Software Quality (CISQ) [Kra21] aufzeigt: Für das Jahr 2020 schätzte das CISQ die Kosten unentdeckter Softwarefehler für die US-amerikanische Wirtschaft auf Basis öffentlicher Nachrichten und Informationen auf 1,56 Billionen US-Dollar, 2018 lag dieser Wert noch bei 1,28 Billionen US-Dollar.

Im Zuge dessen rückt im Bereich der Software-Entwicklung vermehrt die Bedeutung adäquater Methoden des Softwaretestens in den Vordergrund mit dem Ziel mögliche Fehler frühzeitig zu erkennen und zu eliminieren. Dabei wurde in der Vergangenheit in vielen Fällen auf die Erfahrung und Expertise von Software-Entwickler\*innen vertraut, systematische Verfahren zur Generierung von Testfällen fanden nur wenig Beachtung. Als eine von verschiedenen systematischen Methoden zur Testfallgenerierung etabliert sich das Prinzip des Combinatorial Testing: Basierend auf dem Ansatz, dass die Interaktion einiger, weniger Parameter für ein Großteil aller Softwarefehler verantwortlich ist, ermöglicht Combinatorial Testing die effektive und kostengünstige Erstellung von Testbeständen und bietet zudem die Möglichkeit die Güte jener Testbestände auf mathematisch-kombinatorischer Ebene zu quantifizieren.

## **1.2. Ziel der Arbeit**

Ziel dieser Arbeit ist es, die Grundlagen des Softwaretestens und die Theorie des Combinatorial Testings herauszuarbeiten und basierend darauf am Beispiel eines Anwendungsfalls

## *1. Einleitung*

aus dem Bereich der aktuariellen Software die praktische Umsetzung von Combinatorial Testing auszuprobieren. Mittels verschiedener untersuchten Teilespekte soll aufgezeigt werden, wie Combinatorial Testing die Qualitätssicherung, im Speziellen die Generierung von adäquaten Testfällen, optimieren kann.

Als Anwendungsbeispiel dient die Beratungs- und Kalkulationsplattform für Lebensversicherungsprodukte der ALH-Gruppe 'Easy Web Leben' [ALHb], welche zur Beantwortung folgender, übergeordneten Fragestellung dieser Arbeit genutzt wird: Können Testfälle bei komplexen Systemen wie Easy Web effizient mittels kombinatorischer Methoden erzeugt werden und welche Vorteile ergeben sich bei der Nutzung der Prinzipien des Combinatorial Testing?

Unter anderem soll im Rahmen dieser Arbeit auch untersucht werden, welche Algorithmen und Tools zur Erstellung von Mengen an Testfällen existieren und in einem empirischen Vergleich die Nutzbarkeit verschiedener Algorithmen und Tools verglichen werden.

## **1.3. Verwandte Arbeiten**

Combinatorial Testing hat sich in den vergangenen Jahren zu einer verbreitete Methode zum Testen verschiedener Eingabe- und Konfigurationsmöglichkeiten von Software entwickelt: Verschiedene Algorithmen und Tools wurden in der Vergangenheit vorgestellt, die eine möglichst schnelle und effiziente Nutzung der Methoden des Combinatorial Testing ermöglichen. Dies bewirkt insbesondere, dass die Nutzung der Methoden des Combinatorial Testing auch in Bereichen der Forschung und Industrie weit verbreitet ist: Verschiedenste Forschungsarbeiten wie [LGW<sup>+</sup>16, HWKK15, SJB<sup>+</sup>19, Ozc17, DKS14, RKK17] untersuchten für unterschiedlichste Anwendungsfälle die Nutzbarkeit und Vorteile von Combinatorial Testing. Über alle Forschungsarbeiten hinweg zeigt sich, dass Combinatorial Testing sowohl im Bereich der Wissenschaft als auch der Industrie einen Mehrwert bietet und zunehmend an Relevanz gewinnt.

Im Kontext der Finanz- und Versicherungswirtschaft begrenzen sich die Erkenntnisse im Wesentlichen auf die Arbeit von Mehta und Philip [MP13], in welcher die Anwendung von Combinatorial Testing am Beispiel verschiedener Softwarekomponenten des ehemaligen IT-Beratungsunternehmens iGATE (übernommen durch Capgemini) vorgestellt wird. Mehta und Philip konnten herausarbeiten, dass die Anwendung der Methoden des Combinatorial

## *1. Einleitung*

Testing die Anzahl benötigter Testfälle in erheblichem Maße reduziere, zugleich jedoch die Anwendung ein hohes Verständnis über die mathematisch-kombinatorischen Grundlagen erfordere und die richtige Wahl der Faktoren und ihrer Werte eine entscheidende Rolle spielen.

Die Untersuchungen dieser Arbeit fügt sich der aufgeführten Forschung an und soll insbesondere den Forschungsstand in Bezug auf die Anwendung im Kontext von aktuarieller Software erweitern. Darüber hinaus existieren in der Forschung keine empirischen Vergleiche aktuell verfügbarer Tools und Algorithmen in Bezug auf die Nutzung in einem praxisrelevanten Anwendungsfall.

## **1.4. Struktur der Arbeit**

Die Ausführungen dieser Arbeit folgen folgender Struktur: Zunächst werden in Kapitel 2 die theoretische Grundlagen des Softwaretestens (vgl. Abschnitt 2.1) und der Methode des Combinatorial Testing (vgl. Abschnitt 2.2) erläutert. Der Abschnitt zum Combinatorial Testing beinhaltet unter anderem auch die Vorstellung verschiedener Algorithmen und Tools zur Generierung von Mengen an Testfällen. Anschließend folgt in Kapitel 3 die Vorstellung des Anwendungsfall mit detaillierten Ausführungen zum Testobjekt 'Easy Web Leben' und relevanten versicherungstechnischen Grundlagen (vgl. Abschnitt 3.1). Darüber hinaus werden im Anschluss verschiedene Teilaspekte zur Beantwortung der grundlegenden Fragestellungen aufgegriffen und deren Implementierung (vgl. Abschnitt 3.2) und Ergebnisse vorgestellt (vgl. Abschnitt 3.3). Abschließend werden die Erkenntnisse der Resultate der untersuchten Teilfragestellungen in Kapitel 4 diskutiert und im Kontext der übergeordneten, grundlegenden Fragestellung dieser Arbeit eingestuft. Mögliche Limitierungen und Optionen für zukünftige Forschungsarbeiten werden ebenfalls aufgeführt.

## **2. Theoretische Grundlagen**

Im folgenden Kapitel werden die grundlegenden Konzepte und Methoden für die Beantwortung der Fragestellung dieser Arbeit vorgestellt. Dabei folgt zunächst ein grundlegender Einblick in die Begrifflichkeiten der Qualitätssicherung, des Softwaretestens und der Testfallerzeugung. Im zweiten Abschnitt dieses Kapitels wird das Konzept des Combinatorial Testing im Genauen erläutert, welches die wesentliche Strategie bei der Erzeugung von Testfällen des in Kapitel 3 vorgestellten Anwendungsfalls darstellt.

### **2.1. Einführung ins Softwaretesten**

Im Allgemeinen betrachtet ist das Softwaretesten ein Teilbereich der übergeordneten Qualitätssicherung, die 'alle qualitätsrelevanten Aktivitäten und Prozesse' zur Optimierung der Softwarequalität beinhaltet [LL10, S. 269]. Insbesondere stehen laut Ludewig und Lichter [LL10] bei der Qualitätssicherung diejenigen Maßnahmen im Vordergrund, die zu einem hohen Vertrauen in das Softwareprodukt bei allen beteiligten Stakeholdern beitragen sollen. Der Begriff der Qualität selbst kann dabei aus vielen verschiedenen Sichtweisen betrachtet werden, unter anderem ist eine Unterteilung in die Produktqualität und eine Projektqualität gebräuchlich [LL10, S. 66]. Dabei fokussiert sich die Produktqualität auf die Eigenschaften und Merkmale des letztendlichen Endprodukts im Rahmen der zuvor definierten Anforderungen, die Projektqualität blickt eher auf den Prozess der Entwicklung und Wartung des jeweiligen Produkts auf Prozessebene [LL10, S. 66 ff.].

Auf Basis dessen lässt sich eine hierarchische Auflistung der verschiedenen Ebenen, die Qualitätssicherung umfassen, erstellen: Ludewig und Lichter [LL10, S. 271 ff.] führen organisatorische, konstruktive und analytische Maßnahmen als wesentliche Ebenen der Qualitätssicherung auf. Diese lassen sich im Hinblick auf die zuvor eingeführte Definition des

## 2. Theoretische Grundlagen

Qualitätsbegriffs insofern zuordnen, als organisatorische Maßnahmen sich im Wesentlichen auf die Prozessqualität fokussieren, analytische Maßnahmen eher der Produktqualität zugeordnet werden können und konstruktive Maßnahmen in beiden Bereichen des Qualitätsbegriffs eine Rolle spielen.

Als konkrete Beispiele gelten in Bezug auf konstruktive und organisatorische Maßnahmen unter anderem ein gelungenes Projektmanagement (organisatorisch) oder die präventive Schulung von Mitarbeiter (konstruktiv) [LL10, S. 272]. Die analytischen Methoden können zudem in nicht-mechanische und mechanische Verfahren unterteilt werden. Nicht-mechanische Analysen können beispielsweise manuelle Code-Reviews durch separate Programmierer sein, mechanische Analyse umfassen statische Analysen wie die Compiler-Analysen (beispielsweise Pufferüberläufe) und die Methoden des Softwaretestens. Da beim Softwaretesten im Gegensatz zu den statischen Analysen ein Testobjekt auf einem Rechner ausgeführt wird, spricht man in diesem Fall auch von dynamischen Softwaretests [SL19, S. 135]. Abbildung 2.1 zeigt die hierarchische Struktur der verschiedenen Komponenten der Qualitätssicherung im Software-Bereich.

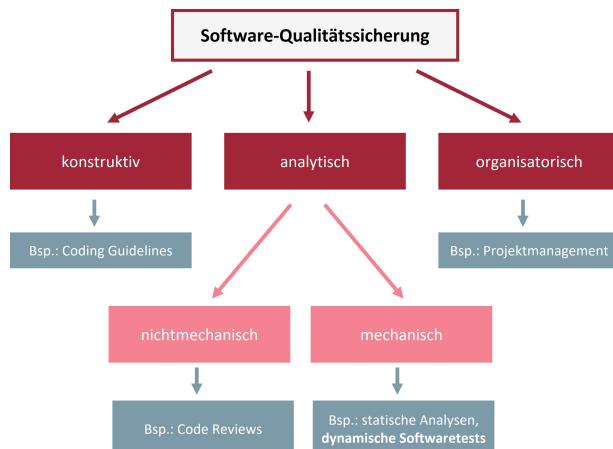


Abbildung 2.1.: Die verschiedenen Komponenten der Qualitätssicherung in der Software-Entwicklung in der Übersicht [LL10, S. 271].

Als Teilgebiet der Qualitätssicherung ist das dynamische Softwaretesten also ein wesentlicher Bestandteil bei der Sicherstellung der Produktqualität, deren wichtigsten Anforderungskriterien der ISO-/IEC-Standard 25010 [Int11] als internationale Norm für Software und IT-Systeme zusammenfasst: Im Kern werden dabei von Softwareprojekten eine hohe Funktionalität, Effizienz, Sicherheit, Kompatibilität, Verlässlichkeit, Usability, Wartbarkeit und Portierbarkeit gefordert.

## 2. Theoretische Grundlagen

Dynamische Softwaretests besitzen demnach im übergeordneten Sinne die Aufgabe, diese Anforderungen zu prüfen und sicherzustellen. Im Konkreten bedeutet dies, dass dynamische Softwaretests anhand vordefinierter Testfälle mögliche Fehler vor der tatsächlichen Nutzung der Software aufdecken sollen und zudem überprüfen, ob ein Softwaresystem die an das System gestellte Spezifikationen erfüllt [Som12, S. 246]. Testfälle werden typischerweise entweder anhand eines systematischen Vorgehens künstlich erstellt oder ergeben sich aus Erfahrungswerten beziehungsweise der laufenden Produktion (vgl. dazu Abschnitt 2.1.2).

Der Begriff des Fehlers kann dabei im Rahmen des Testens von Software vielschichtig interpretiert werden und nimmt unterschiedliche Rollen ein: Ein Fehlverhalten (engl.: failure) bezieht sich auf ein unerwartetes Ergebnis eines Programms, ein Codefehler (engl.: defect) beschreibt die zugehörigen fehlerhaften Codezeilen und ein Denkfehler (engl.: error) ist ein konzeptioneller Fehler, welcher vor der tatsächlichen Implementierung erfolgt. Alle drei Fehlerdefinitionen sind eng miteinander verknüpft: Ein Denkfehler kann bereits in der Konzeption dazu führen, dass trotz einer mutmaßlich korrekten Implementierung Codefehler und somit unterschiedliches Fehlverhalten auftreten kann. Gleichzeitig kann es auch bei der Implementierung durch menschliches Versagen zu Codefehlern kommen, die in der Konzeption korrekt waren und somit keinen Denkfehler darstellen. Abbildung 2.2 zeigt die logische Struktur dieser Begrifflichkeiten auf und greift diese Thematik auf.

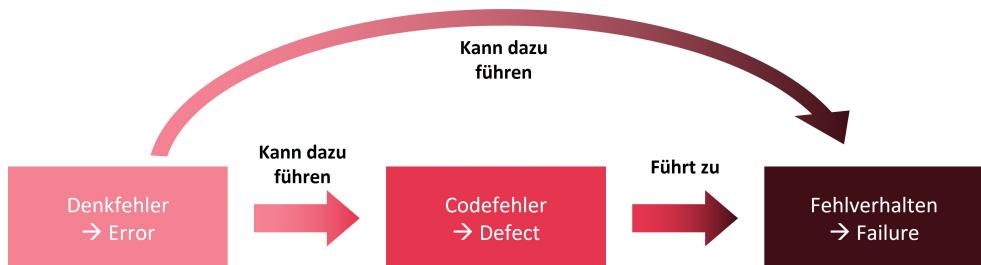


Abbildung 2.2.: Ein Denkfehler bei der Implementierung führt zu einem Codefehler, welcher wiederum ein Fehlverhalten der Software auslöst.

Grundsätzlich gilt es zu berücksichtigen, dass dynamische Softwaretests nicht in der Lage sind, alle Fehler eines Systems vollumfänglich aufzudecken [Som12, S. 247]. Dafür sind die zu testenden Systeme meist zu komplex und vollumfängliche Tests, wie in Abschnitt 2.1.1.3 aufgeführt, nicht umsetzbar. Software-Engineering Pionier Edward Dijkstra stellte in diesem Zusammenhang bereits 1972 fest: 'Tests können nur die Anwesenheit von Fehlern anzeigen, nicht ihre Abwesenheit' [DDH72].

## *2. Theoretische Grundlagen*

Spillner et al. [SLS11, S. 8] fassen die Kernaufgaben, welche dynamische Softwaretests erfüllen sollen, unter folgenden Aspekten zusammen:

- Entdeckung von Fehlverhalten innerhalb der betroffenen Software
- Schaffung von Vertrauen in das System bei allen betroffenen Stakeholdern
- Anregung frühzeitiger Analyse und Dokumentation in der Entwicklung und Wartung um Codefehler zu verhindern

### **2.1.1. Testfall-Design**

Dynamische Softwaretests werden durch mehrere konkrete Tests an einem bestimmten Testsystem durchgeführt. Jeder Test unterliegt dabei einem spezifischen Testfall, der alle erforderlichen Komponenten zur Durchführung eines Tests bereithält. Diese sind die komplette Systemumgebung, Eingabedaten und das erwartete Verhalten beziehungsweise die erwartete Ausgabe des Systems [Sch12, S. 86].

Ein Testfall sollte dabei auf sinnvolle Art und Weise entwickelt werden: Dazu gehört laut Ludewig und Lichter [LL10, S. 480] eine eindeutig definierte Systemumgebung, ein systematisches Vorgehen bei der Wahl der Eingabedaten, klare Kriterien zur Testevaluation und eine detaillierte Dokumentation der Testergebnisse. Wenn diese vier Aspekte auf einen Testfall zutreffen, sprechen Ludewig und Lichter von einem 'systematischen Test'.

Eine der wesentlichen Herausforderungen des dynamischen Softwaretestens besteht darin, ein derartiges systematisches Vorgehen zu entwickeln, um Testfälle möglichst effizient zu erzeugen: Grundsätzlich ist das Ziel des Testfall-Designs mit möglichst wenig Aufwand, also einer geringen Anzahl an Testfällen, möglichst viele Fehler im Sinne eines Fehlverhaltens zu entdecken [LL10, S. 498]. Dabei sollte laut Ludewig und Lichter [LL10, S. 498] ein Testfall möglichst repräsentativ für viele andere Testfälle sein und zudem idealerweise sehr fehlersensitiv und redundanzarm.

Um diese Anforderungen abdecken zu können wurden in der Vergangenheit verschiedene Techniken ausgearbeitet, die ein systematisches und effizientes Testfall-Design vereinfachen sollen. Dabei entstanden verschiedene Grundprinzipien, die sich aus einer unterschiedlichen Betrachtungsweise auf ein Softwaresystem ergeben. Diese sind nicht komplementär zu verstehen, sondern vielmehr ergänzend zur genauen Spezifizierung der Testfälle. Im Folgenden

## 2. Theoretische Grundlagen

werden drei relevante Betrachtungsweisen vorgestellt: Stufenbasiertes Testen, Black Box- & White-Box-Tests und Abdeckungsbasiertes Testen.

Konkrete Testmethoden wie beispielsweise Combinatorial Testing (vgl. Abschnitt 2.2) oder die Äquivalenzklassenmethode (vgl. Abschnitt 2.1.2) lassen sich demnach meist unterschiedlichen Bereichen der aufgeführten Betrachtungsweisen zuordnen: So ist die Äquivalenzklassenmethode genauso wie das Combinatorial Testing auf fast allen Stufen des stufenbasierten Testens anwendbar, aber zugleich dem Black-Box-Ansatz zuzuordnen. Im Bereich des abdeckungsbasierten Testens spielt die Äquivalenzklassenmethode zudem bei der Abdeckung der Eingabewerte eine wichtige Rolle.

### 2.1.1.1. Stufenbasiertes Testen

Das stufenbasierte Testen ist eine der abstraktesten Betrachtungsweisen im Hinblick auf die Erstellung von Testfällen: Basierend auf verschiedenen Stufen innerhalb eines Softwareprojekts werden beim stufenbasierten Testen Testfälle passend zu den Anforderungen und Aktivitäten im jeweiligen Stadium des Projekts erarbeitet [AO08, S. 5].

Dabei kann auf verschiedene Stufen klassischer Prozessmodelle in der Softwareentwicklung zurückgegriffen werden: Häufig dient dabei das V-Modell als Basis, das im Auftrag des Bundesministeriums für Verteidigung entwickelt wurde und in der Softwarewelt in angepasster Form verbreitete Anerkennung fand [LL10, S. 190]. Das V-Modell ist als eine Weiterentwicklung des von Royce [Roy87] 1970 entwickelten Wasserfallmodells zu verstehen: Dieses sieht vor, dass die Entwicklung eines Produkts im Allgemeinen in verschiedene Phasen unterteilt werden, bei der jede vorhergehende Phase essenzielle Grundlage der folgenden Phase ist [Som12, S. 57 f.]. Das V-Modell erweitert diesen Ansatz, indem aus jeder dieser Phasen des V-Modells Testfälle passend zu den jeweiligen Aktivitäten dieser Phase abgeleitet werden [LL10, S. 190].

Üblicherweise werden dabei die folgenden Phasen aus dem Wasserfall- und V-Modell übernommen und daraus resultierende Testfälle unterschieden [AO08, S. 5 f.]. Abbildung 2.3 visualisiert die hierarchische Struktur und das Zusammenspiel der verschiedenen Stufen des V-Modells:

- Anforderungsanalyse & Abnahmetests: Zu Beginn eines jeden Softwareprojekts werden die konkreten Anforderungen in Zusammenarbeit mit dem Auftraggeber herausgearbeitet. Diese Anforderungen werden in einem abschließenden Stadium eines

## *2. Theoretische Grundlagen*

Projekts anhand von Abnahmetests mittels echter Daten von tatsächlichen oder potenziellen Nutzern getestet.

- Systemarchitektur & Systemtest: Beim Systementwurf werden konkrete Designentscheidungen bezüglich der grundlegenden Architektur von Software beschlossen, insbesondere die Zuordnung bestimmter Anforderungen an unterschiedliche Hard- und Softwarekomponenten. Aus dieser Phase resultieren Systemtests, die sämtliche Komponenten des Projekts auf Basis der Architektur testen.
- Systementwurf & Integrationstest: Während der Systementwurf sich auf die grundlegende Architektur fokussiert, werden in der Phase des Subsystementwurfs die konkreten Strukturen und Komponenten der zu entwickelnden Software spezifiziert. Um die Interoperabilität dieser verschiedenen Komponenten zu testen, werden Integrationstests formuliert.
- Software-Design/Implementierung & Modul-/Unit-Tests: Die Implementierung umfasst die konkrete Umsetzung des Softwareentwurfs in Programmcode. Jede der einzelnen Komponenten, die meist unabhängig voneinander entwickelt werden, können mittels Unit-Tests bezogen auf kleine Einheiten wie Packages, Methoden oder Klassen getestet werden. Je nach Komplexität des Softwaresystems können weitere übergeordnete Module oberhalb der kleinsten Einheiten ausgearbeitet werden. Bevor letztlich mehrere Komponenten gemeinsam getestet werden, können einzelne Module vorgeschaltet in Modultests geprüft werden.

Die Umsetzung des stufenbasierten Testens muss jedoch nicht unbedingt anhand der vorgegebenen Stufen des V-Modells erfolgen, sondern erlaubt auch selbstdefinierte Phasenmodelle: Craig und Jaskiel [CJ02, S. 98 f.] führen beispielsweise bestimmte Produktrisiken, personelle oder zeitliche Anforderungen als wesentliche Faktoren bei der Stufenzbildung auf.

### **2.1.1.2. White Box- & Black Box-Tests**

Neben der Fokussierung auf die verschiedenen Phasen im Softwareentwicklungsprozess orientiert sich die Literatur bei der Erstellung von Testfällen in den meisten Fällen an der Unterscheidung zwischen Black-Box- und White-Box-Tests. Dabei bezieht sich der Begriff der White-Box beziehungsweise Black-Box darauf, inwiefern die Struktur der zu testenden Softwareeinheit bei der Erstellung der Testfälle bekannt ist.

## 2. Theoretische Grundlagen

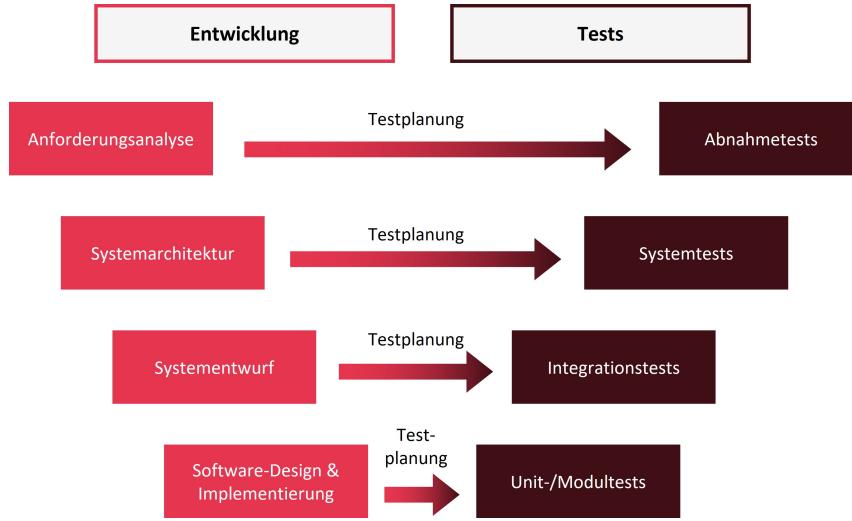


Abbildung 2.3.: Das V-Modell dient als Grundlage für das stufenbasierte Testen. Verschiedene Phasen im Entwicklungsprozess einer Software werden mit unmittelbaren Testfällen verknüpft [CJ02, S. 101].

**Black Box-Tests** Beim Black-Box-Testen ist die Struktur der zu testenden Softwareeinheit wie beispielsweise der Aufbau des Programmcodes unbekannt, einzig und allein Beschreibungen der Software-Spezifikationen dienen als Grundlage für den Testfall [Sch12, S. 91]. Die Spezifikationen können dabei je nach Entwicklungsstadium im Stufenmodell (vgl. Abschnitt 2.1.1.1) unterschiedlich aussehen und somit zu verschiedenen Testfällen führen: So basieren Abnahmetests im Wesentlichen auf dem Black-Box-Ansatz, werden aber auf eine gänzlich andere Art und Weise durchgeführt als Unit-Tests, die ebenfalls im Rahmen eines Black-Box-Ansatzes umgesetzt werden können.

Laut Schneider [Sch12, S. 91] sollen im Rahmen von Black-Box-Tests alle definierten Anforderungen an ein System getestet werden: Meist liegen diese nur in Form von Fließtext oder Tabellen vor und müssen dementsprechend in sinnvolle Testfälle umgewandelt werden [Sch12, S. 92]. Im Falle der Stufe der Anforderungsanalyse und Abnahmetests sind diese schriftlichen Beschreibungen beispielsweise textuelle Erläuterungen der Softwarefunktionalität, bei der Implementierung und Unit-Tests können dies unter anderem Methodenkommentare sein.

Verschiedene Methoden wie beispielsweise die Äquivalenzklassenmethode, die Grenzwertanalyse oder das zustandsbasierte Testen können den Black-Box-Tests zugeordnet werden (vgl. [Sch12, S. 94 ff.]). Details hierzu folgen in Abschnitt 2.1.2.

## 2. Theoretische Grundlagen

**White Box-Tests** Im Gegensatz zu den Black-Box-Tests setzen White-Box-Tests, oder auch Glass-Box-Tests genannt, voraus, dass die zu testende Softwareeinheit durchsichtig in ihrer Struktur ist und somit Testfälle auf Basis dessen erstellt werden können [Sch12, S. 91]. Konkret möchte man laut Schneider [Sch12, S. 91] bei White-Box-Tests kritische Stellen im Programmcode oder an den Schnittstellen verschiedener Softwarekomponenten anhand einer Analyse der vorhandenen Softwarestruktur entdecken.

Die meiste Verwendung finden White-Box-Test auf der Modul- und Implementierungsebene. Dort entsteht laut Schneider durch den hohen Grad an individueller Entwicklungsarbeit oftmals eine hohe Komplexität im Softwaresystem, welche zu schwer ermittelbaren Fehlverhalten im Code führen kann [Sch12, S. 108]. Daher soll laut Schneider [Sch12, S. 108 ff.] mit White-Box-Tests im Wesentlichen sichergestellt werden, dass alle geschriebenen Programmteile auch tatsächlich verwendet werden. Dies wird meist mittels Maßen für die Codeüberdeckung umgesetzt: Metriken der Anweisungsüberdeckung, Zweigüberdeckung und Pfadüberdeckung werden dafür in vielen Fällen als Gütemaß herangezogen [Sch12, S. 109 ff.].

Die Messbarkeit der Güte eines Testverfahrens steht zunehmend nicht nur bei den Methoden des White-Box-Testens im Vordergrund wie der folgende Unterunterabschnitt zum abdeckungsbasierten Testen aufzeigt: Die Messung der Anweisungsüberdeckung, Zweigüberdeckung und Pfadüberdeckung stellt dabei einen relevanten Vertreter der Abdeckungsprüfung von Testverfahren im Kontext von Graphen-basierter Abdeckung dar.

### 2.1.1.3. Abdeckungsbasiertes Testen

Während sich die Literatur in der Vergangenheit bei der Testfallentwicklung im Wesentlichen auf die zuvor aufgeführten Perspektiven der Phasen auf Prozessebene (vgl. Abschnitt 2.1.1.1) und der Sichtbarkeit der Struktur von Software (vgl. Abschnitt 2.1.1.2) fokussierte, rückt zunehmend eine weitere dritte Betrachtungsweise in den Fokus der Forschung: Auf Basis von verschiedenen Metriken, welche Abdeckungen bestimmter Eigenschaften des Softwaresystems messen, steht beim abdeckungsbasierten Testen vor allem die Quantifizierung der Güte eines Testverfahrens zur Sicherstellung der Softwarequalität im Fokus [CJ02]. Diese quantitativen Verfahren können unter anderem helfen, verschiedene Stakeholder eines Softwareprojekts von der Güte eines Testbestandes zu überzeugen [KKL<sup>+</sup>10].

## *2. Theoretische Grundlagen*

Neben dem Ansatz, durch verlässliche Metriken Vertrauen in eine Menge von Testfällen zu schaffen, basiert das Konzept des abdeckungsbasierten Testens auf der Erkenntnis, dass ein vollständiges Testen aller denkbaren Zustände, die ein Softwaresystem einnehmen kann, unmöglich ist [AO08, S. 16 f.]. Konkret führen Ammann und Offutt [AO08, S. 16] das Beispiel eines Java Compilers an, dessen Limitierung durch die maximale Kapazität des Compiler-Parsers gegeben ist. Der Java Compiler kann theoretisch jede beliebige Zeichenfolge einer .java-Datei aufnehmen, versucht diese anhand seiner internen Logik zu interpretieren und daraus ein ausführbares Programm in Maschinencode zu erzeugen. Die Länge der Eingabe ist durch die Menge an Zeichen limitiert, die der zugehörige Parser aufnehmen kann: Dies entspricht im Falle von Java bei einer maximalen Dateimenge von 64 kB pro Methode [Dev21] und einer UTF-8-Kodierung mit 1 bis 4 Bytes pro Zeichen mindestens einer Anzahl von 16.000 Zeichen bei voller Auslastung der 64 kB. Berücksichtigt man nun noch alle denkbaren Kombinationen, die durch Permutieren der 144.697 aktuell im Unicode belegten Zeichen [uni] zustande kommen können, so ergibt sich eine Mindestanzahl von  $16.000^{144.697}$  verschiedenen Kombinationen, die ein Java-Parser berücksichtigen müsste: Eine Dimension, die kein Rechner auf dieser Welt testen kann.

Dementsprechend können Testfälle meist nur einen Anteil aller denkbaren Kombinationen abdecken, sodass es sehr hilfreich ist, die Qualität einer Menge an Testfällen beurteilen zu können: Dies wird beim abdeckungsbasierten Testen durch die Nutzung von quantifizierbaren Metriken vorgenommen [AO08, S. 17]. Jeder Testfall kann bei seiner Durchführung ein Testkriterium erfüllen oder auch nicht, sodass sich über die gesamte Menge aller zusammenhängender Testfälle ein Anteil ergibt, der die Erfüllung des Testkriteriums im Gesamten quantifiziert [AO08, S. 17].

Konkret bedeutet dies: Für jedes Abdeckungskriterium lässt sich bestimmen, welche Eigenschaften die Menge aller Testfälle benötigt, um das Kriterium vollständig zu erfüllen. Bei Combinatorial Testing sind dies beispielsweise alle  $t$ -fachen Kombinationsabdeckungen (vgl. Abschnitt 2.1.1.3) der EingabevARIABLEN. Die Quantifizierung der Güte der Menge aller Testfälle ergibt sich schließlich durch die Quote der erfüllten Eigenschaften, die das Abdeckungskriterium fordert. Jeder Testfall kann dann seinen Teil dazu beitragen, dass das betroffene Kriterium erfüllt wird. Dabei kann es passieren, dass mehrere Testfälle den 'gleichen' Beitrag im Sinne des Testkriteriums liefern. In anderen Worten: Manche Testfälle sind überflüssig, da die geforderte Eigenschaft des Abdeckungskriteriums bereits durch einen anderen Testfall oder die Kombination mehrerer Testfälle abgedeckt wird.

## *2. Theoretische Grundlagen*

Ammann und Offutt [AO08] unterscheiden vier wesentliche Kategorien von Abdeckungen im Bereich des dynamischen Softwaretestens.

- Graphen-basierte Abdeckung orientiert sich an verschiedenen Graphstrukturen, die sich im Rahmen eines Softwarekonzepts ergeben wie beispielsweise die Verzweigungen und Wiederholungen innerhalb des Programmcodes. Dabei können Metriken wie Knoten- oder Kantenabdeckungen zur Quantifizierung der Abdeckung herangezogen werden [AO08, S. 27 ff.].
- Logische Abdeckung basiert auf den Prinzipien der Prädikatenlogik, die sich analog zu Graphstrukturen auf verschiedene Bereiche eines Softwaresystems anwenden lassen, unter anderem die Verzweigungen innerhalb des Programmcodes oder die Umformulierung von Anforderungsspezifikation in Wenn-dann-Verbindungen [AO08, S. 131 ff.]. Die Abdeckung der daraus abgeleiteten Testfälle können mit Methoden der Prädikatenlogik wie beispielsweise der Klausel- und Prädikatenabdeckung gemessen werden [AO08, S. 106 ff.].
- Die Abdeckung der möglichen Eingabewerte berücksichtigt die Grundvoraussetzung jedes Softwaresystems: Jedes Programm verwendet in gewisser Weise Eingaben, so dass alle beliebigen Softwaretests auch Elemente des Eingabeuniversums verwenden [AO08, S. 150]. Insbesondere dann, wenn die Struktur einer Software unbekannt ist, stellt die Analyse der möglichen Eingaben und Ausgaben gemeinsam mit der Anwendung von Erfahrungswerten die einzige Möglichkeit zum Testen von Softwareeinheiten dar. Im Vergleich zu den anderen abdeckungsbasierten Ansätzen stehen beim expliziten Testen der Eingabewerte die kombinatorischen Eingabemöglichkeiten im Vordergrund [AO08, S. 150 ff.]. Combinatorial Testing ist dabei die relevanste Methode zur Umsetzung dieser Strategie, welche im Folgenden in Abschnitt 2.2 ausführlich vorgestellt wird.
- Syntax-basierte Abdeckung verwendet die Ideen der Automatentheorie und benutzt dabei syntaktische Beschreibungen von zu testenden Softwareeinheiten, um Testfälle abzuleiten [AO08, S. 170 ff.]. Im Zusammenhang mit Programmcode spielt vor allem der Begriff der Mutationen eine wichtige Rolle, bei dem elementare Codestücke durch leichte Veränderungen ersetzt werden und so Fehlverhalten provoziert wird. Als Metrik werden dabei zumeist die Abdeckung der Symbole und Produktionsregeln und weitere komplexe Kombinationen verwendet [AO08, S. 172].

## 2. Theoretische Grundlagen

### 2.1.2. Beispiele für Teststrategien

Der folgende Abschnitt greift die grundlegende Herangehensweise in Bezug auf das dynamische Softwaretesten aus Abschnitt 2.1.1 auf und führt einige konkrete Beispiele für Teststrategien ein. Diese Beispiele beschränken sich auf mögliche Anwendungen im Zusammenhang mit dem in Kapitel 3 vorgestellten Anwendungsfällen. Da dieser sich im Wesentlichen auf einen spezifikationsbezogenen Ansatz fokussiert, werden insbesondere die Techniken des White-Box-Testens (vgl. Abschnitt 2.1.1.2) an dieser Stelle ausgeklammert. Detaillierte Ausführungen zu den verbreiteten White-Box-Strategien der Anweisungs-, Zweig- und Pfadüberdeckung lassen sich in [Sch12, S. 108 ff.] nachlesen.

Da der Fokus dieser Arbeit auf Combinatorial Testing liegt, werden zudem die Prinzipien dieser Teststrategie nicht an dieser Stelle, sondern im folgenden, separaten Abschnitt 2.2 vorgestellt.

#### 2.1.2.1. Äquivalenzklassenmethode

Da vollständiges Testen - wie zuvor erläutert - nicht möglich ist, erfordern systematische Tests Einschränkungen bei der Wahl der Parameter. Die Äquivalenzklassenmethode löst dieses Problem derart, dass alle denkbaren Eingaben in verschiedene Äquivalenzklassen partitioniert werden [Sch12, S. 94]. Dabei sollten laut Schneider [Sch12, S. 94 ff.] die unterschiedlichen Klassen derart konstruiert werden, dass Vertreter aus derselben Klasse sich in Bezug auf das Fehlerverhalten innerhalb der Software identisch verhalten. Als Beispiel führt Schneider eine automatisierte Schwimmbadbakasse auf, die für Jugendliche unter 18 Jahren einen vergünstigten Preis anbietet. In diesem Fall würde sich eine Äquivalenzklasse für die Variable 'Alter' durch die Menge 'Jugendlich' =  $\{n < 18 \mid n \in \mathbb{N}\}$  und eine weitere Klasse 'Erwachsen' =  $\{n \geq 18 \mid n \in \mathbb{N}\}$  ergeben.

Je nach Spezifikation und denkbaren Eingaben müssen die Äquivalenzklassen auch mögliche Falscheingaben berücksichtigen. Eine wesentliche Herausforderung bei Anwendung der Äquivalenzklassenmethode besteht laut Schneider darin, eine passende Auswahl der Klassen zu treffen, die einerseits nicht zu kleinteilig, andererseits aber auch nicht zu pauschalisierend sein sollte [Sch12, S. 95].

## 2. Theoretische Grundlagen

### 2.1.2.2. Grenzwertanalyse

Die Grenzwertanalyse basiert auf der Annahme, dass es häufig unmittelbar an den Grenzen verschiedener Eingabeparameter zu Fehlverhalten kommt, da diese oftmals nicht genau definiert sind oder von Programmierenden nicht berücksichtigt werden [SLS11, S. 120]. Dementsprechend überprüft die Grenzwertanalyse die Grenzen kritischer Werte und stellt somit eine Erweiterung der Äquivalenzklassenmethode dar [SLS11, S. 120 f.].

Für jede Äquivalenzklasse ergeben sich laut Spillner et al. [SLS11, S. 120 f.] automatisch kritische Grenzen, die mittels der Grenzwertanalyse abgedeckt werden können. Im Beispiel der Schwimmbadkasse würde dies bedeuten, dass ein besonderes Augenmerk auf die Werte 17 und 18 gelegt werden sollte. Neben diesen offensichtlichen Grenzen der verschiedenen Äquivalenzklassen werden im Rahmen der Grenzwertanalyse häufig auch extreme, unerwartete Werte wie beispielsweise die maximal oder minimal mögliche Eingabe berücksichtigt werden [SLS11, S. 122]. Sowohl die Grenzwertanalyse als auch die Äquivalenzklassenmethode lässt sich auf allen Ebenen des Stufenmodells (vgl. Abschnitt 2.1.1.1) anwenden und ist in Bezug auf das abdeckungsbasierte Testen vor allem im Bereich der Abdeckung der Eingaben relevant.

### 2.1.2.3. (Adaptive) Random Testing

In einigen Fällen möchte man eine Softwareeinheit testen, bei der weder eine detaillierte Beschreibung der Spezifikation, noch der Programmcode selbst vorliegt. Dafür bietet sich die Verwendung des Zufallsprinzips an, welche die Grundlage des Random Testings ist. Dabei wird für das Eingabeuniversum eine Wahrscheinlichkeitsverteilung angenommen, aus welcher Eingabewerte zufällig entnommen werden [SLS11, S. 141 f.]. Grundsätzliche Vorteile einer zufallsbasierten Teststrategie ergeben sich laut Spillner [SLS11, S. 142] vor allem durch eine realitätsnahe Abbildung der tatsächlich verwendeten Werte der Softwareeinheit und durch die Möglichkeit, statistische Methoden zur Quantifizierung der Systemzuverlässigkeit anwenden zu können.

Unter anderem White und Cohen [WC80] konnten herausfinden, dass Fehlverhalten meist sehr gebündelt in Bereichen von Eingabewerten auftreten, sodass bereits durchgeführte Testfälle, die kein Fehlverhalten aufdeckten, möglichst weit im Eingabeuniversum von neuen Testfällen entfernt sein sollten [ABC<sup>+</sup>13]. Aus diesem Grund entwickelten Chen

## *2. Theoretische Grundlagen*

et al. [CLM04] die Methode des Adaptive Random Testing, welche versucht, die Testfälle möglichst weit und gleichmäßig über das Eingabeuniversum zu verteilen. Verschiedene Strategien und Algorithmen auf Basis von Distanzmaßen, Ausschlussverfahren oder evolutionären Algorithmen wurden in der Vergangenheit entwickelt, um Adaptive Random Testing in der Praxis umzusetzen [HXCL12].

### **2.1.2.4. Erfahrungsbasiertes Testen**

Neben den aufgeführten systematischen Ansätzen Testfälle zu Erzeugung existiert eine weitere Methode zur Testfallerzeugung, die in der Anwendungspraxis eine bedeutende Rolle spielt: Erfahrungsbasiertes Testen kann vor allem Fehler aufdecken, die systematische Ansätze übersehen [SL19, S. 210]. Das Wissen und die Expertise der testenden Person bildet dabei die Grundlage, um mögliche Probleme innerhalb einer Softwareeinheit mit Tests aufzudecken. Spillner et al. [SL19, S. 213] betonen, dass erfahrungsbasiertes Testen nicht als erstes Mittel der Wahl bei der Erstellung von Testfällen verwendet werden sollte, sondern vielmehr als Ergänzung anderer systematischer Ansätze zu verstehen ist.

Insbesondere interagiert das erfahrungsbasierte Testen häufig mit anderen konkreten Testmethoden wie beispielsweise der Äquivalenzklassenmethode oder der Grenzwertanalyse: So stellt die Wahl der Äquivalenzklassen beziehungsweise die Wahl kritischer Grenzwerte eine wesentliche Herausforderung der beiden Testmethoden dar, sodass an dieser Stelle Erfahrungswerte und Expertise besonders hilfreich sein können.

Erfahrungsbasiertes Testen lässt sich nur teilweise den drei zuvor aufgeführten Grundprinzipien des Testens zuordnen und kann selbst nur als 'semi-systematisches' Verfahren bezeichnet werden: Die Methode lässt sich nicht eindeutig als Black Box- oder White Box-Technik einstufen [SL19, S. 213], auch ein Abdeckungskriterium, welches die Güte von erfahrungsbasierten Tests messen kann, existiert in den meisten Fällen nicht. Die Expertise erfahrener Tester\*innen kann auf allen Ebenen des stufenbasierten Testens nützlich sein. Sie kommt jedoch meist auf höheren Teststufen zum Einsatz, da auf den niedrigeren Ebenen genügend Informationen über die Spezifikationen, wie beispielsweise der Programmcode selbst, zur Verfügung stehen [SL19, S. 213].

In der Praxis haben sich in Bezug auf erfahrungsbasiertes Testen verschiedene Teilaspekte herausgebildet [SL19, S.210 ff.]:

## *2. Theoretische Grundlagen*

- 'Error Guessing' (deutsch: Fehlerraten) basiert auf der Idee, Fehler aus der Vergangenheit oder in Zukunft zu erwartende Fehler bei der Testfallerstellung zu berücksichtigen [SL19, S.210 f.].
- Checklistenbasiertes Testen verwendet eine Checkliste zur Testfallerzeugung, die eine erfahrene Testperson in der Vergangenheit angelegt hat und die als Anleitung für zukünftige Testfälle dient [SL19, S. 211 f.]. In diesem Fall kann ein Abdeckungskriterium für die Güte einer Menge von Testfällen definiert werden, indem der Anteil der erfüllten Aspekte der Checkliste ermittelt wird [SL19, S. 211].
- Exploratives Testen ist ein Verfahren, das einen kontinuierlichen Prozess bestehend aus Testfallerzeugung und Auswertung jener Testfälle beschreibt [SL19, S. 211]. Über die Zeit hinweg entwickelt laut Spillner und Linz [SL19, S. 212] die testende Person zunehmend mehr Wissen und Verständnis über das Testobjekt und kann dieses für neue, verbesserte Testfälle anwenden. Darüber hinaus kann das Wissen des explorativen Testens angewendet werden, um systematische Tests zu entwickeln.

## **2.2. Combinatorial Testing**

Combinatorial Testing (deutsch: Kombinatorisches Testen) ist analog zu den Beispielen aus Abschnitt 2.1.2 eine Strategie zur systematischen Erzeugung von Testfällen. Da diese Methode in dieser Arbeit im Mittelpunkt steht, wird diese an dieser Stelle ausführlicher als die zuvor aufgeführten Teststrategien vorgestellt. Zunächst sollen dabei die grundlegende Motivation für Combinatorial Testing und die wesentlichen Prinzipien im Fokus stehen. Anschließend werden verschiedene Metriken zur Quantifizierung der Güte eines Testbestandes im Zusammenhang von Combinatorial Testing erläutert und verschiedene Algorithmen und Tools zur Anwendung von Combinatorial Testing vorgestellt.

### **2.2.1. Einführung in Combinatorial Testing**

Als wesentliche Motivation für die Verwendung kombinatorischer Methoden bei der Erzeugung von Testfällen ergibt sich, wie bei fast allen Testfallstrategien auch, die Erkenntnis, dass vollständige Tests in der Realität nicht umsetzbar sind (vgl. Abschnitt 2.1). Combinatorial Testing greift diese Tatsache auf und versucht auf effiziente Art und Weise das

## 2. Theoretische Grundlagen

Eingabeuniversum einer zu testenden Softwareeinheit möglichst sinnvoll mittels kombinatorischer Methoden im Sinne von quantifizierbaren Metriken (vgl. Abschnitt 2.1.1.3) abzudecken. Dabei kann Combinatorial Testing auf allen Ebenen des in Abschnitt 2.1.1.1 aufgeführten Stufenmodells zum Einsatz kommen, wie verschiedene Beispiele in diesem Abschnitt aufzeigen werden. Zudem lässt sich Combinatorial Testing als Black-Box-Technik einstufen, da die Struktur der zu testenden Softwareeinheit für den Testablauf irrelevant ist.

Grundannahme beim Combinatorial Testing ist es, dass selten die Eingaben einzelner Parameter eines Testobjekts für Fehler verantwortlich sind, sondern vielmehr die Interaktion einiger weniger Parameter zu häufigen Fehlern führt [KKL<sup>+</sup>10]. Kuhn et al. [KWG04] konnten in einer Analyse verschiedener Software-Systeme aufzeigen, dass die Interaktion von sechs oder weniger Parameter für fast alle auffindbaren Softwarefehler verantwortlich sind. Abbildung 2.4 zeigt die Resultate vier verschiedener Software-Systeme der Untersuchungen von Kuhn et al. [KWG04]: Für verschiedene eingebettete Systeme medizinischer Geräte mit einer Anzahl von  $10^3$  bis  $10^4$  Parametern und einer verteilten Datenbank der amerikanischen Raumfahrtbehörde NASA mit  $10^5$  Parametern war die Interaktion vier unterschiedlicher Variablen für 100% der aufgetretenen Fehler verantwortlich. Sechs verschiedene Variablen sorgten darüber hinaus für 100% der Fehler eines HTTP-Servers mit  $10^5$  Parametern und eines Webbrowsers mit  $2 \cdot 10^6$  Parameter.

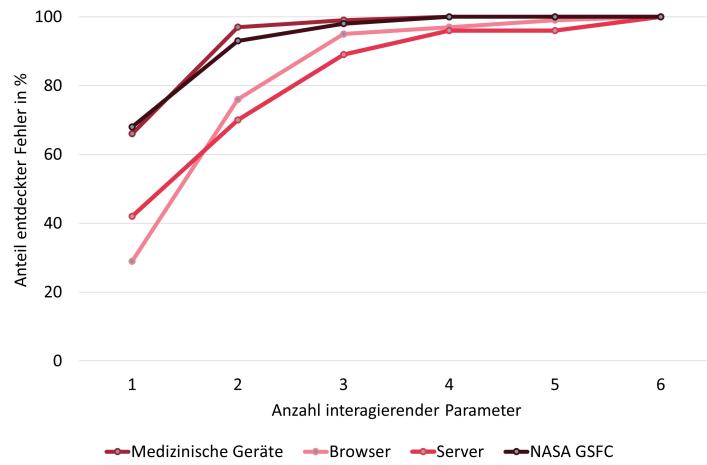


Abbildung 2.4.: Fehlerverhalten verschiedener Software-Systeme nach Untersuchungen von Kuhn et al. [KWG04]: Die Interaktion von sechs oder weniger Parameter ist demnach für 100% der Fehler der vier untersuchten Software-Systeme verantwortlich.

## 2. Theoretische Grundlagen

Das folgende Beispiel in Abbildung 2.5 soll die Problematik bei der Entstehung von Fehlern durch die Interaktion mehrerer Parameter verdeutlichen: Abbildung 2.5 zeigt ein einfaches Beispiel angelehnt an Kuhn et al. [KKL<sup>+</sup>10], bei welchem zwei Fehler unter verschiedenen Bedingungen entstehen: Fehler Nummer eins wird lediglich durch die Kombination 'Height' < 10 und 'Width' < 10 erzeugt (Codezeile 4), Fehler Nummer zwei durch Dreierkombination 'Length' ≥ 10 / 'Width' ≥ 10 / 'Height' ≥ 10 (Codezeile 11). Würde man die Werte der Parameter 'Length', 'Width' und 'Height' bei der Erstellung einer Menge an Testfällen zufällig auswählen, könnte es passieren, dass diese Kombinationen durch keinen der Testfälle abgedeckt wird und somit die Fehler unentdeckt bleiben.

```

1      if ('Height' < 10) {
2          // good code, no problem
3          if ('Width' < 10) {
4              // faulty code #1! BOOM!
5          }
6      } else if ('Height' >= 10) {
7          // good code, no problem
8          if ('Width' >= 10) {
9              // good code, no problem
10         if ('Length' >= 10) {
11             // faulty code #2! BOOM!
12         }
13     }
14 } else {
15     // good code, no problem
16 }
```

Abbildung 2.5.: Beispiel für die Interaktionsproblematik beim Testen verschiedener Parameter: Bei den Kombinationen 'Height' < 10 / 'Width' < 10 und 'Length' ≥ 10 / 'Width' ≥ 10 / 'Height' ≥ 10 kommt es zu einem Fehlverhalten [KKL<sup>+</sup>10].

Um diesem Problem zu begegnen, hat sich laut Kuhn et al. [KKL<sup>+</sup>10] in der Anwendungspraxis das sogenannte Pairwise-Testing etabliert, bei dem alle Kombinationen von Paaren der Eingabeparameter getestet werden. Im Beispiel von Abbildung 2.5 bedeutet dies, dass alle  $\binom{3}{2} = 3$  mögliche Parameterpaare mit ihren jeweils  $2^2 = 4$  verschiedenen Wertekonstellationen (jeder Parameter < 10 und ≥ 10) getestet werden sollten. Insgesamt ergibt sich dementsprechend für  $t = 2$  im Beispiel von Abbildung 2.5 eine Anzahl von 12 zu testenden Variablen-Wert-Kombinationen. Eine Abdeckung via Pairwise-Testing wür-

## 2. Theoretische Grundlagen

de gesichert Fehler Nummer 1 in Abbildung 2.5 aufdecken, jedoch nicht garantieren, dass auch Fehler Nummer zwei entdeckt wird.

Dieses Problem löst der Ansatz des Combinatorial Testing als Erweiterung des Pairwise-Prinzips derart, dass die Zahl des zu prüfenden Interaktionsparameters variabel ist [KKL<sup>+</sup>10]. Konkret entspricht ein Combinatorial Testing-Ansatz mit Interaktionsparameter  $t = 2$  dem Pairwise-Konzept, bei  $t = 3$  muss jede denkbare Dreierkonstellation der Eingabeparameter mindestens in einem Testfall berücksichtigt werden. Im Beispiel aus Abbildung 2.5 existieren drei verschiedenen Variablen, weshalb bei  $t = 3$  als einzige Parameterkonstellation diejenige Kombination bestehend aus allen drei Variablen 'Length', 'Width' und 'Height' abgedeckt werden muss.: Die Gesamtzahl der zu prüfenden Wertekombinationen der drei Variablen ergibt sich aufgrund der binären Trennung der Variablen in einen Bereich  $> 10$  und einen Bereich  $\leq 10$  als  $\binom{3}{3} \cdot 2^3 = 8$ . Unter diesen acht acht Wertekombinationen befindet sich schließlich auch die Kombination 'Length'  $\geq 10$  / 'Width'  $\geq 10$  / 'Height'  $\geq 10$ , weshalb Fehler Nummer zwei in Abbildung 2.5 gesichert entdeckt werden würde.

Aus mathematischer Sicht lassen sich eine Mindest- und Maximalanzahl notwendiger Tests ( $T_{\min}, T_{\max}$ ) zur zuvor beschriebenen  $t$ -fachen Kombinationsabdeckung (vgl. dazu auch Abschnitt 2.2.2.4) anhand der Wahl des Interaktionsparameters  $t$ , der Anzahl  $n$  aller Eingabeparameter  $x_i$  und der Anzahl der möglichen Ausprägungen  $y_i$  pro Parameter ermitteln: Unter der Annahme, dass der Interaktionsparameter  $t$  maximal der Anzahl  $n$  der vorhandenen Parameter entspricht, stellt das Produkt der  $t$  größten Werte der verschiedenen Parameterausprägungen  $y_i$  eine untere Schranke für die notwendigen Tests zur  $t$ -fachen Kombinationsabdeckung dar. Seien die Parameter also nach der Anzahl ihrer möglichen Ausprägungen geordnet, es gelte also  $y_1 \geq y_2 \geq \dots \geq y_n$ . Dann gilt:

$$T_{\min} = \prod_{i=1}^{\min\{n,t\}} y_i \quad (2.1)$$

Eine obere Schranke für die Anzahl der Tests ist unterdessen die Anzahl aller denkbaren Kombinationen der Eingabeparameter:

$$T_{\max} = \prod_{i=1}^n y_i \quad (2.2)$$

## 2. Theoretische Grundlagen

Eine große Spannweite zwischen  $T_{\min}$  und  $T_{\max}$  entsteht insbesondere dann, wenn die Differenz zwischen der Anzahl der Parameter  $n$  und des Interaktionsparameters  $t$  groß wird. Im Gegensatz dazu liegen  $T_{\min}$  und  $T_{\max}$  nah beisammen, wenn  $n$  und  $t$  ebenfalls dicht aneinander liegen. Im Besonderen gilt für  $t = n$ , dass  $T_{\min} = T_{\max}$ . Combinatorial Testing kann seine Vorteile vor allem dann ausspielen, wenn die Differenz zwischen  $T_{\min}$  und  $T_{\max}$  sehr groß ist: Durch die Fokussierung auf die Interaktion von  $t$  Parametern können in einem Testfall mehrere Parameterkombinationen gleichzeitig abgedeckt werden, was die Anzahl der tatsächlich benötigten Tests in Relation zu  $T_{\max}$  sehr nahe an der unteren Schranke  $T_{\min}$  hält.

Das in Tabelle 2.1 aufgeführte Beispiel aus [KKL<sup>+</sup>10] verdeutlicht dieses Prinzip: Die Tabelle zeigt eine mögliche Reihe an Testfällen für  $n = 10$  binäre ( $y_i = 2 \forall i = 1, \dots, n$ ) Eingabeveriablen A bis J bei einer Abdeckung aller Dreierkombinationen, also  $t = 3$ . Eine derartige Tabelle mit einer vollständigen Abdeckung aller  $t$ -fachen Kombinationen der Eingabeparameter nennt man im Kontext des Combinatorial Testing auch Covering Array oder Orthogonal Array.

Würde man alle möglichen Kombinationen der 10 Variablen abdecken wollen, würde sich eine Anzahl von  $T_{\max} = 2^{10} = 1024$  Testfällen ergeben. Dadurch, dass nur die Kombinationen aus drei Parametern berücksichtigt werden müssen, kann die Anzahl der Testfälle auf 13 reduziert werden. Beispielhaft wird in den drei unterschiedlichen roten Farbtönen anhand der Parameterkombinationen A-B-C, D-E-G und H-I-J aufgezeigt, inwiefern mit einem Testfall mehrere Konstellationen abgedeckt werden. Im Konkreten wird durch den ersten Testfall nicht nur für die Parameterkombination A-B-C das Werte-Tupel (0,0,0) abgedeckt, sondern auch für alle  $\binom{n}{t} = \binom{10}{3} = 120$  Parameterkombinationen – unter anderem auch für die Konstellation D-E-G und H-I-J. Gleiches gilt für den zweiten Testfall und das Wertetupel (1,1,1). Testfall Nummer drei sorgt bei der Parameterkonstellation A-B-C für keine zusätzliche Abdeckung, im Falle der Kombinationen D-E-G und H-I-J werden jedoch die Wertetupel (0,1,0) und (0,0,1) neu abgedeckt.

Dieses Vorgehen lässt sich über die 13 Testfälle in Tabelle 2.1 hinweg ausweiten und auf alle 120 Parameterkombinationen übertragen, sodass letztlich alle Binärkonstellationen aus drei Variablen vollständig berücksichtigt werden. Es zeigt sich also, dass in diesem Fall die tatsächlich benötigte Anzahl an Testfällen nahe an der unteren Schranke  $T_{\min} = 2^3 = 8$  liegt, insbesondere im Vergleich zu  $T_{\max}$ .

## 2. Theoretische Grundlagen

A	B	C	D	E	F	G	H	I	J
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Tabelle 2.1.: Ein Covering Array für die Kombination aller Dreierkombinationen bei der Wahl von 10 binären EingabevARIABLEN A-J. Die verschiedenen Rotfärbungen zeigen die vollständigen Abdeckungen der Variablenkombinationen A-B-C, D-E-G und H-I-J [KKL<sup>+</sup>10]

Für den Sonderfall konstanter Ausprägungen der verschiedenen EingabevARIABLEN, also  $y := y_i = \text{const. } \forall i = 1, \dots, n$ , könnten Cohen et al. [CDFP97] aufzeigen, dass die Anzahl der notwendigen Tests  $T$  zur Abdeckung aller  $t$ -Kombinationen proportional zum Produkt aus  $y^t$  und der logarithmierten Anzahl der Variablen  $n$  wächst, wie Gleichung 2.3 zeigt. Insbesondere steigt die Anzahl der Testfälle bei konstanter Anzahl der Werteausprägung  $y$  und konstantem Interaktionsparameter  $t$  in Abhängigkeit der Anzahl der EingabevARIABLEN  $n$  lediglich logarithmisch an.

$$T \sim y^t \cdot \log n \quad (2.3)$$

**Konfigurationstests & Tests der Eingabewerte** Combinatorial Testing wird im Allgemeinen in zwei Teildisziplinen unterschieden, die beide auf die zuvor aufgeführten kombinatorischen Prinzipien zurückgreifen [KKL<sup>+</sup>10]. Konfigurationstests fokussieren sich auf die verschiedenen Möglichkeiten, die ein Softwaresystem in Bezug auf Konfigurationsparame-

## 2. Theoretische Grundlagen

ter einnehmen kann und welche Wechselwirkungen bei der Kombination dieser Parameter auftreten können [KKL<sup>+10</sup>]. Typischerweise sind damit laut Kuhn et al. [KKL<sup>+10</sup>] Softwaresysteme gemeint, die auf verschiedenen Betriebssystemen, verschiedenen Browsern oder unter verschiedenen Standards agieren müssen und somit eine hohe Interoperabilität erfordern. Für den Anwendungsfall dieser Arbeit ist dieser Aspekt nur in Teilen relevant.

Im Gegensatz zu den Konfigurationstests orientieren sich die Tests der Eingabewerte darauf, welche konkrete Parameter in eine spezifische Komponente oder ein Softwaresystem eingegeben werden können [KKL<sup>+10</sup>]. Dies können beispielsweise die Auswahlfelder einer Eingabemaske wie im Anwendungsfall in Kapitel 3 oder die möglichen Variablen einer Methode wie in Abbildung 2.5 betreffen.

Eine besondere Herausforderung beim Testen der Eingabewerte besteht laut Kuhn et al. [KKL<sup>+10</sup>] darin, dass Eingabeparameter meist sehr viele unterschiedliche Werte annehmen können und oftmals auch Eingaben aus kontinuierlichen Zahlenbereiche möglich sind. Da die Anzahl der Tests mit der Anzahl an möglichen Werten für jede Variable exponentiell wächst (vgl. Gleichung 2.3), würde die Berücksichtigung aller möglichen Eingaben in solchen Fällen zu einer unüberschaubaren Menge an Testfällen führen.

Um diesem Problem zu begegnen, sollten laut Kuhn et al. [KKL<sup>+10</sup>] unter anderem die Strategien der Äquivalenzklassenbildung und der Grenzwertanalyse (vgl. Abschnitt 2.1.2) als Methoden zur Reduzierung der Testfälle herangezogen werden. Insgesamt empfehlen Kuhn et al. [KKL<sup>+10</sup>] die Anzahl verschiedener Werte beziehungsweise Klassen pro Variable auf zehn zu limitieren, da diese zur  $t$ -fachen Potenz in das Wachstum der Testfälle einfließt (vgl. Gleichung 2.3). Da die Anzahl der verschiedenen Parameter  $n$  nur logarithmisch auf das Wachstum der Testfälle Einfluss nimmt (vgl. Gleichung 2.3), ist dieser Wert im Vergleich weniger kritisch zu betrachten [KKL<sup>+10</sup>].

### 2.2.2. Maße für Testabdeckung

Als einer der vier hauptsächlichen Vertreter des abdeckungsbasierten Testens (vgl. Abschnitt 2.1.1.3) erfordert das Testen der Eingabewerte wie im Falle von Combinatorial Testing Metriken, welche die Güte einer Menge von Testfällen bestimmen kann.

Im Folgenden werden die wesentlichen Metriken vorgestellt, die sich im Zusammenhang mit kombinatorischen Testmethoden etabliert haben. Die ersten beiden Metriken sind dabei

## 2. Theoretische Grundlagen

eher rein theoretischer Natur und spielen bei der praktischen Anwendung von Combinatorial Testing eine untergeordnete Rolle. Die Ausführungen werden durch folgendes Beispiel, angelehnt an Ammann und Offutt [AO08, S. 160 ff.], mit drei verschiedenen Variablen  $a, b, c$ , den möglichen Werten  $a = \{A, B\}, b = \{1, 2, 3\}, c = \{x, y\}$  und der folgenden beispielhaften Menge  $T$  an Testfällen in Tabelle 2.2 bekräftigt:

a	b	c
A	1	x
A	2	x
A	3	y
B	1	y
B	3	x
	.	.

Tabelle 2.2.: Testbestand  $T$

### 2.2.2.1. Vollständige Kombinationsabdeckung

Auch wenn vollständiges Testen in verschiedener Hinsicht meist nicht durchführbar ist (vgl. Unterabschnitte 2.1.1) lässt sich anhand der Idealvorstellung eines vollständigen Tests eine Metrik im kombinatorischen Sinne ableiten: Die vollständige Kombinationsabdeckung bezieht sich auf alle denkbaren Kombinationen der Eingabeparameter und besitzt damit die obere Schranke für Combinatorial Testing  $T_{\max}$  als Bezugsgröße [AO08, S. 160]. Im Beispiel der Variablen  $a, b, c$  wären dies alle  $2 \cdot 3 \cdot 2 = 12$  denkbaren Konstellationen. Die vollständige Kombinationsabdeckung des Testbestands  $T$  liegt demnach bei  $\frac{5}{12} \approx 42\%$ .

### 2.2.2.2. Einfache Variablenabdeckung

Die einfache Variablenabdeckung markiert in gewisser Weise die gegensätzliche Extreme zur vollständigen Abdeckung aller möglichen Kombinationen bei der vollständigen Kombinationsabdeckung: Bei der einfachen Variablenabdeckung ergibt sich die Bezugsgröße zur Bestimmung der Testfallabdeckung dadurch, dass für jede Variable isoliert betrachtet jeder mögliche Wert mindestens einmal vorkommen sollte [AO08, S. 160 f.]. Der Testbestand  $T$  erfüllt dieses Kriterium offensichtlich zu 100%, bereits die drei Testfälle  $(A, 1, x), (B, 2, y), (A, 3, x)$  würden dafür ausreichen [AO08, S. 160 f.].

## 2. Theoretische Grundlagen

### 2.2.2.3. Base Choice-Abdeckung

Beim Ansatz der Base-Choice-Abdeckung wird laut Ammann und Offutt [AO08, S. 162] für jeden Eingabeparameter eine Basiswahl festgelegt, anhand derer ein Basistestfall erstellt wird. Zusätzliche Testfälle sollten darüber hinaus durch Verändern eines einzelnen Parameters und Festhalten der Basiswerte aller anderen Parameter erzeugt werden. Im Beispiel der Variablen  $a, b, c$  könnte man beispielsweise die Kombination  $(A, 1, x)$  als Basiswahl festlegen. Dann enthält eine vollständige Base-Choice Testabdeckung die Testfälle  $(A, 1, x), (B, 1, x), (A, 2, x), (A, 3, x), (A, 1, y)$  [AO08, S. 162]. Tatsächlich vorhandene Testfälle in der Menge  $T$  sind nur die Basiswahl selbst und  $(A, 2, x)$ , sodass eine Testabdeckung von  $\frac{2}{5} = 40\%$  erreicht wird.

### 2.2.2.4. $t$ -fache Kombinationsabdeckung

Die  $t$ -fache Kombinationsabdeckung orientiert sich am Grundprinzip des Combinatorial Testing, welches in Abschnitt 2.2.1 vorgestellt wurde: Alle Kombinationen von  $t$  verschiedenen Variablen sollten idealerweise abgedeckt werden [KKL<sup>+</sup>10]. So entspricht die  $t$ -fache Kombinationsabdeckung dem Anteil der  $t$ -fachen Kombinationen, die eine Menge von Testfällen abdeckt [KKL<sup>+</sup>10]. Tabelle 2.1 zeigt eine Menge von Testfällen, die eine vollständige 3-fache Kombinationsabdeckung erfüllen.

Wird eine Menge an Testfällen, wie in Abschnitt 2.2.1 beschrieben, auf Grundlage des Interaktionsparameters  $t$  erstellt, erfüllt diese automatisch zu 100 % die  $t$ -fache Kombinationsabdeckung [KKL<sup>+</sup>10]. Für  $t = 2$  spricht man auch von der paarweisen Kombinationsabdeckung, da dort alle Werte der EingabevARIABLEN paarweise kombiniert werden [KKL<sup>+</sup>10]. Im Beispiel der Variablen  $a, b, c$  existieren für  $t = 2$  drei verschiedene 2-fach-Kombinationen  $\{(a, b), (a, c), (b, c)\}$  von denen im Testbestand  $T$  lediglich die Paarung  $(a, c)$  alle möglichen Optionen beinhaltet. Bei  $(a, b)$  fehlt die Option  $(B, 2)$  und bei  $(b, c)$  die Option  $(2, y)$ . Dies entspricht einer Abdeckung von  $\frac{1}{3} \approx 33,3\%$ .

Allgemein entspricht die Anzahl der zu prüfenden  $t$ -fachen Kombinationen  $K_t$  dem Binomialkoeffizienten aus der Anzahl der Eingabeparameter und dem Interaktionsparameter  $t$ :

$$K_t = \binom{n}{t}$$

## 2. Theoretische Grundlagen

Um festzustellen, ob eine der  $\binom{n}{t}$  Parameterkombinationen vollständig in einem Testdatensatz vorhanden ist, müssen alle Variablen-Wert-Konfigurationen für jeden Parameter geprüft werden. Beim Testbestand  $T$  sind dies genau 16 Stück.

### 2.2.2.5. $(t+k)$ -fache Kombinationsabdeckung

Jede Menge an Testfällen, die einen hohen Anteil einer  $t$ -fachen Kombinationsabdeckung aufweist, wird auch einen gewissen Anteil einer  $(t+k)$ -fachen Kombinationen abdecken [KKL<sup>+</sup>10]. Insbesondere dann, wenn verschiedene Mengen von Testfällen auf Basis einer  $t$ -fachen Kombinationsabdeckung erstellt wurden und somit in dieser Hinsicht eine Abdeckung von 100% erfüllen, wird laut Kuhn et al. [KKL<sup>+</sup>10] mittels der  $(t+k)$ -fachen Kombinationsabdeckung eine Vergleichbarkeit ermöglicht: Neben der absoluten Anzahl an Testfällen, welche zur vollständigen Abdeckung vonnöten sind, können unter anderem die  $(t+1)$ -fache oder  $(t+2)$ -fache Kombinationsabdeckung als Vergleichskriterium zwischen zwei Mengen von Testfällen herangezogen werden. Der Testbestand  $T$  besitzt wie zuvor aufgeführt eine 2-fache Kombinationsabdeckung von ungefähr 33,3 %. Prüft man nun die Abdeckung bezüglich  $(t+1) = 3$  ergibt sich eine Abdeckung von 0%, da die einzige vorhandene Dreierkonstellation  $(a, b, c)$  offensichtlich nicht vollständig vorhanden ist.

### 2.2.2.6. Variablen-Wert-Abdeckung

Bei der Betrachtung der  $t$ -fachen Kombinationsabdeckung wird für jeden Eingabeparameter  $x_i$  lediglich geprüft, ob alle  $t$ -fachen Kombinationen abgedeckt sind und dabei nicht berücksichtigt, wie viele Werte-Kombinationen zu einer vollständigen Abdeckung im Hinblick auf die jeweiligen Variablen  $x_i$  fehlen würden: Die Variablen-Wert-Abdeckung greift laut Kuhn et al. [KKL<sup>+</sup>10] diese Problematik auf und prüft für jede  $t$ -fache Variablenkombination den Anteil der abgedeckten Kombinationsmöglichkeiten der möglichen Werte. Wird beispielsweise  $t = 2$  gewählt, existieren bei binären Eingabeparametern vier verschiedene Kombinationsmöglichkeiten für jedes Paar der Parameter.

Der Testbestand  $T$  besitzt für  $t = 2$  die  $K_t = \binom{n}{t} = 3$  verschiedenen Parameterpaarungen  $(a, b), (a, c), (b, c)$ . Für das Paar  $(a, b)$  existieren aufgrund der Anzahl der möglichen Ausprägungen von  $a$  und  $b$  genau  $2 \cdot 3 = 6$  verschiedene Wertekonstellationen, dasselbe gilt für das Parameterpaar  $(b, c)$ . Die Paarung  $(a, c)$  besitzt  $2 \cdot 2 = 4$  vier mögliche Wertekonstellationen, da beide Parameter zwei Ausprägungen besitzen. Insgesamt ergibt sich

## 2. Theoretische Grundlagen

eine Anzahl von 16 verschiedenen Variablen-Wert-Konstellationen, die für eine vollständige Variablen-Wert-Abdeckung abzuprüfen sind. Die vier Kombinationen von  $(a, c)$  sind vollständig vorhanden, bei  $(a, b)$  und  $(b, c)$  fehlt jeweils eine der sechs Optionen. Dementsprechend liegt die Variablen-Wert-Konfiguration hier bei  $\frac{14}{16} \approx 87,5\%$ . Zum Vergleich: Der Wert der 2-fachen Kombinationsabdeckung liegt bei 33,3%.

### 2.2.2.7. $(p, t)$ -Vollständigkeit

Über die Variablen-Wert-Konfiguration hinaus definieren Kuhn et al. [KKL<sup>+</sup>10] die  $(p, t)$ -Vollständigkeit als ein weiterführendes Maß zur Quantifizierung der kombinatorischen Testgüte. Die  $(p, t)$ -Vollständigkeit wird dabei definiert als der Anteil der  $K_t = \binom{n}{t}$  Parameterkombinationen, deren Variablen-Wert-Konfigurationsabdeckung mindestens  $p$  überschreiten.

Im Testbestand  $T$  fehlen in Bezug auf  $t = 2$  einzig bei den Parameterpaaren  $(a, b)$  und  $(b, c)$  eine der jeweils sechs Kombinationen, sodass diese Kombinationen für  $p \leq \frac{5}{6}$  die Anforderung an die  $(p, t)$ -Vollständigkeit erfüllen. Zudem erfüllt die Kombination  $(a, c)$  aufgrund der vollständigen Variablen-Wert-Abdeckung für jedes  $p$  die  $(p, t)$ -Vollständigkeit. Wählt man also beispielsweise  $p = 75\%$ , liegt die  $(p, t)$ -Vollständigkeit des Testbestands  $T$  insgesamt bei 100%, da alle drei Parameterpaare die Variablen-Wert-Konfigurationsabdeckung von 75% überschreiten.

### 2.2.3. Algorithmen zur Testfallerzeugung

In der Vergangenheit wurden verschiedene Algorithmen entwickelt, die eine Menge an Testfällen mit kombinatorischer Abdeckung, also im Wesentlichen Covering Arrays, erstellen können. Khalsa und Labiche [KL14] konnten in einer Metaanalyse im Jahr 2014 75 verschiedene Algorithmen und Tools entdecken, die auf unterschiedliche Art und Weise kombinatorische Methoden zur Testfallerzeugung anwenden. Diese Arbeit markiert die zum aktuellen Zeitpunkt größte Übersicht über die vorhandenen Algorithmen und Tools zu Combinatorial Testing.

Laut Khalsa und Labiche [KL14] lassen sich die Algorithmen zur Erstellung von Testmengen bei Combinatorial Testing zwei grundlegenden Kategorien zuordnen, den Testbasierten und Parameter-basierten Methoden: Parameter-basierte Algorithmen berück-

## 2. Theoretische Grundlagen

sichtigen laut Khalsa und Labiche [KL14] zunächst nur  $t$  Eingabeparameter und erstellen für diesen Fall eine Testmenge mit vollständiger  $t$ -facher Abdeckung. Anschließend werden die Testfälle dann so erweitert, dass diese mit Werten der  $n - t$  fehlenden Parametern belegt werden. Falls notwendig, werden weitere Testfälle auf dem Weg zu einer vollständigen  $t$ -fachen Abdeckung ergänzt. Diese Erweiterung kann beispielsweise als Greedy-Verfahren oder anhand rekursiver, algebraischer Methoden vorgenommen werden [KL14]. Ein Beispiel hierfür ist der IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2).

Im Gegensatz dazu wird bei der Test-basierten Variante ein Testfall derart erzeugt, dass dieser in einem Schritt möglichst viele einfache  $t$ -fache Konfigurationen (vgl. Abschnitt 2.2.2) abdeckt und dabei im Vergleich zu den Parameter-basierten Algorithmen alle Eingabeparameter berücksichtigt [KL14]. Erweiterungsschritte für nicht berücksichtigte Parameter wie im Falle der Parameter-basierten Algorithmen sind hier nicht notwendig. Beispiel für ein Test-basiertes Verfahren ist der AETG-Algorithmus (vgl. Abschnitt 2.2.3.1).

Ungeachtet dieser Unterteilung konnten Khalsa und Labiche [KL14] fünf verschiedene Algorithmen-Klassen den vorhanden Combinatorial Testing Tools und Algorithmen zuordnen:

- Greedy-Verfahren: Eine optimale Lösung in Bezug auf eine Bewertungsfunktion versucht man bei Greedy-Verfahren durch Iterationsschritte lediglich auf Basis der lokal verfügbaren Informationen zu finden [Sch01, S. 185]. In Bezug auf Combinatorial Testing bedeutet dies, dass ausgehend von einer bestehenden Menge an Testfällen weitere Testfälle möglichst viele der noch nicht berücksichtigten Kombinationen abdecken [KL14] sollten. Greedy-Verfahren machen den Großteil (53 %) der von Khalsa und Labiche [KL14] entdeckten Algorithmen und Tools für Combinatorial Testing aus.
- Meta-Heuristiken: Im Allgemeinen wird bei heuristischen Algorithmen versucht, das Auffinden einer optimalen Lösung eines Optimierungsproblems durch Zuhilfenahme 'problem-spezifischer Informationen' [Sch01, S. 319], sogenannten 'Heuristiken', zu beschleunigen [Sch01, S. 319]. Im Konkreten werden im Kontext von Combinatorial Testing unter anderem die Methoden der genetischen Algorithmen, der Partikel-schwarmoptimierung oder des Simulated Annealing verwendet [KL14]. Im Vergleich zu Greedy-Methoden sind laut Khalsa und Labiche [KL14] Meta-Heuristiken meist langsamer in ihrer Ausführung, liefern aber häufig bessere Lösungen im Sinne einer kleineren Menge an benötigten Testfällen.

## 2. Theoretische Grundlagen

- Adaptive Random-/Adhoc-Verfahren: Diese Kategorie der Algorithmen-Klassen für Combinatorial Testing fokussiert sich im Grundsatz auf eine zufallsgesteuerte Erzeugung von Testfällen [KL14]. Adhoc-Verfahren erzeugen laut Khalsa und Labiche [KL14] Testfälle auf Grundlage einer zuvor angenommenen Wahrscheinlichkeitsverteilung. Bei Adaptive Random-Methoden wird durch ein Distanzmaß, beispielsweise dem Hamming-Abstand, gewährleistet, dass die erzeugten Testfälle sich nicht zu stark überschneiden und so eine geringe Menge an Testfällen zur  $t$ -fachen Abdeckung ausreicht [KL14].
- Algebraische Verfahren: Algebraische Verfahren erzeugen Testfälle anhand einer vorgegebenen mathematischen Funktion oder vordefinierten mathematischen Regeln [KL14]. Unter anderem werden dabei auch rekursive Methoden verwendet, um die Komplexität der Problematik auf kleinere Teilprobleme zu reduzieren.
- Hybride Methoden: Die Konzepte verschiedener zuvor aufgeführten Konzepte zu vereinen, steckt als Grundidee hinter den hybriden Verfahren [KL14]. Ein Beispiel hierfür ist der modifizierte IPOG-D-Algorithmus (vgl. Abschnitt 2.2.3.2).

Neben den unterschiedlichen Strategien zur Testfallerzeugung existieren zwischen den verschiedenen Algorithmen und Tools laut Khalsa und Labiche [KL14] wesentliche Unterschiede unter anderem in der maximal unterstützten Höhe des Interaktionsparameters  $t$  und der Möglichkeit Bedingungen an das Testsystem zu stellen, sodass gewisse kombinatorische Möglichkeiten ausgeschlossen werden können. Zudem besitzt nicht jeder Algorithmus die Möglichkeit sogenannte Mixed Covering Arrays abzudecken, also Testfälle, bei denen jeder Eingabeparameter eine unterschiedliche Anzahl an Werten einnehmen kann [KL14].

Im Folgenden werden beispielhaft relevante Vertreter der Algorithmen zur Erstellung von Covering Arrays vorgestellt. Diese decken nur einen Bruchteil der existierenden Methoden zur Testmengen-Erstellung dar und fokussieren sich demnach insbesondere auf diejenigen Algorithmen, die im Rahmen dieser Arbeit eine wichtige Rolle spielen.

### 2.2.3.1. AETG

Als mitunter erste Wissenschaftler überhaupt beschäftigten sich Cohen et al. [CDFP97] 1997 mit den Methoden des Combinatorial Testing und der Frage, wie auf effiziente Art und Weise Covering Arrays erstellt werden können. Der von Cohen et al. [CDFP97] entwickelte Algorithmus und das zugehörige AETG (Automatic Efficient Test Generator)

## 2. Theoretische Grundlagen

System waren über lange Zeit das de facto Standardtool in Bezug auf kombinatorische Testmethoden. Das kommerzielle AETG System ist jedoch in seiner ursprünglichen Form nicht mehr verfügbar, soll aber aufgrund seiner historischen Bedeutung kurz vorgestellt werden.

Der AETG-Algorithmus ist ein Greedy-Verfahren, das grundsätzlich eine beliebige Höhe des Interaktionsparameters  $t$  abdecken kann [CDFP97]. Die tatsächliche Realisierung des AETG-Systems wurde jedoch lediglich für eine paarweise Kombinationsabdeckung ( $t = 2$ ) entwickelt [KL14]. Darüber hinaus besitzt das AETG System die Möglichkeit unerwünschte Kombinationen durch explizite Angabe von 'verbotenen Tupeln' auszuschließen [CDFP97, KL14].

Der konkrete Algorithmus, der als Test-basierter Algorithmus einzustufen ist, soll im Folgenden kurz erläutert werden. Eine detailliertere Beschreibung der Vorgehensweise lässt sich bei Cohen et al. [CDFP97] finden.

Seien  $n$  verschiedene Eingaparameter  $x_1, \dots, x_n$ , die Anzahl der verschiedenen Ausprägungen der Eingaparameter  $y_1, \dots, y_n$  gegeben und die verschiedenen Ausprägungen des i-ten Eingabeparameters als  $z_{ij}$  mit  $j = 1, \dots, y_i$  definiert. Unter der Annahme, dass bereits  $r$  Testfälle erzeugt wurden, entsteht der  $(r + 1)$ -te Testfall durch folgendes Prinzip:

1. Zufällig wird ein Parameter  $x_{\tilde{i}}$  gewählt und für diesen Parameter die Ausprägung  $z_{\tilde{i}j}$  ermittelt, die bisher durch die wenigsten  $t$ -fachen Kombinationen abgedeckt ist. Anschließend werden die übrigen Parameter zufällig geordnet: Sei  $i_1, \dots, i_{n-1}$  die zufällige erzeugte Indexfolge von 1 bis  $n - 1$ . Dann werden die Parameter als Folge  $a_1 := x_{i_1}, a_2 := x_{i_2}, \dots, a_{n-1} := x_{i_{n-1}}$  geordnet.
2. Nun wird für jeden Parameter der erstellten Folge  $a_i$  der Wert des  $(r + 1)$ -ten Testfalls folgendermaßen bestimmt: Angenommen es seien bereits die Werte für  $a_1, \dots, a_j$  festgelegt, dann ergibt sich der Wert des Parameters  $a_{j+1}$  dadurch, dass für jeden möglichen Wert von  $a_{j+1}$  geprüft wird, wie viele neue  $t$ -fachen Überdeckungen in Kombination mit den bereits gewählten Werten von  $a_1, \dots, a_j$  entstehen würden. Davon wird schließlich das Maximum ausgewählt.

Dieses Vorgehen soll anhand des Beispiels aus Abschnitt 2.2.2 und des Testbestands  $T$  (vgl. Tabelle 2.2) mit den drei verschiedenen Variablen  $a, b, c$  ( $\hat{=} x_1, x_2, x_3$ ) und den möglichen Werten  $a = \{A, B\}$  ( $\hat{=} \{z_{11}, z_{12}\}$ ),  $b = \{1, 2, 3\}$  ( $\hat{=} \{z_{22}, z_{22}, z_{23}\}$ ),  $c = \{x, y\}$  ( $\hat{=} \{z_{31}, z_{32}\}$ ) verdeutlicht werden. Der Interaktionsparameter wird auf  $t = 2$  festgesetzt.

## 2. Theoretische Grundlagen

Angenommen es wurde bereits der erste Testfall des Testbestands  $T$  ( $A, 1, x$ ) erzeugt, dann soll nun erläutert werden wie AETG einen zweiten Testfall erstellt: Im ersten Schritt des Algorithmus wird zunächst zufällig einer der drei Parameter gewählt, beispielsweise  $b$ . Bisher wurden für diesen Parameter die Paare  $(A, 1)$  und  $(1, x)$  abgedeckt, sodass für die Ausprägung  $z_{21} = 1$  bereits zwei der insgesamt 12 verschiedenen Kombinationen der Variablen  $b$  abgedeckt wurden. Die Ausprägungen  $z_{22} = 2$  und  $z_{23} = 3$  wurden bisher noch nicht abgedeckt, eine von beiden wird somit für den nächsten Testfall als Grundlage  $z_{ij}$  gewählt. Dies soll  $z_{22} = 2$  sein. Anschließend erfolgt die zufällige Ordnung der verbleibenden Variablen in einer Folge, diese soll  $a_1 := a$  und  $a_2 := c$  entsprechen.

Im zweiten Schritt des Algorithmus wird zunächst für  $a_1 = a$  der Wert mit der maximalen, zusätzlichen Abdeckung gewählt. Da für die Variable  $a$  bisher lediglich die Paarung  $(A, 1)$  abgedeckt wird, würde sowohl die Wahl des Wertes  $A$  als des Wertes  $B$  eine zusätzliche Abdeckung der Parameter  $a$  und  $b$  bedeuten. Angenommen an dieser Stelle wird der Wert  $A$  gewählt, ergibt sich im folgenden, identischen Schritt für Parameter  $a_2 = c$  die Wahl des Wertes  $y$  zur Vervollständigung des Testfalls: Eine Ergänzung durch den Wert  $x$  würde zu der Wertepaarung  $(A, x)$  führen, die jedoch bereits über den ersten Testfall abgedeckt ist. Damit ist die Anzahl der zusätzlichen  $t$ -fachen Überdeckung bei Wahl des Wertes  $y$  höher, weshalb dieser in diesem Schritt des AETG-Algorithmus den Vorzug erhält. Als neuer Testfall ergibt sich also das Wertetupel  $(A, 2, y)$ .

### 2.2.3.2. IPOG

Der IPOG-Algorithmus (In-Parameter-Order-Generalization) stellt als Parameter-basierte Variante ein alternatives Greedy-Verfahren zum AETG-Algorithmus dar, das insbesondere im ACTS-Tool (vgl. Abschnitt 2.2.4.1) eine wesentliche Rolle spielt. IPOG wurde 2007 von Lei et al. [LKK<sup>+</sup>08] vorgestellt und stellt eine Erweiterung des allgemeinen IPO-Algorithmus [LT98] dar, der die grundlegende Idee der Parameter-basierten Erzeugung in einfacherster Form für paarweises Testen ( $t = 2$ ) umsetzt.

Sei  $n$  die Anzahl der verschiedenen Eingabeparameter  $x_i$ ,  $y_i$  die Anzahl der möglichen Ausprägungen des Parameters  $x_i$ ,  $t$  der Interaktionsparameter und  $T$  die aus dem Algorithmus resultierende Menge an Testfällen. Zudem seien die Parameter nach der Anzahl ihrer möglichen Ausprägungen geordnet, es gelte also  $y_1 \geq y_2 \geq \dots \geq y_n$ . Dann funktioniert der IPOG-Algorithmus folgendermaßen. Eine vollständige Beschreibung inklusive Pseudocode lässt sich bei Lei et al. [LKK<sup>+</sup>08] nachlesen:

## 2. Theoretische Grundlagen

1. Die Menge der ausgegebenen Testfälle  $T$  wird als leere Menge initialisiert. Dann wird zunächst für die ersten  $t$  Parameter  $x_1, \dots, x_t$  eine vollständige Kombinationsabdeckung (vgl. Abschnitt 2.2.2.1) gebildet und in die Menge  $T$  eingefügt. Falls  $n > t$  folgt nun eine iterative Erweiterung der Testfälle in  $T$  für jeden Parameter  $x_j$  mit  $t < j \leq n$ , also die Parameter, die unter den ersten  $t$  Parametern nicht dabei waren.
2. In jeder Iteration wird für den aktuellen Parameter  $x_i$  mit  $t < i \leq n$  eine Menge  $\pi$  berechnet, die alle  $t$ -fachen Kombinationen mit den bereits berücksichtigten Parametern  $j$  mit  $1 \leq j < i$  beinhaltet, die abgedeckt werden müssen. Man beachte, dass die Testmenge  $T$  bei der  $i$ -ten Iteration bereits eine vollständige  $t$ -fache Abdeckung für die Parameter  $x_1, \dots, x_{i-1}$  besitzt. Die fehlenden Testfälle zur Abdeckung von  $x_1, \dots, x_i$  werden nun in zwei Schritten vorgenommen, die Lei et al. als 'horizontales' und 'vertikales Wachstum' [LKK<sup>+</sup>08] bezeichnen.
3. Beim horizontalen Wachstum wird für jeden bereits existierenden Testfall ein zusätzlicher Wert für den Parameter  $x_i$  angefügt und dabei jener Wert ausgewählt, der in der zuvor erstellten Kombinationsmenge  $\pi$  die meisten Kombinationen abdecken kann. Neu abgedeckte Wertekombinationen werden stets aus  $\pi$  entfernt.

Bei der folgenden vertikalen Erweiterung wird für alle noch verbleibenden  $t$ -fachen Kombinationen in der Menge  $\pi$  grundsätzlich ein neuer Testfall erstellt, bei dem die Werte der aktuell gewählten Parameter der  $t$ -fach Kombination durch die jeweiligen Werte der  $t$ -fach Kombination vorgegeben werden und alle anderen Werte auf sogenannte 'Don't Cares' gesetzt werden. Dies soll gewährleisten, dass die Werte der nicht betroffenen Variablen für weitere Erweiterungen flexibel bleiben. Denn neben der Erzeugung eines Testfalls wird stets bei jeder Iteration des vertikalen Wachstums geprüft, ob eine Kombination durch Anpassungen derartiger 'Don't Cares' abgedeckt werden kann und somit kein neuer Testfall erstellt werden muss. Trifft dies zu, werden die 'Don't Cares' durch die passenden Werte der jeweiligen Parameter ersetzt und werden gewissermaßen zu 'Do Cares'.

Beispielhaft wird dieses Verfahren anhand des Beispiels aus Abschnitt 2.2.2 mit den drei verschiedenen Variablen  $a, b, c$  und den möglichen Werten  $a = \{A, B\}, b = \{1, 2, 3\}, c = \{x, y\}$  verdeutlicht werden. Der Interaktionsparameter sei auf  $t = 2$  festgesetzt: Im ersten Schritt des Algorithmus werden die Parameter der Anzahl ihrer Ausprägungen nach absteigend sortiert. Dies ergibt die Folge  $y_1 = b, y_2 = a, y_3 = b$  ( $b$  und  $a$  könnten vertauscht

## 2. Theoretische Grundlagen

werden). Anschließend wird eine Tabelle zur vollständigen Kombinationsabdeckung der ersten  $t = 2$  Parameter erstellt, die in Tabelle 2.3 auf der linken Seite dargestellt ist.

a	b
A	1
A	2
A	3
B	1
B	2
B	3

a	b	c
A	1	x
A	2	y
A	3	x
B	1	y
B	2	x
B	3	y

Tabelle 2.3.: Struktur des IPOG-Algorithmus anhand des Beispiels aus Abschnitt 2.2.2) und  $t = 2$ : Zunächst wird für  $t$  Parameter eine Tabelle mit einer vollständigen Kombinationsabdeckung erstellt (linke Seite). Anschließend wird diese Tabelle horizontal (rechte Seite) und vertikal erweitert. Im Beispiel entfällt die vertikale Erweiterung, da bereits nach der horizontalen Erweiterung alle Variablenpaare vollständig abgedeckt sind.

Im nächsten Schritt des IPOG-Algorithmus wird nun die Menge  $\pi$  ermittelt, die alle noch nicht abgedeckten Wertekombinationen beinhaltet. Im Beispiel gilt:

$$\pi = \{(A, x), (A, y), (B, x), (B, y), (1, x), (2, x), (3, x), (1, y), (2, y), (3, y)\}$$

Im Folgenden werden die Testfälle der zuvor erstellten vollständigen Kombinationsabdeckung für  $t = 2$  (vgl. linke Seite Tabelle 2.3) 'horizontal' erweitert. Für den ersten Testfall ( $A, 1$ ) ergibt sich sowohl für den Wert  $c = x$  als auch  $c = y$  eine zusätzliche Abdeckung von zwei Kombinationen. In diesem Beispiel soll an dieser Stelle der Wert  $c = x$  gewählt werden. Für den nächsten Testfall wird nun der Wert  $c = y$  gewählt, da dieser zwei Kombinationen neu abdecken würde und  $c = x$  nur einen. Dieses Vorgehen wird für alle sechs Testfälle der anfänglichen Tabelle mit vollständiger Kombinationsabdeckung wiederholt. Die rechte Seite der Tabelle 2.3 zeigt das Resultat. Im Beispiel sind damit alle drei Paarungen der drei Variablen  $a, b, c$  vollständig abgedeckt, sodass der vertikale Erweiterungsschritt des IPOG-Algorithmus entfällt.

Der IPOG-Algorithmus zählt während seiner Ausführung alle denkbaren  $t$ -fachen Kombinationen auf und besitzt laut Lei et al. [LKK<sup>+</sup>08] die Zeitkomplexität von  $O(y_1^{t+1} \cdot n^{t-1} \cdot \log n)$ , wobei  $n, t$  und  $y_1$  den zuvor eingeführten Definitionen entsprechen. Dement-

## 2. Theoretische Grundlagen

sprechend ist der allgemeine IPOG-Algorithmus insbesondere bei sehr großen Systemen ineffektiv [LKK<sup>+</sup>08], was zu verschiedenen Erweiterungen des Algorithmus führte.

Der IPOG-D-Algorithmus [LKK<sup>+</sup>08] vereint die Methoden des allgemeinen IPOG-Algorithmus und eines algebraisch-rekursiven Ansatzes, um bei der Erstellung der Testfälle nicht alle Kombinationsmöglichkeiten explizit aufzählen zu müssen. Im Konkreten wird dabei der erste Schritt des zuvor erläuterten IPOG-Algorithmus, also die initiale Erstellung einer  $t$ -fachen Abdeckung für die ersten  $t$  Parameter, mittels einer rekursiven Methode optimiert. Ein Verfahren von Chateauneuf [CCK99] für die effiziente Erstellung von Covering Arrays mit Interaktionsparameter  $t = 3$  dient dabei als Grundlage. Insbesondere bei einer großen Anzahl an verschiedenen Werten für die Parameter  $x_i$  agiert der IPOG-D-Algorithmus laut Lei et al. [LKK<sup>+</sup>08] wesentlich schneller als der IPOG-Algorithmus: Anhand des Beispiels von  $n = 20$  unterschiedlichen Eingabeparameter mit je vier verschiedenen Werten und dem Interaktionsparameter  $t = 5$  konnten Lei et al. aufzeigen, dass der IPOG-D-Algorithmus lediglich 3% der benötigten Zeit zur Ausführung im Vergleich zum IPOG-Algorithmus benötigt. Zugleich fiel die resultierende Menge an Testfällen bei IPOG-D zur vollständigen fünffachen Abdeckung um 51 % größer aus.

Ein grundlegend anderer Ansatz zur Zeit- und Testfalloptimierung wird bei den IPOG-F- und IPOG-F2-Algorithmen [FLL<sup>+</sup>08] gewählt, die sich neben einigen kleineren Optimierungen des IPOG-Algorithmus auf die effiziente horizontale Erweiterung im IPOG-Algorithmus fokussieren. Mittels der Methoden des dynamischen Programmierens wird laut Forbes et al. [FLL<sup>+</sup>08] die bestmögliche Abdeckung wesentlich schneller gefunden als beim ursprünglichen IPOG-Algorithmus. Während der allgemeine IPOG-Algorithmus alle  $\binom{n-1}{i-1}$  Kombinationsoptionen für den neu hinzugefügten Parameter  $x_i$  überprüft, wird der horizontale Erweiterungsschritt bei den beiden IPOG-F-Algorithmen anhand von Werten aus zwei Tabellen ermittelt, die Daten der zuvor gespeicherten Kombinationen beinhalten. Zudem können so mehr Informationen bei der bestmöglichen Wahl des 'Erweiterungswertes' berücksichtigt werden, was sich letztlich in einer kleineren Menge an Testfällen zur vollständigen  $t$ -fachen Abdeckung bemerkbar macht [FLL<sup>+</sup>08].

IPOG-F und IPOG-F2 unterscheiden sich durch die Verwendung einer Heuristik bei der Auswahl der horizontalen Erweiterung innerhalb der Tabellen des Ansatzes der dynamischen Programmierung bei IPOG-F2: IPOG-F2 ist demnach wesentlich effizienter in der Rechenleistung und benötigt weniger Speicherplatz, erstellt aber größere Mengen an Testfällen im Vergleich zu IPOG-F [FLL<sup>+</sup>08].

## 2. Theoretische Grundlagen

Laut Forbes et al. [FLL<sup>+</sup>08] ergibt sich für beide IPOG-F-Varianten eine ähnliche Worst-Case-Zeitkomplexität wie beim allgemeinen IPOG-Algorithmus. Durch experimentelle Untersuchungen konnten die Autoren jedoch aufzeigen, dass sowohl IPOG-F als auch IPOG-F2 erhebliche Ersparnisse in der Durchführungszeit im Vergleich zu IPOG einbringen. IPOG-F spart zusätzlich rund 5% der notwendigen Testfälle gegenüber IPOG ein [FLL<sup>+</sup>08].

### 2.2.3.3. CASA

CASA [GCD09, GCD11] steht für 'Covering Arrays by Simulated Annealing' und stellt ein meta-heuristisches Verfahren zur Erstellung von Covering Arrays dar. CASA basiert auf den Grundsätzen des Simulated Annealing, was erstmals von Stevens [Ste99] im Zusammenhang von Covering Arrays erwähnt wurde. Garvin et al. [GCD09] erweiterten das Grundkonzept von Stevens derart, dass Bedingungen des zu testenden Systems formuliert werden konnten und diese bei der Testfallerzeugung berücksichtigt wurden. 2011 ergänzten sie ihr erweitertes Konzept durch weitere Verbesserungen des verwendeten Algorithmus im Hinblick auf die Menge an erzeugten Testfällen, die möglichst gering ausfallen sollte [GCD11]. CASA wurde von der University of Nebraska-Lincoln verwaltet, konnte jedoch nach den Recherchen im Rahmen dieser Arbeit nicht mehr als anwendbares Tool aufgefunden werden. Jedoch ist der Algorithmus in das web-basierte CTWedge-Tool (vgl. ??) eingebettet.

Im Allgemeinen fußt Simulated Annealing auf dem Prinzip der lokalen Verbesserungsstrategien [Sch01, S. 330]: Ausgehend von einem zufälligen Startzustand wird die unmittelbare Nachbarschaft jenes Zustands untersucht und dabei lokale Verbesserungsschritte zum optimalen Ergebnis mithilfe einer Kostenfunktion vorgenommen. Dieses Prinzip, das auch unter dem Konzept des Hill Climbing bekannt ist [Sch01, S.327], verharrt jedoch bei lokalen Optima, sodass nicht unbedingt die bestmögliche globale Lösung gefunden wird [Sch01, S.327]. Simulated Annealing löst dieses Problem, indem mit einer zeitlich abnehmender Wahrscheinlichkeit auch Lösungen akzeptiert werden, die schlechter als zuvor ermittelte Teillösungen sind. Eine detailliertere Beschreibung dieses Grundprinzips lässt sich in Schöning finden [Sch01, S.329 ff.].

Der von Garvin et al. [GCD11] entwickelte CASA-Algorithmus, der auf das Konzept des Simulated Annealing zurückgreift, besitzt zwei grundlegende Komponenten, welche die Autoren als 'Outer Search' und 'Inner Search' bezeichnen:

## 2. Theoretische Grundlagen

1. Die 'Outer Search' bildet den Rahmen des Algorithmus, der über einen möglichen Bereich eine Binärsuche für die optimale Anzahl an Testfällen  $T$  durchführt. Im Vorfeld wird daher eine untere Schranke  $T_{\min}^*$  und  $T_{\max}^*$  vorgegeben, die diesen Rahmen bilden.  $T_{\min}^*$  und  $T_{\max}^*$  liegen dabei zwischen der minimalen und maximalen Anzahl an notwendigen Testfällen zur  $t$ -fachen Kombinationsabdeckung  $T_{\min}$  und  $T_{\max}$  (vgl. Abschnitt 2.2.1). Für jeden möglichen Kandidaten  $T^*$  zwischen  $T_{\min}^*$  und  $T_{\max}^*$  wird dann in der 'Inner Search' versucht eine Testmenge zu bilden. Falls eine Testmenge  $T$  mit vollständiger Abdeckung gefunden wurde, wird die obere Grenze innerhalb der Binärsuche angepasst und versucht eine Testabdeckung mit geringerer Anzahl an Testfällen zu erreichen.
2. In der 'Inner Search' findet der Prozess des Simulated Annealing statt: Die Kostenfunktion, die es zu optimieren gilt, ist die Anzahl an nicht abgedeckten  $t$ -fachen Kombinationen, die idealerweise 0 betragen soll. Auf Basis einer Startkonfiguration, die anhand vorheriger Iterationen der 'Outer Search' abgeleitet wird, werden im Folgen Nachbarschaftsveränderungen in Form von Anpassungen einzelner Werte eines Parameters vorgenommen. Dabei wird stets geprüft, inwiefern sich die Anzahl der abgedeckten Kombinationen verändert und unter dem Prinzip des Simulated Annealing mit einem 'Cooldown' der Akzeptanz-Wahrscheinlichkeit für schlechtere Lösungen Anpassungen durchgeführt.

Ergänzende Erweiterungen dieses grundlegenden Algorithmus nahmen Garvin et al. [GCD11] insbesondere im Hinblick auf die effiziente Einbettung von Bedingungen an das Testsystem vor, zudem verändert der optimierte CASA-Algorithmus in der 'Inner Search' nicht nur einzelne Werte eines Parameters in einem Updateschritt, sondern ganze  $t$ -fache Mengen. Dies bewirkt, dass die Iterationsschritte größer ausfallen und so vor allem nahe der Optimalitätsgrenze von keinen fehlenden Kombinationen weniger Schritte benötigt werden, um das Optimum zu finden [GCD11]. Außerdem beobachteten Garvin et al., dass bei einer fehlgeschlagenen 'Inner Search' – also kein passendes Covering Array mit  $N$  Testfällen konnte erstellt werden – eine reine Binärsuche in der 'Outer Search' frühzeitig geringe Werte für  $N$  als Optimallösung ausschließt. Als Lösung dafür setzt der modifizierte CASA-Algorithmus auf eine einseitige Einschränkung als Alternative zur Binärsuche [GCD11].

In einer experimentellen Untersuchung konnten Garvin et al. [GCD11] aufzeigen, dass der CASA-Algorithmus im Vergleich zu einer modifizierten Version des AETG-Algorithmus

## 2. Theoretische Grundlagen

(vgl. Abschnitt 2.2.3.1) durchschnittlich 25 % weniger Konfigurationen zur vollständigen Testabdeckung erzeugt. Zudem konnten die Autoren ermitteln, dass insbesondere bei großen Systemen mit längerer Durchführungszeit der Algorithmen CASA schneller Ergebnisse liefert als der vergleichene, modifizierte AETG-Algorithmus. Ein Vergleich zwischen CASA und den Versionen des IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) existiert in dieser Form nicht.

### 2.2.4. Tools zur Testfallerzeugung

Die Landschaft der verfügbaren Tools und Algorithmen zur Anwendung von Combinatorial Testing ist trotz einer derart ausführlichen Übersicht, wie Khalsa und Labiche [KL14] ausarbeiteten, recht unübersichtlich. Viele der insgesamt 75 Algorithmen und Tools, die Khalsa und Labicher finden konnten, sind lediglich in Form von Pseudocode zugänglich und können daher nicht unmittelbar auf einen konkreten Anwendungsfall umgemünzt werden. Des Weiteren gibt es einige kommerzielle Anwendungen, die im Rahmen dieser Arbeit nicht berücksichtigt werden. Eine weitere Einschränkung der verfügbaren Tools für den in Kapitel 3 vorgestellten Anwendungsfall ergibt sich dadurch, dass viele Tools lediglich paarweise Abdeckungen ( $t = 2$ ) ermöglichen [KL14]. Eine Übersicht über viele der aktuell nutzbaren Tools lässt sich bei Czerwonka [Czeb] finden.

Ungeachtet dessen sind die zur Verfügung stehenden Informationen einiger dort aufgeführter Tools limitiert, sodass die verwendeten Algorithmen nur teilweise bekannt sind. Unter anderem gehört zu dieser Kategorie das Tool Allpairs [Bac12], bei dem lediglich der Quellcode bekannt ist und keine weiteren Informationen vorliegen. Allpairs lässt sich in die Algorithmen der Adhoc-Kategorie einordnen und ist nach Aussagen des Autors Bach [Bac12] wesentlich ineffizienter als andere Algorithmen.

Die wenigen Tools mit einer fundierten wissenschaftlichen Grundlage, die frei zu Verfügung stehen, einen Interaktionsparameter größer als  $t = 2$  berücksichtigen können und zudem zum aktuellen Zeitpunkt abgerufen werden können, sind CASA, ACTS (vgl. Abschnitt 2.2.4.1) und PICT (vgl. Abschnitt 2.2.4.2). CASA entspricht einem Kommandozeilenbasierten Tool, das den zugehörigen CASA-Algorithmus wie in Abschnitt 2.2.3.3 beschrieben ausführt. Dementsprechend werden an dieser Stelle lediglich die Tools ACTS und PICT ausführlich erläutert.

## 2. Theoretische Grundlagen

Abbildung 2.6 zeigt eine Übersicht über die verwendeten Tools und Algorithmen im Anwendungsfall (vgl. Kapitel 3) dieser Arbeit. Die farbliche Markierung ordnet die Algorithmen der Klassifizierung bei der Erstellung von Testmengen zur  $t$ -fachen Kombinationsabdeckung zu. Algorithmen in kursiver Schrift werden im Rahmen dieser Arbeit nicht berücksichtigt.

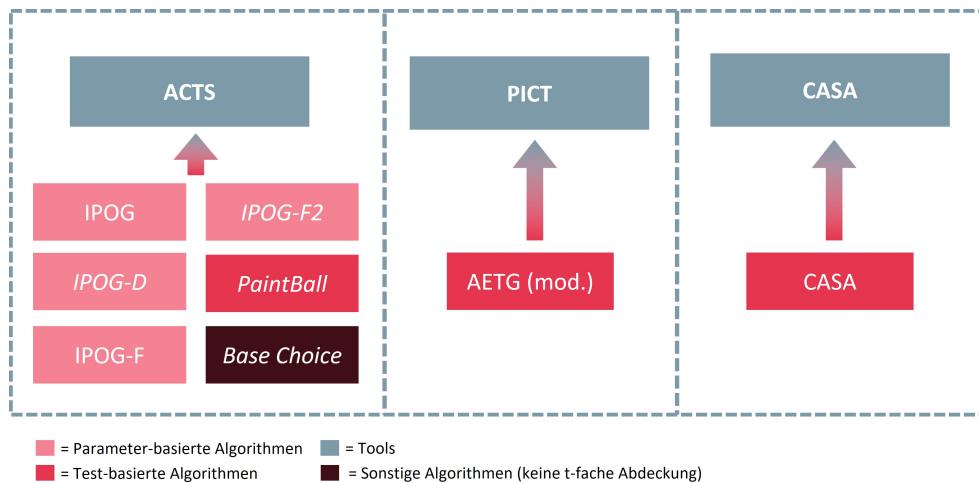


Abbildung 2.6.: Übersicht über die verschiedenen Tools und Algorithmen zur Erstellung von Testmengen nach dem Prinzip des Combinatorial Testing.: Die farbliche Markierung repräsentiert die Klassifizierung der Algorithmen in Bezug auf das Vorgehen bei der Erstellung der Testmengen. Kursiv gedruckte Algorithmen werden im Rahmen dieser Arbeit nicht berücksichtigt.

### 2.2.4.1. ACTS

Das 'Advanced Combinatorial Testing System' (ACTS) [YLKK13] ist eine vom US National Institute of Standards and Technology und der Universität von Texas entwickelte Software zur Erzeugung von Testmengen nach den Prinzipien des kombinatorischen Testens. ACTS wurde erstmalig 2006 vorgestellt, damals noch unter dem Namen FireEye [LKK<sup>+</sup>07]. Das Tool ermöglicht die Erstellung von Testmengen zur  $t$ -fachen Kombinationsabdeckung über eine grafische Benutzeroberfläche, eine Kommandozeilen-Schnittstelle oder eine Java-Programmierschnittstelle (API) [YLKK13]. Abbildung 2.7 zeigt beispielhaft die grafische Oberfläche von ACTS nach Erstellung eines Covering Arrays. ACTS selbst wurde in Java geschrieben.

ACTS ist im Gegensatz zu vielen anderen Tools zur Erstellung von Covering Arrays sehr flexibel in der Art und Weise, wie das zu testende System auszusehen hat und auf welche

## 2. Theoretische Grundlagen

	Tarif	Zuzahlung_zu_Beginn	Beitrag_40b	Beitrag_363	Durchfuehrungsweg	Art_der_Rueckdeckung	Beitrag
1	AREven	0	0	0	Direktzusage	keine	0
2	AREven	0	0	0	Direktzusage	Rueckdeckungsversicherung	0
3	AREven	0	0	2068	Direktversicherung	keine	0
4	AREven	0	0	4068	Direktversicherung	keine	0
5	AREven	0	2148	0	Direktversicherung	keine	0
6	AREven	0	2148	2068	Direktversicherung	keine	0
7	AREven	0	2148	4068	Direktversicherung	keine	0
8	AREven	0	2149	0	Direktversicherung	keine	0
9	AREven	0	2149	2068	Direktversicherung	keine	0
10	AREven	0	2149	4068	Direktversicherung	keine	0
11	AREven	2000	0	0	Direktversicherung	keine	0
12	AREven	2000	0	0	Direktzusage	Rueckdeckungsversicherung	0
13	AREven	2000	0	2068	Direktversicherung	keine	0
14	AREven	2000	0	4068	Direktversicherung	keine	0
15	AREven	2000	2148	0	Direktversicherung	keine	0
16	AREven	2000	2148	2068	Direktversicherung	keine	0
17	AREven	2000	2148	4068	Direktversicherung	keine	0
18	AREven	2000	2149	0	Direktversicherung	keine	0
19	AREven	2000	2149	2068	Direktversicherung	keine	0
20	AREven	2000	2149	4068	Direktversicherung	keine	0
21	AREven	99	0	0	Direktversicherung	keine	0
22	AREven	99	0	2068	Direktversicherung	keine	0
23	AREven	99	0	4068	Direktversicherung	keine	0
24	AREven	99	2148	0	Direktversicherung	keine	0
25	AREven	99	2148	2068	Direktversicherung	keine	0
26	AREven	99	2148	4068	Direktversicherung	keine	0
27	AREven	99	2149	0	Direktversicherung	keine	0
28	AREven	99	2149	2068	Direktversicherung	keine	0
29	AREven	99	2149	4068	Direktversicherung	keine	0
30	AREven	1000001	0	0	Direktzusage	Rueckdeckungsversicherung	0
31	AREven	n	n	n	Direktzusage	Rueckdeckungsversicherung	n

Abbildung 2.7.: Screenshot der grafischen Benutzeroberfläche von ACTS nach Erstellung eines Covering Arrays. Screenshot erstellt am 15.12.2021.

Art und Weise Testfälle generiert werden sollen: Grundsätzlich kann ACTS Mixed Covering Arrays erstellen: Dies bedeutet, dass die Eingabeparameter eine unterschiedliche Anzahl an verschiedenen Werten annehmen können. Zudem unterstützt ACTS Bedingungen an das Testsystem: Mittels einer speziellen Syntax können bestimmte Parameterbeziehungen durch aussagenlogischen Ausdrücke ausgeschlossen werden. Diese Syntax orientiert sich im Wesentlichen an typischen logischen Operationen 'Und', 'Oder', 'Nicht' und arithmetischen Vergleichsoperatoren '<', '>', ' $\leq$ ', ' $\geq$ ' (vgl. dazu [YLKK13]). Für die Umsetzung dieses Konzepts ist in ACTS ein externes Framework namens Choco [sol] integriert, das für jeden erzeugten Testfall überprüft, ob die zuvor formulierten Bedingungen verletzt werden.

Unter Berücksichtigung dieser Bedingungen erstellt ACTS Covering Arrays bis zu einer Abdeckung des Interaktionsparameters  $t = 6$ . Dass ACTS keine Abdeckung höherer Werte für den Interaktionsparameter ermöglicht, liegt darin begründet, dass Kuhn et al. [KWG04] in einer experimentellen Untersuchung aufzeigen konnten, dass fast alle Fehler von Softwaresystemen auf die Interaktion von sechs oder weniger Parameter zurückzuführen sind (vgl. Abschnitt 2.2).

ACTS ermöglicht es zudem für verschiedene Gruppen von Eingabeparameter einen unterschiedlichen Wert für den Interaktionsparameter zu fordern und ermöglicht es somit verschiedene Parameterbeziehungen zu priorisieren. In der Literatur wird dies 'Mixed Strength Test Generation' genannt [YLKK13, Cze06]. Wird beispielsweise der Interaktionsparameter  $t = 2$  gewählt und es ist jedoch bekannt, dass die Parameter  $x_1, x_3, x_4$  und

## 2. Theoretische Grundlagen

$x_7$  in ihrer Interaktion besonders kritisch zu betrachten sind, so kann für diese Parametermenge bei ACTS eine 'Relation' [YLKK13] definiert werden, bei der eine Abdeckung mit  $t = 3$  gefordert wird. ACTS besitzt eine interne Logik, die zunächst die verschiedenen Relationen verknüpft und eventuell überflüssige Kombinationen herausfiltert. Erst danach erstellt ACTS eine Übersicht über die zu erstellenden Kombinationen und bildet ein Covering Array [YLKK13].

Eine weitere Eigenschaft, welche ACTS abdeckt, ist das explizite Testen auf fehlerhafte Eingaben, sogenanntes 'Negative Testing' [DL19]. ACTS bietet die Möglichkeit neben der standardmäßigen Definition der Werte eines Parameters auch explizit Werte zu definieren, die erwartungsgemäß zu Fehler führen sollen. Diese werden dann bei der Erstellung der Menge an Testfällen gesondert berücksichtigt: Konkret bedeutet dies, dass ACTS sicherstellt, dass pro Testfall maximal ein fehlerhafter Wert vorkommen kann. Dies hat das Ziel mögliche Überlagerungen von fehlerhaften Werten zu vermeiden [DL19]. Zudem erstellt ACTS Testfälle derart, dass jeder fehlerhafte Wert mit jeder  $(t - 1)$ -fachen Kombination von gültigen Werten kombiniert wird [DL19].

Bei der Erstellung von Covering Arrays haben die Anwender\*innen von ACTS die Wahl zwischen verschiedenen Algorithmen: ACTS wurde konzeptionell auf die Nutzung der unterschiedlichen IPOG-Algorithmen ausgerichtet (vgl. Abschnitt 2.2.3.2) und besitzt dementsprechend die Auswahlmöglichkeiten des klassischen IPOG-Algorithmus [LKK<sup>+</sup>08], des IPOG-D-Algorithmus [LKK<sup>+</sup>08] und der beiden IPOG-F-Algorithmen [FLL<sup>+</sup>08] (vgl. Abschnitt 2.2.3.2). Darüber hinaus implementierten die Entwickler von ACTS einen zufallsbasierten Algorithmus namens 'PaintBall'. Zudem existiert bei ACTS die Möglichkeit ein Covering Array im Sinne einer Base Choice-Abdeckung zu erstellen, wie sie in Abschnitt 2.2.2 vorgestellt wurde. Beschränkungen bei der Auswahl der Algorithmen ergeben sich jedoch bei der Verwendung der Prinzipien von Mixed Covering Arrays und Constraints: Mixed Covering Arrays und Constraints können nur unter Anwendung des allgemeinen IPOG-Algorithmus und der IPOG-F-Variante erstellt werden. [DL19].

Neben der Erzeugung einer Menge von Testfällen zur vollständigen  $t$ -fachen Abdeckung besteht bei ACTS die Möglichkeit ein bestehendes Covering Array bei Anpassungen der Parameter und deren Werten zu ergänzen ohne dabei eine komplett neue Testfallmenge erstellen zu müssen. Allgemein bietet ACTS für jede Menge an Testfällen zudem die Möglichkeit zu prüfen, ob und inwiefern eine  $t$ -fache Abdeckung erreicht wird.

## 2. Theoretische Grundlagen

### 2.2.4.2. PICT

PICT [Cze06, Czea] wurde 2006 erstmalig von Jacek Czerwonka vorgestellt und basiert auf dem AETG-Algorithmus (vgl. Abschnitt 2.2.3.1). Das Tool wird über die Kommandozeile angesteuert, nimmt dabei eine Textdatei mit der Spezifikation des Testsystems als Eingabe an und gibt die Menge der erzeugten Testfälle auf der Kommandozeile aus [Czea]. Czerwonka [Cze06] betont, dass sich das entwickelte Tool verstärkt auf die (Rechen-)Effizienz und die Nutzbarkeit im praktischen 'Test-Alltag' ausrichtet und dementsprechend keine grundlegend neuen Methoden zur Generierung von Covering Arrays beinhaltet. Konkret bedeutet dies: PICT besitzt eine Reihe zusätzlicher Eigenschaften und Funktionalitäten im Vergleich zur allgemeinen Erstellung von Covering Arrays wie sie beispielsweise im Rahmen vom AETG-Algorithmus durchgeführt werden.

So veränderte Czerwonka den Algorithmus der Erstellung der Covering Arrays des AETG-Ansatzes derart, dass PICT auf Basis eines fest definierten Seeds, also einem vordefinierten Startwert für die Erstellung pseudozufälliger Zahlen, arbeitet und keine zufällige Komponente besitzt (vgl. im Gegensatz dazu Abschnitt 2.2.3.1).

Neben dieser Änderung am Algorithmus selbst fügte Czerwonka verschiedene Optionen ein, welche die Menge der zu erzeugenden Kombinationen, die als Grundlage des Algorithmus dient, flexibel hält [Cze06]. Konkret bedeutet dies: Der Interaktionsparameter  $t$  kann bei PICT bis zu einem Wert von  $t = 6$  beliebig gewählt werden [KL14], zudem unterstützt das Tool das Prinzip der 'Mixed Test Strength Generation' (vgl. Abschnitt 2.2.4.1) [Cze06]. PICT erlaubt es den Anwendenden darüber hinaus eine Hierarchie zu definieren, sodass tiefer positionierte Komponenten nur auf ihrer jeweiligen Hierarchiestufe kombiniert werden können und somit die Anzahl der Testfälle im Gesamten reduziert werden kann [Cze06].

Wie auch ACTS ist PICT in der Lage, Bedingungen des zu testenden Systems zu berücksichtigen: Mittels einer 'IF-THEN'-Logik kann spezifiziert werden, welche Voraussetzungen die resultierenden Testfälle erfüllen sollen. PICT wandelt diese aussagenlogischen Formeln intern in eine Menge an auszuschließenden Kombinationen um, die dann beim modifizierten AETG-Erzeugungsalgorithmus exkludiert werden [Cze06]. Eine weitere Ergänzung von PICT ist die Formulierung von 'Seeds': Dies sind Testfälle, die in jedem Fall im Covering Array erscheinen müssen. Zudem unterstützt PICT explizit sogenanntes 'Negative Testing', also gezieltes Testen auf Fehlverhalten, und die Gewichtung von Werten einzelner Parameter, die bei 'Don't Cares' im Algorithmus berücksichtigt wird.

# **3. Anwendungsfall**

Im folgenden Kapitel wird der Anwendungsfall vorgestellt, der als Grundlage zur Beantwortung der Fragestellungen dieser Arbeit (vgl. Abschnitt 1.2) dient. Der Anwendungsfall befasst sich mit der Untersuchung der Webanwendung Easy Web Leben [ALHb] (<https://www.al-h.de/appserver/easyweb/>), im Folgenden Easy Web genannt. Die Plattform dient zur Erstellung von Angeboten im Bereich von Lebensversicherungsprodukten. Die Struktur dieses Kapitels sieht wie folgt aus: Zunächst werden einige relevante versicherungstechnische Begriffe eingeführt, anhand derer sich eine detaillierte Beschreibung der Easy Web-Plattform anschließt. Daraufhin wird die konkrete Methodik und Implementierung zur Beantwortung der Fragestellungen erläutert, zudem die Ergebnisse der korrespondierenden Untersuchungen vorgestellt.

## **3.1. Vorstellung des Testobjekts**

Das 'Beratungscockpit' Easy Web [ALHb] soll Vermittelnde und der Kundschaft der ALH-Gruppe bei der Ermittlung der individuellen Bedürfnisse im Bereich der Lebensversicherung unterstützen. Dafür steht eine Reihe verschiedener Produkte und Tarife zur Verfügung, die anhand unterschiedlicher Parameter an die individuellen Anforderungen der Kundschaft angepasst werden können und letztlich in einem konkreten Antrag zum Abschluss einer Versicherung münden können. Easy Web ermittelt parallel dazu anhand der eingegebenen Parameter verschiedene Komponenten des resultierenden Versicherungstarifs wie den Beitragsverlauf, Kapitalzahlungen, Todesfallleistungen, zugehörige Rentenleistung, etc.

Easy Web unterscheidet grundlegend zwischen Produkten der privaten Vorsorge und der betrieblichen Altersvorsorge (bAV). Der Anwendungsfall dieser Arbeit fokussiert sich auf den Bereich der bAV.

### 3. Anwendungsfall

#### 3.1.1. Versicherungstechnische Grundlagen

Die betriebliche Altersvorsorge (bAV) hat im deutschen Rentensystem gemeinsam mit der gesetzlichen und der privaten Vorsorge das Ziel, zu 'einer angemessenen Gesamtversorgung beim Ausscheiden aus dem Arbeitsleben' [BK17, S. 12] jeder Person beizutragen. Die gesetzliche, private und betriebliche Altersvorsorge werden in diesem Zusammenhang auch häufig als die drei Säulen des deutschen Rentensystems bezeichnet [LP16, S. 3].

Der Begriff der bAV beinhaltet 'alle Leistungen, die einem Arbeitnehmer zur Altersvorsorge, Hinterbliebenenversorgung oder Invaliditätsversorgung von seinem Arbeitgeber aus Anlass des Arbeitsverhältnisses zugesagt worden sind' [BK17, S. 1]. Klassischerweise ermöglicht die bAV lebenslängliches Einkommen in der Rente, sichert also ein Langlebigkeitsrisiko ab. Es können aber auch biometrische Risiken abgesichert werden, wie beispielsweise bei Berufsunfähigkeits-, Erwerbsunfähigkeits- und Risikolebensversicherungen.

Ursprünglich wurde die bAV als 'zusätzliche Sozialleistung des Arbeitgebers' [BK17, S. 7] eingeführt, sodass die Finanzierung der bAV aus wirtschaftlicher Sicht einzig über separate Beiträge des Arbeitgebers erfolgte (Arbeitgeber-finanziert). Seit dem Jahr 2002 besitzt jeder Arbeitnehmer das Recht auf Umwandlung seines Lohnes für die Nutzung einer bAV [BK17, S. 23], zudem muss dieses Vorhaben vom Arbeitgeber nach § 1 a BetrAVG mit einem Zuschuss unterstützt werden [BK17, S. 29], falls dieser Sozialversicherungsbeiträge durch die Entgeldumwandlung einspart. Steuert der Arbeitgeber nur den gesetzlich verpflichtenden Teil zur Entgeldumwandlung des Arbeitnehmers bei, spricht man von einer Arbeitnehmer-finanzierten bAV. Leistet der Arbeitgeber im Rahmen einer Entgeldumwandlung des Arbeitnehmers Beiträge zur bAV über den verpflichtenden Teil hinaus, so erfolgt die Finanzierung der bAV als sogenannte Misch-Finanzierung.

Für die konkrete Ausgestaltung einer bAV stehen verschiedene Durchführungswege zur Verfügung, die im Folgenden basierend auf Buttler und Keller [BK17] vorgestellt werden:

**Direktzusage** Bei der Direktzusage übernimmt der Arbeitgeber die unmittelbare Verantwortung für die mit dem Arbeitnehmer getroffenen Vereinbarungen über die Erbringung einer Vorsorgeleistung. Der Arbeitnehmer besitzt einen Rechtsanspruch auf die versprochene Leistung gegenüber dem Arbeitgeber. Diesen weist der Arbeitgeber in seiner Bilanz als Pensionsrückstellungen aus. Zusätzlich kann der Arbeitgeber die versprochenen Leistungen über eine Rückdeckungsversicherung absichern. Finanziert wird die Direktzusage

### 3. Anwendungsfall

üblicherweise durch den Arbeitgeber (Arbeitgeber-finanziert), eine Entgeldwandelung ist jedoch auch möglich (Arbeitnehmer-finanziert) [BK17, S. 140].

**Direktversicherung** Die Direktversicherung beschreibt das Vorgehen, bei welchem der Arbeitgeber eine Lebensversicherung auf das Leben des Arbeitnehmers bei einer Lebensversicherungsgesellschaft abschließt und die Beiträge übermittelt. Versicherungsnehmer ist somit der Arbeitgeber, bezugsberechtigt für die abgeschlossene Versicherungsleistung sind jedoch der Arbeitnehmer beziehungsweise dessen Hinterbliebene. Finanziert wird die Direktversicherung allein vom Arbeitnehmer als Abzug vom Bruttolohn (Arbeitnehmer-finanziert), allein vom Arbeitgeber als separate Arbeitgeberleistung (Arbeitgeber-finanziert) oder als Mischform (Misch-finanziert) beider zuvor aufgeführten Varianten [LP16, S. 33].

**Unterstützungskasse** Die Unterstützungskasse ist eine von einem oder mehreren Trägerunternehmen etablierte Versorgungseinrichtung, die rechtlich selbstständig arbeitet und die Vorsorgeleistungen der bAV für die Arbeitnehmer der beteiligten Unternehmen übernimmt. Die Unterstützungskasse fungiert dabei nur als Dienstleister für die betroffenen Unternehmen, rechtlich verbleibt der Anspruch der Versicherungsleistung des Arbeitnehmers beim Arbeitgeber. Unterstützungskassen werden entweder durch die Beiträge zur Entgeldumwandlung der Arbeitnehmer (Arbeitnehmer-finanziert) oder über direkte Zuwendungen der Arbeitgeber (Arbeitgeber-finanziert) finanziert.

**Pensionskasse** Pensionskassen sind Versicherungsunternehmen, die sich im Gegensatz zu Lebensversicherungsunternehmen ausschließlich um die Durchführung von Rentenversicherungen im Rahmen von bAV-Verträgen kümmern. Konzeptionell gleicht die Durchführung einer bAV via Pensionskasse in weiten Teilen der Direktversicherung, Beiträge werden wie bei der Direktversicherung vom Arbeitgeber übermittelt. Die Finanzierung erfolgt wie bei der Direktversicherung über den Arbeitnehmer (Arbeitnehmer-finanziert) oder Arbeitgeber (Arbeitgeber-finanziert) allein oder gemeinsam (Misch-finanziert).

Bis 2005 wurden Pensionskassen im Vergleich zu allgemeinen Lebensversicherern steuerlich besser gestellt, aktuell unterscheiden sich die beiden Durchführungswege nur marginal: So erhalten Arbeitnehmer ihre Versicherungsleistung im Falle der Pensionskasse erst bei Beendigung ihrer Erwerbstätigkeit und nicht zu einem vordefinierten Zeitpunkt wie bei

### 3. Anwendungsfall

einer Direktversicherung [BK17, S. 191]. Zudem entfallen bei der Pensionskasse bei einem Unternehmensaustritt mit einem Pensionskassenvertrag steuerliche Vorteile für den Arbeitnehmer, falls dieser weiterhin privat Beiträge entrichtet [BK17, S. 192].

**Pensionsfonds** Der Pensionsfonds existiert seit 2002 als zusätzlicher Durchführungsweg der bAV. Wie bei der Direktversicherung und der Pensionskasse auch agiert ein Pensionsfonds als rechtlich selbständiger Versorgungsträger, der Arbeitnehmern bei Abschluss eines bAV-Vertrags über den Arbeitgeber einen Rechtsanspruch auf Versicherungsleistungen einräumt. Die bAV-Durchführung via Pensionsfonds ermöglicht im Vergleich zur Direktversicherung und Pensionskasse größere Freiheiten in Bezug auf den Rechnungsszins, dieser wird im Falle des Pensionsfonds jedoch nicht gegenüber dem leistungsberechtigten Arbeitnehmer garantiert. Falls der Pensionsfonds diesen Zins auf dem Kapitalmarkt nicht erwirtschaften kann, muss somit der Arbeitnehmer mit einer geringeren Leistung auskommen oder der Arbeitgeber die Versorgungslücke nachfinanzieren.

Für die Durchführungswege Direktversicherung, Pensionskasse und Pensionsfonds gilt, dass der Gesetzgeber die Beiträge zur bAV bis zu einer vordefinierten Grenze steuerfrei stellt und somit die Brutto-Steuerlast des Arbeitnehmers reduziert wird [BK17, S. 109]. Im Leistungsfall müssen die ausbezahlten Rentenbeiträge jedoch regulär versteuert werden [BK17, S. 122]. Dabei profitieren Arbeitnehmer meist von einem niedrigeren Steuersatz im Rentenalter als zur Zeit der Berufstätigkeit [Ver].

§ 3 Nr. 63 EStG regelt die Grenzen für die Beitragsfreistellungen: Bis zum Jahr 2017 war die Steuerfreiheit bei den aufgeführten Durchführungswegen auf 4% der Beitragsbemessungsgrenze der gesetzlichen Rentenversicherung (West) fixiert – plus weitere 1.800 Euro, falls kein weiterer bAV-Vertrag unter Nutzung der Pauschalbesteuerung durch § 40 b EStG bestand [BK17, S. 111].

Seit dem 01.01.2018 entfällt die damalige in § 3 Nr. 63 EStG festgeschriebene zusätzliche Förderung von 1.800 Euro, falls die Pauschalbesteuerung durch § 40 b EStG nicht genutzt wird [BK17, S. 111]. Die Fördergrenze für die Direktversicherung, Pensionskasse und Pensionsfonds beträgt nun einheitlich 8% der Beitragsbemessungsgrenze der gesetzlichen Rentenversicherung (West) [BK17, S. 111]). Hat ein Arbeitnehmer jedoch vor dem 01.01.2018 einen Beitrag nach § 40 b Absatz 1 und 2 EStG a.F. pauschal besteuern lassen, kann dieser auch weiterhin durch die Pauschalbesteuerung von § 40 b EStG erfasst werden

### *3. Anwendungsfall*

[BK17, S. 114 f.]: Dieser Beitrag wird dann von der erhöhten 8%-Fördergrenze abgezogen, damit Arbeitnehmer nicht zu sehr ihren steuerlichen Förderbeitrag ausweiten können [BK17, S. 111].

Im Jahr 2021 lag die Beitragsbemessungsgrenze der gesetzlichen Rentenversicherung für Westdeutschland bei 85.200 Euro jährlich [Bun21], was somit einer Fördergrenze von 6.816 Euro im Sinne des § 3 Nr. 63 EStG entspricht.

§ 40 b EStG ermöglichte bis 2004 die Beiträge zur bAV mit einer pauschalen Steuer von 20% zu versteuern [BK17, S. 113]. Seitdem besteht die Möglichkeit zur Pauschalbesteuerung nur noch unter bestimmten Voraussetzungen wie bei Verträgen von umlagefinanzierten Pensionskassen oder bei bestehenden Altverträgen [BK17, S. 113]. Unter anderem erfordert die Anwendung von § 40 b EStG auch, dass der Arbeitnehmer beim Abschluss des bAV-Vertrages in seinem ersten Arbeitsverhältnis war [BK17, S. 116]. Der Gesetzgeber begrenzt die Nutzung der Steuerpauschalisierung nach § 40 b EStG auf 1.752 Euro je Arbeitnehmer [BK17, S. 116]. Falls der Vertrag im Rahmen eines Gruppenvertrages mehrerer Arbeitnehmer abgeschlossen wurde und der jährliche Aufwand des Arbeitgebers pro Arbeitnehmer im Durchschnitt nicht 1.752 Euro überschreitet, erhöht sich dieser Maximalbetrag auf 2.148 Euro [BK17, S. 117].

Bei den Durchführungs wegen der Direktzusage und Unterstützungskasse verbleibt der Leistungsanspruch des Arbeitnehmers beim Arbeitgeber und wird nicht auf ein drittes Unternehmen übertragen. Der Arbeitnehmer ist nicht 'verfügungsberechtigt' [BK17, S. 137] über die Mittel, die der Arbeitgeber zurückstellt. Deshalb sind die Beiträge in diesen Fällen grundsätzlich nicht zu versteuern [BK17, S. 138]. Erst bei Eintritt des Versorgungsfalles erhält der Arbeitnehmer Zugriff auf die Leistungen, sodass diese Einkünfte dann im Rahmen von 'Einkünften nichtselbständiger Arbeit' [BK17, S. 138] versteuert werden müssen. Es existiert jedoch ein Versorgungsfreibetrag (§ 19 Abs. 2 EStG), der die Höhe der Besteuerung abmildert [BK17, S. 138]. Zudem gilt auch in diesem Fall, dass die allgemeine Steuerlast im Rentenalter meist geringer ausfällt als während der Zeit der Berufstätigkeit [Ver].

#### **3.1.2. Easy Web**

Easy Web [ALHb] ist ein webbasierter Portal zur Ermittlung verschiedenster Komponenten von Lebensversicherungsprodukten der ALH-Gruppe mit dem Ziel Versicherungsvermit-

### 3. Anwendungsfall

telnden bei der Beratung von Kundinnen und Kunden zu unterstützen. Abbildung 3.1 zeigt die Startseite von Easy Web beim Aufruf des Portals. Neben der Erstellung und Speicherung von 'Kundenberatungen' anhand verschiedener Altersvorsorge-Lösungen der ALH-Gruppe (siehe 'Neue Kundenberatung starten' und 'Vorhandene Kundenberatung öffnen' in Abbildung 3.1) besteht bei Easy Web auch die Möglichkeit ein Kundenprofil im Sinne der europäischen Versicherungsvertriebs-Richtlinie IDD zu erstellen (siehe 'Neue IDD-Kundenberatung starten' in Abbildung 3.1). Dies soll im Rahmen dieser Arbeit jedoch nicht berücksichtigt werden.



Abbildung 3.1.: Die Startseite von Easy Web [ALHb]: Es können Kundenprofile zur Kalkulation verschiedener Altersvorsorgeprodukte angelegt und gespeichert werden. Zudem besteht die Möglichkeit ein Kund\*innenprofil im Rahmen der IDD-Versicherungsvertriebsrichtlinie anzulegen. Screenshot erstellt am 15.12.2021.

Verschiedene Konzepte der privaten und betrieblichen Altersvorsorge stehen bei Easy Web zur Auswahl. Mittels einer Registerkarte können Nutzer\*innen von Easy Web bei einer Kund\*innenberatung zwischen den Bereichen der privaten und betrieblichen Altersvorsorge wechseln. Für beide Bereiche steht zudem eine weitere Registerkarte bereit, die zwischen Rentenprodukten, Berufsunfähigkeitsprodukten und Risikolebensversicherungen unterscheidet. Für jede dieser Registerkarte erscheinen mehrere Tarife, die zur Kalkulation ausgewählt werden können. Der Anwendungsfall dieser Arbeit beschränkt sich auf den Bereich der betrieblichen Altersvorsorge und den Bereich der Renten- beziehungswei-

### 3. Anwendungsfall

se 'Vorsorge'-Produkte. Dies entspricht der Zahlung einer lebenslangen Altersrente, einer einmaligen Kapitalauszahlung oder einer Mischung aus den beiden zuvor genannten Varianten bei Erreichen des vordefinierten Rentenalters. Abbildung 3.2 zeigt das Auswahlmenü für den Fall der bAV.

The screenshot shows the 'Beratungscockpit E@SY WEB LEBEN' interface. At the top, there are buttons for 'Abmelden' and the 'ALTE LEIPZIGER' logo. Below the header, there are tabs for 'Privat' (selected) and 'bAV'. Under 'bAV', there are three sub-tabs: 'Vorsorge' (selected), 'BU', and 'Risiko'. On the left, a sidebar lists 'Beratungstools', 'IDD-Beratung', 'Vorteilsrechner bAV' (expanded), 'Neuer Vorschlag', and 'Konzepte & Tarife'. Below the tabs, there is a section titled 'Konzeptauswahl Vorsorge' with four icons: 'Treuhandmodell' (hand icon), 'Solventa' (gavel icon), 'Pensionsfonds' (§ symbol), and 'International' (globe icon). The main content area is titled 'Tarife Vorsorge' and contains a table:

	Klassisch	Fondsgebunden
<b>Direktversicherung</b>	<ul style="list-style-type: none"> <li>&gt; Klassische Rente mit Rentengarantie (AR10)</li> <li>&gt; Klassische Rente mit Guthabenschutz (AR20)</li> </ul>	<ul style="list-style-type: none"> <li>&gt; Fondsrente mit Garantien (FR20)</li> <li>&gt; Smarte Rente (HR20)</li> </ul>
<b>U-Kasse</b>	<ul style="list-style-type: none"> <li>&gt; Klassische Rente mit Rentengarantie (AR10)</li> <li>&gt; Klassische Rente mit Guthabenschutz (AR20)</li> <li>&gt; Rente mit Rentengarantie (RV15)</li> <li>&gt; Rente mit Guthabenschutz (RV25)</li> <li>&gt; Rente ohne Todesfallsleistung (RV30)</li> <li>&gt; Lebensversicherung (LV10)</li> </ul>	<ul style="list-style-type: none"> <li>&gt; Fondsrente mit Garantien (FR20)</li> <li>&gt; Smarte Rente (HR20)</li> </ul>
<b>Direktzusage</b>	<ul style="list-style-type: none"> <li>&gt; Flexible Rente mit Rentengarantie (AR15)</li> <li>&gt; Flexible Rente mit Guthabenschutz (AR25)</li> <li>&gt; Klassische Rente mit Rentengarantie (AR10)</li> <li>&gt; Klassische Rente mit Guthabenschutz (AR20)</li> <li>&gt; Rente mit Rentengarantie (RV15)</li> <li>&gt; Rente mit Guthabenschutz (RV25)</li> <li>&gt; Rente ohne Todesfallsleistung (RV30)</li> <li>&gt; Lebensversicherung (LV10)</li> </ul>	<ul style="list-style-type: none"> <li>&gt; Flexible Rente mit Rentengarantie (AR15)</li> <li>&gt; Flexible Rente mit Guthabenschutz (AR25)</li> <li>&gt; Smarte Rente (HR20)</li> </ul>
<b>Pensionskasse</b>	> Pensionskasse (PK10)	

At the bottom of the interface, there are links for 'Impressum' and 'Datenschutz'.

Abbildung 3.2.: Das Auswahlmenü von Easy Web bei einer Kund\*innenberatung: Es wird zwischen privater und betrieblicher Altersvorsorge unterschieden, zudem können Tarife verschiedener Durchführungswege ausgewählt werden. Screenshot erstellt am 15.12.2021.

Easy Web ermöglicht die Auswahl von Produkten der bAV über vier der fünf möglichen Durchführungswege (vgl. Abschnitt 3.1.1), die je nach Wahl des Durchführungsweges über eines der verbundenen Unternehmen der ALH-Gruppe abgewickelt werden [ALHa]. Dabei gilt es zu berücksichtigen, dass Easy Web bzw. die ALH-Gruppe die Durchführungswege intern in einer anderen Art und Weise zuordnet als im allgemeinen Gebrauch: Easy Web unterscheidet im Bereich der Durchführungswege lediglich zwischen Direktversicherung, Pensionskasse und Direktzusage und führt zusätzlich den Parameter 'Art der Rückdeckung' ein. Dieser kann die Werte 'keine', 'Unterstützungskasse' oder 'Rückdeckungsversicherung' annehmen.

### 3. Anwendungsfall

- Direktversicherung / keine Rückdeckung: Abgewickelt über die Alte Leipziger Lebensversicherung a.G.
- Direktzusage / Unterstützungskasse: Abgewickelt über die Alte Leipziger Unterstützungskasse e.V., rückdeckungsversichert über die Alte Leipziger Lebensversicherung a.G.
- Direktzusage / Rückdeckungsversicherung: Abgewickelt über die Alte Leipziger Lebensversicherung a.G. als Rückdeckungsversicherung für eine Direktzusage eines Unternehmens.
- Pensionskasse / keine Rückdeckung: Abgewickelt über die Alte Leipziger Pensionskasse AG

Bei der Wahl der Anlagestrategie der Beiträge der Kundschaft und damit unmittelbar auch der garantierten Leistungen existieren im Bereich der Altersvorsorge die Varianten der klassischen, fondsgebundenen und hybriden Tarife. Bei klassischen Tarifen wird das Guthaben der Kundinnen und Kunden einzig im Deckungskapital des Versicherers angelegt [ALHc]. Bei fondsgebundenen Tarifen werden die Beiträge hingegen in Investmentfonds investiert und unterliegen somit den Schwankungen des Kapitalmarktes [ALHc]. Hybride Tarife vereinen die beiden Varianten der klassischen und fondsgebundenen Tarife und setzen auf mehrere Töpfe in der Anlage der Beiträge der Kundschaft [ALHc]. Im Bereich der bAV vertreibt die ALH-Gruppe keine Produkte mit reiner Fondsanlage, sondern bezeichnet Produkte mit hohem Anteil in der Fondsanlage und zusätzlichen Garantiekomponenten als fondsgebundene Produkte [ALHc]. Diese Produkte werden in der Eingabemaske von Easy Web auch als fondsgebunden angezeigt (vgl. Abbildung 3.2).

Im folgenden werden die verschiedenen Tarife vorgestellt, welche bei Easy Web ausgewählt werden können. Die Erläuterungen basieren auf [ALHc]:

- Rente mit Rentengarantie (RV15) / mit Guthabenschutz (RV25) / ohne Todesfallleistung (RV30): Als Variante der klassischen Tarife wandern die Beiträge der Versicherten bei diesen Tarifen vollständig in das Deckungskapital der Alten Leipziger Lebensversicherung a. G. und werden zudem mit einem Rechnungszins vollständig garantiert. Zusätzlich bestimmt die Wahl des Tarifs zusätzliche Bausteine im Todesfall des Leistungsempfängers: Bei Wahl der Rentengarantie wird ein bestimmtes Alter der versicherungsnehmenden Person bestimmt oder eine bestimmte Dauer ab

### *3. Anwendungsfall*

Rentenbeginn festgelegt, bis zu welchem die Rente auch im Todesfall an Hinterbliebene weiterhin ausbezahlt wird. Bei Wahl des Guthabenschutzes wird das verbleibende Guthaben nach Abzug der bereits gezahlter Renten an Hinterbliebene ausbezahlt. Die dritte Variante verzichtet auf jegliche Leistungen im Todesfall. Diese drei Tarife können bei Easy Web nur über die Durchführungswege der Direktzusage / Rückdeckungsversicherung oder Direktzusage / Unterstützungskasse ausgewählt werden.

- Moderne klassische Rente mit Rentengarantie (AR10) / mit Guthabenschutz (AR20): Die Tarife AR10 und AR20 gleichen in ihrer Anlagestruktur den Tarifen RV15, RV25 und RV30. Die Beiträge werden im Deckungsstock in sichere Kapitalanlagen investiert und werden für die Rentenbezugszeit garantiert. Die moderne klassische Rente basiert auf einem niedrigeren Garantiezins als die Tarife RV15, RV25 und RV30 und ermöglicht zudem eine flexible Gestaltung der Beitragshöhe. Der Todesfallschutz (mit Guthabenschutz / Rentengarantie) der modernen klassischen Rente gleicht den korrespondierenden Varianten der Tarife RV15 und R25. Die moderne klassische Rente ist in Easy Web über die Durchführungswege Direktversicherung, Direktzusage / Rückdeckungsversicherung und Direktzusage / Unterstützungskasse verfügbar.
- Pensionskasse (PK10): Dieser Tarif ist der einzige Tarif, der in Easy Web über den Durchführungsweg der Pensionskasse zur Verfügung steht. Er gleicht in seiner Anlagestruktur der modernen klassischen Rente mit Rentengarantie. Im Allgemeinen unterscheiden sich die beiden Tarife PK10 und AR10 kaum, die wesentliche Ausnahme bildet dabei die Abwicklung des Tarifs über die Pensionskasse im Falle des Tarifs PK10 und über die Lebensversicherungsgesellschaft im Falle des Tarifs AR10.
- Flexible Rente mit Rentengarantie (AR15) / mit Guthabenschutz (AR15): Die flexible Rente der ALH-Gruppe entspricht einem statischen hybriden Tarif. Kundinnen und Kunden können den Anteil des Investments in einen selbst gewählten Investmentfonds individuell spezifizieren. Der verbleibende Anteil der Beiträge wird dann im Deckungsstock angelegt und mit einem Rechnungszins garantiert. Für den Anteil der Anlage im Investmentfonds wird zudem ein Rentenfaktor garantiert. Die Tarifvarianten bezüglich des Todesfallschutzes entsprechen denen der modernen klassischen Rente. Ausgewählt werden kann die flexible Rente in Easy Web nur über den Durchführungsweg Direktzusage / Rückdeckungsversicherung.
- Fondsrente mit Garantien (FR20): Die Fondsrente mit Garantien ist ebenfalls ein hybrider Tarif, der im Gegensatz zu den Tarifen AR15 und AR25 einem 'dynamischen'

### 3. Anwendungsfall

schen 3-Topf-Modell' entspricht. Die Anlage der Beiträge setzt sich hier aus dem klassischen Sicherungsvermögen im Deckungsstock des Versicherers, einem Wertsiccherungsfonds und einem zusätzlichen individuell gewählten Investmentfonds zusammen. Die Beiträge werden dabei auf die drei verschiedenen Töpfe dynamisch während der Laufzeit aufgeteilt, sodass das Gesamtguthaben eine Verlustobergrenze in vordefinierten Zeiträumen nicht überschreitet. Das Gesamtguthaben der Kundinnen und Kunden wird bis zu einem vordefinierten Grad garantiert. Im Todesfall des Bezugsberechtigten der Rente besitzt der Tarif eine Rentengarantie mit Rentengarantiezeit. Die Fondsrente ist in Easy Web via Direktversicherung und Direktzusage / Unterstützungskasse verfügbar.

- Smarte Rente (HR20): Die smarte Rente entspricht ebenfalls einem dynamischen hybriden Tarif, der jedoch lediglich auf zwei Anlagetöpfe setzt. Neben der Anlage im Deckungsstock werden die Beiträge lediglich in einen Investmentfonds investiert, der von der ALH-Gruppe vorgeben ist. Zudem unterscheidet sich die smarte Rente gegenüber der Fondsrente derart, dass bei der smarten Rente das Vertragsguthaben vor Rentenbeginn in den Deckungsstock umgeschichtet wird, sodass das garantierte Kapital des Vertrages zu Rentenbeginn sichergestellt ist. Die möglichen Durchführungswege der smarten Rente sind die Direktversicherung, die Unterstützungskasse und die Direktzusage. Auch die smarte Rente besitzt eine Rentengarantie mit Rentengarantiezeit im Todesfall. Wie die moderne klassische Rente kann die smarte Rente bei Easy Web über die Durchführungswege Direktversicherung, Direktzusage / Rückdeckungsversicherung und Direktzusage / Unterstützungskasse ausgewählt werden.

Wählt man nun einen der zur Verfügung stehenden Tarife aus, öffnet sich eine Eingabemaske zur Eingabe verschiedener Parameter, welche zur Rentenkalkulation zur Rate gezogen werden. Neben Angaben zur versicherten Person müssen Nutzer\*innen von Easy Web zudem Informationen zum gewählten Tarif, eventuellen Zusatzversicherungen und der optionalen Beitragsdynamik angeben. Darüber hinaus werden je nach Wahl des Tarifs und Durchführungsweges detaillierte Angaben zum Versicherungsschutz verlangt: Dies umfasst bei allen Tarifen den Versicherungsbeginn, die Art der Finanzierung, die Höhe des Beitrags, das Renteneintrittsalter und die Periodizität der Beitragszahlungen. Ein weiteres Fenster unter der Bezeichnung 'Details', das jedoch nur bei bestimmten Tarifen angezeigt wird, erlaubt zudem die Angabe zusätzlicher detaillierter Faktoren: So können dort beispielsweise Zuzahlungen zu Beginn des Vertragsabschlusses (bei den Durchführungswegen

### 3. Anwendungsfall

The screenshot shows the 'Beratungcockpit' interface of E@SY WEB LEBEN. The top navigation bar includes 'Abmelden' and the logo 'ALTE LEIPZIGER'. The main workspace is titled 'Hauptversicherung' and displays the following information:

- Versicherte Person:** Herr Max Mustermann (01.01.1990)
- Titel:** Max
- Nachname:** Mustermann
- Geschlecht:** Männlich
- Geburtsdatum:** 01.01.1990
- Tarif:** AR10 - Klassische Rente mit Rentengarantiezeit
- Tarifgruppe:** Einzeltarif
- Zusatzversicherung:** Berufsunfähigkeit
- Dynamik:** gemaß (radio button selected) 4% 8% der BBG
- Versicherungsschutz:**
  - Versicherungsbeginn:** 01.01.2021
  - Finanzierung:** Mischfinanziert (AG-Betrag unverfallbar)
  - AN-Betrag:** 400,00 € darin enthaltene VL: 0,00 €
  - AG-Zuschuss (gesetzlich):** im AG-Betrag enthalten
  - AG-Betrag:**
    - In % des AN-Betrags: 15,00 % 60,00 €
    - als Betrag: 500,00 €
  - Gesamtbetrag:** 960,00 €
  - Versicherungsablauf:** Rentenbeginn entsprechend Geburtsdatum im Alter
  - Rentenbeginn im Alter:** 67 Jahre zum 01.01.2057 - Aufschubzeit 36 Jahre
  - Rentengarantiezeit:** 10 Jahre oder maximal
  - Überschussverwendung in der Rentenbezugszeit:** Rentenzuwachs
  - Beitragszahlungsweise:** Jährlich
  - Steuerbetrachtung:** Keine
- Details:**
  - Garantierte Rentensteigerung:** 0,0 %
  - Rentenzahlungsweise:** Jährlich
  - Zuzahlung zum Beginn:** 1.000,00 €
  - Bereits genutzter jährlicher Beitrag nach § 3 Nr. 63 EStG:** 500,00 €
  - Bereits genutzter jährlicher Beitrag nach § 40b EStG:** 250,00 €
  - Angenommene BBG-Entwicklung für Dynamik:** 2,5 %

Abbildung 3.3.: Die Eingabemaske für den Tarif 'Klassische Rente mit Rentengarantie (AR10)' des Durchführungswegs Direktversicherung. Im oberen Bereich werden Angaben zur versicherten Person verlangt, darunter folgt die Spezifizierung der Parameter für den Tarif. Screenshot erstellt am 15.12.2021.

Direktversicherung, Pensionskasse, Direktzusage) oder die genutzten Freibeträge im Sinne von § 3 Nr. 63 EStG und § 40 b EStG (bei den Durchführungswegen Pensionskasse und Direktversicherung, vgl. Abschnitt 3.1.1) spezifiziert werden. Abbildung 3.3 zeigt die Eingabemaske beispielhaft für den Tarif 'Klassische Rente mit Rentengarantie (AR10)' des Durchführungswegs Direktversicherung.

Nach Ausfüllen dieser einführenden Eingabemaske zur Angebotserstellung führt Easy Web Nutzer\*innen durch weiterführende Formularfelder: Diese können unter anderem die Spezifizierung der Zusatzversicherungen oder die Wahl der Kapitalanlage bei fondsgebundenen

### 3. Anwendungsfall

Tarifen sein. Anschließend berechnet Easy Web ein Ergebnis für die garantierte und zu erwartende Rente.

Fehlerhafte Eingaben werden von Easy Web als solche in den meisten Fällen erst beim Aufruf der Ergebnisseite angezeigt. Dies können einerseits Fehler bezüglich der Kund\*innendaten wie ein fehlendes Geburtsdatum sein, andererseits Fehler bezüglich der Werte zum Versicherungsschutz. In beiden Fällen verweist Easy Web auf die Haupteingabemaske zurück und zeigt den konkreten Eingabefehler an. Abbildung 3.4 zeigt eine beispielhafte Fehlermeldung im Falle einer Eingabe eines jährlichen Beitrags von 0 Euro für den Tarif AR10 und den Durchführungsweg der Direktversicherung.

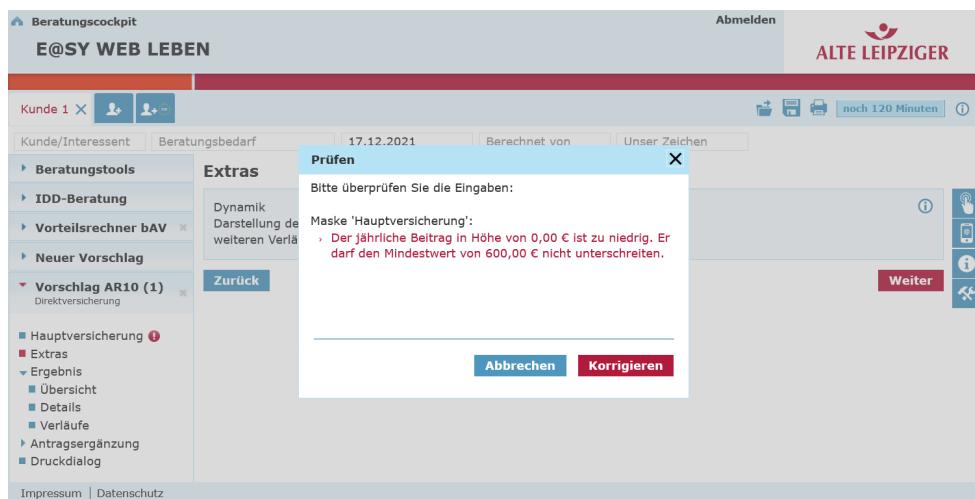


Abbildung 3.4.: Fehlermeldung von Easy Web bei einer Eingabe des Beitrags von 0 Euro im Tarif AR10 des Durchführungswegs Direktversicherung. Screenshot erstellt am 15.12.2021.

## 3.2. Implementierung

Im folgenden Abschnitt wird die konkrete Umsetzung zur Beantwortung der Fragestellung dieser Arbeit vorgestellt. Der zugehörige Programmcode kann öffentlich zugänglich auf der Github-Plattform eingesehen werden [Sin].

Das im vorausgehenden Abschnitt erwähnte Fehlerverhalten von Easy Web steht dabei im Fokus: Anhand der Methoden des Combinatorial Testing (vgl. Abschnitt 2.2) werden

### 3. Anwendungsfall

verschiedene Mengen an Testfällen zur Überprüfung der Plausibilität der möglichen Eingaben in Easy Web erstellt und verglichen. Dies erfolgt anhand mehrerer Ansätze, die unterschiedliche Teilfragestellungen beantworten:

1. Erstellung eines 'Basis-Systems' mit Combinatorial Testing unter Berücksichtigung von Bedingungen des zu testenden Systems: Ist es überhaupt möglich Testfälle für ein System wie Easy Web mittels Combinatorial Testing zu erstellen?
2. Vergleich Combinatorial Testing / Random Testing: Schafft Combinatorial Testing Vorteile gegenüber einem zufälligen Testverfahren?
3. Erweiterung des 'Basis-Systems' zu einem 'Erweiterten System': Inwieweit verändert die Einführung komplexerer Parameterstrukturen – im Speziellen die Einführung der Finanzierungsart und separate Parameter für Arbeitgeber- und Arbeitnehmerbeitrag – die Resultate im Vergleich zum einfacheren 'Basis-System'?
4.  $t$ -fache Kombinatorik für jeden Durchführungsweg: Ist es sinnvoll spezifische Parameter zu extrahieren – im Speziellen die Kombinationen der Durchführungswege und der Art der Rückdeckung – und für deren Werte jeweils unabhängig voneinander Testmengen mittels Combinatorial Testing zu erstellen?
5. Vergleich verschiedener Combinatorial Testing Tools / Algorithmen: Welcher Algorithmus beziehungsweise welches Tool ist für die praktische Anwendung am nützlichsten?

Als Grundlage aller Untersuchungen dienen die Vorsorge-Tarife der bAV in Easy Web, wie sie in Abbildung 3.2 zu erkennen sind. Der Tarif der Lebensversicherung (LV10) wird im Anwendungsfall nicht berücksichtigt.

Der Anwendungsfall bezieht sich aus Gründen der Übersichtlichkeit auf eine reduzierte Auswahl der möglichen Eingabeparameter der verschiedenen Tarife in der Eingabemaske von Easy Web (vgl. Abbildung 3.3): Angaben zur versicherten Person werden im Rahmen dieser Arbeit fest vorgegeben und sind nicht variabel, ebenso Parameter optionaler Zusatzversicherungen<sup>1</sup>. Außerdem werden eine progressive Beitragsentwicklung und jegliche Parameter zur Rentenauszahlung wie Rentenbeginn oder etwaige Todesfallleistungen ausgeschlossen. Letztere Annahme führt dazu, dass für die Tarife AR10 / AR20, AR15

---

<sup>1</sup>Verwendete Ausprägungen der Parameter zur versicherten Person und optionaler Zusatzversicherung: Max Mustermann, männlich, geb. 01.01.1980, keine Zusatzversicherung, keine Dynamik

### 3. Anwendungsfall

/ AR25, RV15 / RV25 / RV30 im Rahmen dieser Arbeit jeweils keine Unterschiede existieren und diese deshalb im Sinne einer Äquivalenzklassenbildung (vgl. Abschnitt 2.1.2.1) zusammengefasst werden: AR10 / AR20 wird im Folgenden als AREven bezeichnet, AR15 / AR25 als AROdd und RV15 / RV25 / RV30 als RVx.

Weitere Annahmen an das im Anwendungsfall untersuchte System sind: Die Beitragszahlung zur bAV erfolgt jährlich mit einem beliebigen, aber festen Beitrag. Dabei wird angenommen, dass die Arbeitgeberzulage stets 0% beträgt. Der Versicherungsbeginn wird auf den 01.01.2021 festgesetzt, der Rentenbeginn soll stets am 01.01. eines Jahres erfolgen. Dies bewirkt, dass das Versicherungsjahr dem Kalenderjahr entspricht und infolgedessen alle Beitragsgrößen sich auf das Kalenderjahr beziehen.

#### 3.2.1. Basis-System

Aus den getroffenen Annahmen ergibt sich ein zu testendes System aus verschiedenen Parametern, welches als Grundlage der Untersuchungen dieser Arbeit dient. Dieses Testsystem wird im Folgenden als 'Basis-System' bezeichnet. Das Basis-System umfasst sieben verschiedene Parameter und 25 Bedingungen, die berücksichtigt werden müssen. Die Bedingungen umfassen Tarifvarianten in Abhängigkeit des Durchführungswegs und der Art der Rückdeckung und Vorgaben in Bezug auf die Wahl der möglichen Werte für die verschiedenen Parameter. Konkrete Beispiele werden dazu in den folgenden Absätzen aufgeführt, zudem kann die vollständige Liste der Bedingungen im Anhang dieser Arbeit nachgelesen werden (vgl. Abschnitt A.1). Tabelle 3.1 zeigt einen Überblick über die verschiedenen Parameter des Basis-Systems und die zugehörigen möglichen Werte.

Parameter	Mögliche Werte
Durchführungsweg	Direktversicherung, Direktusage, Pensionskasse
Art der Rückdeckung <sup>2</sup>	keine, Rückdeckungsversicherung, Unterstützungskasse
Tarif	AREven, AROdd, FR20, HR20, PK10, RVx
Beitrag	0, 24, 299, 599, 600, 5.000, 125.001
Zuzahlung zu Beginn	0, 99, 299, 2.000, 5.001, 1.000.001
Bereits genutzter Beitrag nach § 40 b EStG	0, 2.148, 2.149
Bereits genutzter Beitrag nach § 3 Nr. 63 EStG	0, 2.068, 4.068

Tabelle 3.1.: Die Parameter und möglichen Werte des Basis-Systems

### 3. Anwendungsfall

Das Basis-System unterscheidet nicht zwischen den verschiedenen Finanzierungsarten der bAV: Als Vereinfachung wird angenommen, dass nur ein einziger Wert als Beitragshöhe existiert. Dies bedeutet im Konkreten: Falls die modellierte Rente Arbeitgeber-finanziert / Arbeitnehmer-finanziert ist, fällt der gewählte Beitrag vollständig dem Arbeitgeber / dem Arbeitnehmer zu. Falls die Rente Misch-finanziert ist, soll der Beitrag zur Hälfte auf beide Parteien aufgeteilt werden. Eine detaillierte Betrachtungsweise der Finanzierungsart wird im Erweiterten System (vgl. Abschnitt 3.2.3) aufgegriffen, wobei ebenfalls stets absolute Beiträge als Basis dienen. Im Kontext der Misch-Finanzierung werden daher relative Arbeitgeber-Anteile ausgeschlossen.

Für die Parameter Durchführungsweg, Art der Rückdeckung und Tarif entsprechen die möglichen Werte des Basis-System den Auswahlmöglichkeiten des Auswahlmenüs im Bereich 'Vorsorge' der bAV bei Easy Web (vgl. Abbildung 3.2). Einzige Ausnahme: Der Tarif LV10 wird ausgeschlossen.

Die möglichen Werte für Beitragshöhe, Zuzahlung zu Beginn, Bereits genutzter Beitrag nach § 40 b EStG und Bereits genutzter Beitrag nach § 3 Nr. 63 EStG ergeben sich aus einer Analyse des Fehlerverhaltens von Easy Web. Durch wiederholte Experimente bei der Nutzung von Easy Web und einer Analyse der Produktinformationen [ALHc] wurden untere und obere Schranken für die Höhe der Beiträge ermittelt, die Easy Web als Fehlergrenzen betrachtet. Tabelle 3.2 zeigt die Auflistung der Beitragsgrenzen in Abhängigkeit des Durchführungswegs, der Art der Rückdeckung und der zugehörigen Tarife. In roter Farbe sind die unteren Schranken aufgeführt, in grüner Farbe die oberen Schranken.

Für die Durchführungswege Direktversicherung und Pensionskasse ergibt sich eine kombinierte obere Schranke für Beitragshöhe, Zuzahlung zu Beginn, Bereits genutzter Beitrag nach § 40 b EStG und Bereits genutzter Beitrag nach § 3 Nr. 63 EStG: Die Summe dieser Parameter darf den Wert der 8%-Fördergrenze der Beitragsbemessungsgrenze nach § 3 Nr. 63 EStG nicht überschreiten (vgl. Abschnitt 3.1.1). Höhere Beiträge als der Wert von 6.816 Euro erlaubt Easy Web bei der Eingabe nicht. Darüber hinaus darf bei den Durchführungswege Direktversicherung und Pensionskasse der genutzte Beitrag im Sinne von § 40 b EStG den Maximalbetrag von 2.148 Euro nicht überschreiten (vgl. Abschnitt 3.1.1).

Für die Höhe des Beitrages existiert beim Durchführungsweg der Direktzusage eine obere Schranke, die durch den jährlichen Maximalbeitrag von 125.000 Euro festgesetzt ist. Falls

---

<sup>2</sup>Die Ausprägung der Unterstützungskasse wird in der Literatur üblicherweise als eigener Durchführungsweg betrachtet (vgl. dazu Abschnitt 3.1.1). Easy Web ordnet die Unterstützungskasse intern dem Durchführungsweg Direktzusage zu und führt sie als mögliche Art der Rückdeckung.

### 3. Anwendungsfall

Durchführungs-weg / Art der Rückdeckung	Finanz-ierung	Tarife	Beitrag	Zuzahlung	Verw. Beitrag § 40 b	Verw. Beitrag § 3(63)
Direkt-versicherung / keine	<i>AN, AG, Misch</i>	AREven, HR20, FR20	$\geq 600$	= 0 bzw. $\geq 100$	$\geq 0$	
			Summe aller Beiträge $\leq 6.816$ (BBG-Grenze) & Beitrag 40b $\leq 2.148$			
Pensionskasse / keine	<i>AN, AG, Misch</i>	PK10	$\geq 300$	= 0 bzw. $\geq 300$	$\geq 0$	
			Summe aller Beiträge $\leq 6.816$ (BBG-Grenze) & Beitrag 40b $\leq 2.148$			
Direktzusage / Unterstützungs-kasse	<i>AN, AG</i>	AREven, HR20	$\geq 600$	–		
			$\leq 125.000$ (Verweis auf Direktionsanfrage)			
		FR20	$\geq 300$			
			$\leq 125.000$ (Verweis auf Direktionsanfrage)			
		RVx	$\geq 25$ & wie bei Rückdeckungsversicherung / RVx	–		
			$\leq 125.000$ (Verweis auf Direktionsanfrage)			
Direktzusage / Rückdeckungs-versicherung	<i>AG</i>	AREven, AROdd, HR20	$\geq 600$	$\geq 0$	–	
			$\leq 125.000$ (Verweis auf Direktionsanfrage)	$\leq 1$ Mio.		
		RVx	Ergibt sich aus garantierter Rente von 600 Euro (hängt vom Alter ab)	= 0 bzw. $\geq 500$		
			$\leq 125.000$ (Verweis auf Direktionsanfrage)	$\leq 5.000$		

Tabelle 3.2.: Grenzen (in Euro) der möglichen Eingabewerte der Beitragsparameter in Easy Web in Abhängigkeit des Durchführungswegs, der Rückdeckung und der jeweiligen Tarife. Rot sind untere Schranken markiert, grün obere Schranken

dieser Wert überschritten wird, berechnet Easy Web keine Rente und zeigt eine Fehlermeldung mit einem Verweis auf eine notwendige Direktionsanfrage an. Für die untere Schranke gilt für die Tarife der klassischen RVx-Rente eine besondere Regelung: Die Untergrenze ergibt sich dabei aus einer garantierten jährlichen Rente von 600 Euro, die sich über die

### 3. Anwendungsfall

Faktoren der Beitragshöhe und der Dauer der Beitragszahlungen (beziehungsweise dem Alter der versicherten Person) ermitteln lässt.

Die bereits genutzten Beiträge nach § 40 b EStG nach § 3 Nr. 63 EStG spielen aufgrund der gesetzlichen Regelungen nur für die Durchführungswege Direktversicherung und Pensionskasse eine Rolle, eine Zuzahlung ermöglicht Easy Web lediglich beim Durchführungs weg Direktzusage und der Rückdeckung via Rückdeckungsversicherung. In diesen Fällen wird durch Bedingungen an das zu testende System der Wert 0 für die ausgeschlossenen Parameter festgesetzt. So gilt als Beispiel für die Kombination Direktzusage / Rückdeckungsversicherung: *Durchführungs weg = 'Direktzusage' & Art der Rückdeckung = 'Rückdeckungsversicherung'*  $\Rightarrow$  *Verw. Beitrag § 40 b = 0 \& Verw. Beitrag § 3(63) = 0*.

Darüber hinaus setzt Easy Web in bestimmten Fällen eine Mindestanforderung an die Höhe der Zuzahlung voraus: Die Zuzahlung bei den Kombinationen Direktversicherung / keine Rückdeckung, Pensionskasse / keine Rückdeckung und Direktzusage / Rückdeckungsversicherung / RVx muss entweder gleich 0 sein oder einen Mindestbetrag überschreiten (vgl. Tabelle 3.2).

Die konkreten Werte für Beitragshöhe, Zuzahlung zu Beginn, Bereits genutzter Beitrag nach § 40 b EStG und Bereits genutzter Beitrag nach § 3 Nr. 63 EStG wurden anhand der Analyse aus Tabelle 3.2 und den Methoden der Äquivalenzklassenbildung (vgl. Abschnitt 2.1.2.1) und Grenzwertanalyse (vgl. Abschnitt 2.1.2.2) ermittelt. Für jede Schranke des Systems wurden Äquivalenzklassen oberhalb und unterhalb der Schranke angenommen und darauf aufbauend mindestens ein Wert pro Klasse in das Testsystem aufgenommen. Dies gilt insbesondere auch für kombinierte Schranken mehrerer Parameter wie die 8%-Fördergrenze der Beitragsbemessungsgrenze nach § 3 Nr. 63 EStG. Zudem wurden im Besonderen die Grenzwerte im Sinne der Grenzwertanalyse betrachtet, sodass pro Äquivalenzklasse die betroffenen Schranken möglichst exakt getroffen werden. Beispiel hierfür sind die Werte 2.148 Euro und 2.149 Euro für den bereits genutzten Beitrag nach § 40 b EStG.

Die aus diesem Vorgehen resultierenden Werte für die verschiedenen Beitragsparameter sind in Tabelle 3.3 dargestellt. Dabei gilt es zu berücksichtigen, dass nicht alle Werte der Beitragsparameter, die in Tabelle 3.1 aufgeführt sind, für alle Durchführungswege relevant sind: Durch zusätzliche Bedingungen an das Testsystem werden bei der Erstellung der Testmengen lediglich die in Tabelle 3.3 aufgeführten Werte für die Kombinationen

### 3. Anwendungsfall

der Durchführungswege, Art der Rückdeckung und Tarife bei der Erstellung der Testfälle berücksichtigt. Beispielsweise gilt für die Kombination Direktversicherung / keine Rückdeckung bei allen Tarifen folgende Bedingung: *Durchführungs weg = 'Direktversicherung'*  $\Rightarrow \text{Beitrag} = 0 // \text{Beitrag} = 599 // \text{Beitrag} = 600$ .

Durchführungs weg / Art der Rückdeckung	Tarife	Beitrag	Zuzahlung	Verw. Beitrag § 40 b	Verw. Beitrag § 3 #63
Direktversicherung	AREven, HR20, FR20	0, 599, 600	0, 99, 2.000	0, 2.148, 2.149	0, 2.068, 4.068
Pensionskasse	PK10	0, 299, 599, 600	0, 299, 2.000	0, 2.148, 2.149	0, 2.068, 4.068
Direktzusage / Unterstützungs kasse	AREven, HR20	0, 599, 600, 125.001	–		
	FR20	0, 299, 600, 125.001	–		
	RVx	0, 24, 299, 600, 125.001	–		
Direktzusage / Rückdeckungs versicherung	AREven, AROdd, HR20	0, 599, 600, 125.001	0, 2.000, 1.000.001	–	
	RVx	0, 24, 299, 5.000, 125.001	0, 299, 2.000, 5.001	–	

Tabelle 3.3.: Mögliche Werte für die verschiedenen Beitragsparameter für das Basis-System in Abhängigkeit des Durchführungswege, der Art der Rückdeckung und des Tarifs.

Das vorgestellte Basis-System dient als Grundlage zur Erstellung einer Menge an Testfälle im Sinne von Combinatorial Testing (vgl. Abschnitt 2.2). Für die Beantwortung der grundsätzlichen Fragestellung dieser Arbeit wurden Testmengen mit einem Interaktionsparameter von  $t = 2$  bis  $t = 6$  via ACTS (vgl. Abschnitt 2.2.4.1) und dem allgemeinen IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) erstellt.

#### 3.2.2. Vergleich Combinatorial Testing / Random Testing

Die grundlegende Fragestellung, die durch Erstellung von Testmengen des Basis-Systems beantwortet wird, sollte im Weiteren durch einen Vergleich mit einer naiven, zufallsba-

### 3. Anwendungsfall

sierten Herangehensweise erweitert werden und aufzeigen, inwieweit ein systematisches Vorgehen bei der Erstellung von Testmengen hilfreich sein kann.

Dafür wurde ein Random Testing-Ansatz (vgl. Abschnitt 2.1.2) für das zuvor vorgestellte Basis-System implementiert: Für eine vordefinierte Anzahl  $N$  an Testfällen erzeugt der implementierte Random-Testing-Algorithmus zufällige Testfälle, indem für jeden Parameter des Basis-Systems zufällig ein Wert ausgewählt wird. Der erzeugte Testfall wird dann auf Konsistenz mit den 25 Bedingungen des Basis-Systems geprüft: Falls jener Testfall die Bedingungen des Basis-Systems nicht erfüllt, wird dieser verworfen und ein neuer zufälliger Testfall erstellt. Darüber hinaus prüft der Algorithmus die Testfälle auf Duplikate und verwirft diese im Falle eines bereits existierenden Testfalls.

Als Vergleichsgrößen zum Ansatz des Combinatorial Testing dienen unter anderem Effizienzmetriken des Random Testing-Ansatzes wie die Ausführungszeit und die Anzahl der benötigten Iterationen, die zur Erstellung von  $N$  Testfällen benötigt werden. In Bezug auf die Ausführungszeit gilt bei allen Untersuchungen dieser Arbeit, dass diese unter identischen Bedingungen auf derselben Maschine durchgeführt wurden.

Als weitere Vergleichsgrößen des Random Testings wurden die Abdeckung im Sinne von Combinatorial Testing mit den Metriken der  $t$ -fachen-Abdeckung (vgl. Abschnitt 2.2.2.4), der Variablen-Wert-Abdeckung (vgl. Abschnitt 2.2.2.6) und der  $(0,75-t)$ -Vollständigkeit (vgl. Abschnitt 2.2.2.7) geprüft für  $t \in \{2, 3\}$ . Das Random Testing-Verfahren wurde für  $N \in \{50, 100, 200, 300\}$  mit jeweils 20 Wiederholungen durchgeführt. Die Ergebnisse wurden schließlich als Mittelwert der 20 Wiederholungen berechnet.

#### 3.2.3. Erweitertes System

Das Basis-System beinhaltet keine spezifische Betrachtungsweise der Finanzierungsart und einer damit verbundenen Trennung zwischen Arbeitgeber- und Arbeitnehmer-Beitrag. Das 'Erweiterte System' greift diesen Aspekt auf und soll aufzeigen, inwiefern sich die Erstellung von Testmengen via Combinatorial Testing auf komplexere Systeme skalieren lässt.

Das Erweiterte System ergänzt das Basis-System durch die Parameter Finanzierungsart (Arbeitgeber-, Arbeitnehmer-, Misch-finanziert) und Arbeitnehmer- und Arbeitgeberbeitrag, der Parameter Beitrag des Basis-Systems entfällt. Das Erweiterte System besitzt folglich neun verschiedene Parameter und zusätzliche sieben Bedingungen an das Testsystem im Vergleich zum Basis-System: Neu hinzugefügte Bedingungen umfassen unter anderem

### 3. Anwendungsfall

bestimmte Kombinationen aus Durchführungsweg und Art der Rückdeckung, die spezifische Finanzierungsarten voraussetzen. Beispielsweise wird bei Easy Web die Kombination Direktzusage / Rückdeckungsversicherung nur als Arbeitgeber-finanzierte Variante durchgeführt. Die resultierende Bedingung lautet: *Durchführungsweg = 'Direktzusage'  $\wedge \wedge$  Art der Rückdeckung = 'Rückdeckungsversicherung'*  $\Rightarrow$  Finanzierungsart = 'AG-finanziert'. Die verschiedenen Finanzierungsmöglichkeiten abhängig vom Durchführungsweg, Art der Rückdeckung und Tarif können Tabelle 3.2 entnommen werden. Tabelle 3.4 zeigt eine Übersicht über die Parameter und möglichen Werte des Erweiterten Systems.

Parameter	Mögliche Werte
Durchführungsweg	Direktversicherung, Direktzusage, Pensionskasse
Art der Rückdeckung	keine, Rückdeckungsversicherung, Unterstützungskasse
Tarif	AREven, AROdd, FR20, HR20, PK10, RVx
<b>Finanzierung</b>	AG-finanziert, AN-finanziert, Misch-finanziert
<b>AN-Beitrag</b>	0, 299, 599, 600, 1.500, 125.001
<b>AG-Beitrag</b>	0, 24, 299, 599, 600, 1.500, 125.001
Zuzahlung zu Beginn	0, 99, 299, 2.000, 5.001, 1.000.001
Bereits genutzter Beitrag nach § 40 b EStG	0, 2.148, 2.149
Bereits genutzter Beitrag nach § 3 Nr. 63 EStG	0, 2.068, 4.068

Tabelle 3.4.: Die Parameter und möglichen Werte des Erweiterten Systems: Das Erweiterte System umfasst zusätzliche Parameter bezüglich der Finanzierungsart und der Unterscheidung in Arbeitgeber- und Arbeitnehmerbeitrag

Analog zur Ermittlung der Werte der Parameter Beitragshöhe, Zuzahlung zu Beginn, Bereits genutzter Beitrag nach § 40 b EStG und Bereits genutzter Beitrag nach § 3 Nr. 63 EStG im Basis-System wurden die Werte für die verschiedenen Parameter des Erweiterten Systems auf Basis der Beitragsgrenzen in Tabelle 3.2 festgesetzt. Dabei wurden wie beim Basis-System die Methoden der Äquivalenzklassenmethode (vgl. Abschnitt 2.1.2.1) und Grenzwertanalyse (vgl. Abschnitt 2.1.2.1) kombiniert. Eine Übersicht über die möglichen Werte der verschiedenen Parameter des Erweiterten Systems (Tabelle A.1) und die Bedingungen des Erweiterten Systems (Abschnitt A.2) befindet sich im Anhang dieser Arbeit.

Das Erweiterte System wurde mit dem Basis-System in Bezug auf die Anzahl der erzeugten Testfälle für die Interaktionsparameter  $t \in \{2, 3, 4, 5, 6\}$  verglichen, ebenfalls unter Anwendung des allgemeinen IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) via ACTS (vgl.

### 3. Anwendungsfall

Abschnitt 2.2.4.1): Dabei wurde zudem untersucht, wie sich das Laufzeitverhalten durch die zusätzlichen Parameter verändert.

#### 3.2.4. *t-fache Kombinatorik für jeden Durchführungsweg*

Sowohl beim Basis-System als auch beim Erweiterten System sind die Parameter Durchführungsweg und Art der Rückdeckung Teil der Kombinatorik bei der Erstellung der Testfallmengen mit  $t$ -facher Abdeckung. Dabei kann es jedoch passieren, dass durch die Reduktion der Kombinatorik auf den Wert  $t$  erwünschte Kombinationen verschiedener Parameter unberücksichtigt bleiben. Tabelle 3.5 verdeutlicht dieses Prinzip: Sei  $t = 2$  gegeben, dann erfüllt ein Testfall mit Tarif = 'AREven' und Beitrag = '600' bereits die notwendige 2-fach-Abdeckung für das Parameterpaar Tarif / Beitrag. In Tabelle 3.5 entspricht dies dem ersten Eintrag mit Durchführungsweg Direktversicherung ohne Rückdeckung.

Durchführungsweg	Art der Rückdeckung	Tarif	Beitrag
Direktversicherung	keine	AREven	600
Direktzusage	Unterstützungskasse	AREven	599
Direktzusage	Rückdeckungsversicherung	AREven	125.001

Tabelle 3.5.: Entstehende Problematik bei Einbezug des Durchführungswegs und der Art der Rückdeckung in die  $t$ -fach-Kombinatorik bei der Erstellung von Testmengen: Für  $t = 2$  wird durch den ersten Testfall die Kombination Tarif = 'AREven' / Beitrag = 600 bereits abgedeckt und somit möglicherweise nicht mehr beim Durchführungsweg Direktzusage verwendet.

Die Beitragsgrenze von 600 Euro spielt jedoch nicht nur für den Durchführungsweg der Direktversicherung eine entscheidende Rolle, sondern auch für die Direktzusage, wie Tabelle 3.2 aufzeigt: Die Kombination Tarif = 'AREven' / Beitrag = 600 ist im Beispiel in Tabelle 3.5 bereits abgedeckt, sodass für die Direktzusage andere Wertepaare der Parameter Tarif und Beitrag bevorzugt berücksichtigt werden – im Konkreten zunächst das Paar ('AREven' / 599), anschließend ('AREven' / 125.001). Es kann also passieren, dass für den Durchführungsweg der Direktzusage bei einem Interaktionsparameter  $t = 2$  eine Testmenge ohne einen einzigen Testfall mit dem Beitrag von 600 Euro erzeugt wird.

### 3. Anwendungsfall

Um diesem Effekt zu begegnen, wurde ein erweiterter Ansatz zur Erstellung von Testfällen mittels Combinatorial Testing implementiert: Dieser Ansatz beinhaltet die Extrahierung der Parameter Durchführungsweg und Art der Rückdeckung und die Erstellung von Testmengen via Combinatorial Testing für jede einzelne Wertekombination dieser beiden Parameter. Dieses Vorgehen stellt sicher, dass sich die kombinatorische Entfaltung von Combinatorial Testing lediglich auf die Beitragsparameter auswirkt. Als Analogie zur Benutzeroberfläche von Easy Web entspricht dieses Vorgehen der Trennung zwischen Auswahlmenü des Durchführungsweges und der Art der Rückdeckung (Abbildung 3.2) und der Eingabemaske (Abbildung 3.3) der jeweiligen Tarife.

Tabelle 3.6 stellt die Umsetzung diesen Ansatz modellhaft dar. Bei der Erstellung der jeweiligen Testmengen wurde das Erweiterte System als Grundlage der Implementierung gewählt, da durch die Extrahierung der beiden Parameter Durchführungsweg und Art der Rückdeckung beim Basis-System lediglich fünf Parameter verbleiben würden. Dies würde die Wertekombinationen erheblich reduzieren, insbesondere würde eine Abdeckung mit Interaktionsparameter  $t \in \{5, 6\}$  einer vollständigen Kombinationsabdeckung entsprechen (vgl. Abschnitt 2.2.2.1).

Durchführungsweg	Art der Rückdeckung	Tarif	Finanzierung	AN-Beitrag	AG-Beitrag	Zuzahlung zu Beginn	Verw. Beitrag § 40 b	Verw. Beitrag § 3 #63
Direktversicherung	keine		— $t$ -fache Kombinationsabdeckung —					
Pensionskasse	keine		— $t$ -fache Kombinationsabdeckung —					
Direktzusage	Rückdeckungsversicherung		— $t$ -fache Kombinationsabdeckung —					
Direktzusage	Unterstützungskasse		— $t$ -fache Kombinationsabdeckung —					

Tabelle 3.6.: Modellhafte Umsetzung des Ansatzes der separaten Erstellung von Testmengen mit  $t$ -fach-Abdeckung für jeden Durchführungsweg und jede Art der Rückdeckung.

Unter Verwendung des Erweiterten Systems verbleiben sieben verschiedene Parameter für den Ansatz der  $t$ -fache Kombinatorik für jeden Durchführungsweg und jede Art der Rückdeckung. Die verschiedenen Bedingungen des Erweiterten Systems (vgl. Abschnitt A.2) wurden den jeweiligen Durchführungswegen / Arten der Rückdeckung zugeordnet und angepasst. Die Erstellung der Testfallmengen für die verschiedenen Durchführungswege /

### 3. Anwendungsfall

Arten der Rückdeckung erfolgte via IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) und ACTS (vgl. Abschnitt 2.2.4.1). Anschließend wurden diese um die Werte des Durchführungswegs / der Art der Rückdeckung erweitert und aneinander gefügt.

#### 3.2.5. Vergleich verschiedener Combinatorial Testing Algorithmen

Zur Beantwortung der abschließenden Teilfragestellung wurden verschiedene Tools zur Erstellung von Combinatorial Testing-Testmengen verglichen. Im Konkreten wurden die Algorithmen IPOG (vgl. Abschnitt 2.2.3.2), IPOG-F (vgl. Abschnitt 2.2.3.2), PICT (vgl. Abschnitt 2.2.4.2) und CASA (vgl. Abschnitt 2.2.3.3) untersucht. Weitere von ACTS unterstützte Algorithmen (IPOG-D, IPOG-F2, PaintBall, Base Choice) erfüllen die notwendigen Anforderungen des Testsystems, wie beispielsweise die Berücksichtigung von Bedingungen, nicht und wurden an dieser Stelle ausgeschlossen. Die Anwendung von IPOG und IPOG-F erfolgte über die Benutzeroberfläche von ACTS, PICT und CASA wurden über die Kommandozeile gestartet.

Die 25 Bedingungen des Basis-Systems führen beim Algorithmus CASA dazu, dass die Standardgrößen für eine obere und untere Schranke der Anzahl an Testfälle beim Start des Algorithmus (Outer Search bei CASA, vgl. Abschnitt 2.2.3.3) sehr weit über der tatsächlichen, minimalen Anzahl an Testfällen liegt und der Algorithmus nicht immer terminiert. Dies erfordert die Angabe einer realistischen oberen und unteren Grenze der Anzahl der Testfälle, welche als Parameter über die Kommandozeile definiert werden können: Die Resultate der Algorithmen IPOG und PICT dienten hierfür als Referenzgrößen.

Für alle vier Algorithmen wurden Testmengen für die Interaktionsparameter  $t \in \{2, 3, 4, 5, 6\}$  erstellt. Anschließend wurden die Anzahl der generierten Testfälle, die Ausführungszeit und für  $t \in \{2, 3, 4, 5\}$  die Metriken der  $(t+1)$ -Kombinationsabdeckung (vgl. Abschnitt 2.2.2.5), der  $(t + 1)$ -Variablen-Wert-Abdeckung (vgl. Abschnitt 2.2.2.6) und der  $(0,75-(t + 1))$ -Vollständigkeit (vgl. Abschnitt 2.2.2.7) ermittelt. Für den nicht-deterministischen Algorithmus CASA wurde die Erstellung der Testmenge 30-fach wiederholt und der Median der zuvor aufgeführten Metriken berechnet. Zudem wurde bei CASA für jeden Interaktionsparameter die Testmenge mit der geringsten Anzahl an erzeugten Testfällen ermittelt. Als Testsystem diente das Basis-System (vgl. Abschnitt 3.2.1).

### 3. Anwendungsfall

## 3.3. Ergebnisse

Im folgenden Abschnitt werden die Resultate der Untersuchungen zur Beantwortung der verschiedenen Teilfragestellungen dieser Arbeit vorgestellt. Die Ausführungen folgen dabei der Struktur des vorherigen Abschnitts zur Implementierung (vgl. Abschnitt 3.2).

### 3.3.1. Basis-System

Das Basis-System (vgl. Abschnitt 3.2.1) mit sieben verschiedenen Parametern und 25 Bedingungen besitzt für den Fall des Pairwise-Testing (Interaktionsparameter  $t = 2$ ) 293 verschiedene Variablen-Wert-Kombinationen, die bei der Erstellung einer Testmenge abgedeckt werden müssen. Tabelle 3.7 zeigt die abzudeckenden Variablen-Wert-Konstellationen für die Werte  $t \in \{2, 3, 4, 5, 6\}$ . Mit Erhöhung des Interaktionsparameter  $t$  steigt die Anzahl der abzudeckenden Variablen-Wert-Kombinationen bis zum Wert von 3.081 für  $t = 5$  an. Für den Parameter  $t = 6$  sinkt dieser Wert auf 1.873, was sich durch den steigenden Einfluss der Bedingungen des Testsystems im Zusammenhang mit der geringer werdenden Differenz zwischen Interaktionsparameter und Anzahl der Parameter des Testsystems erklären lässt.

t	Anzahl an Variablen-Wert-Kombinationen
2	293
3	1.214
4	2.617
5	3.081
6	1.873

Tabelle 3.7.: Anzahl der Variablen-Wert-Kombinationen des Basis-Systems

Als Testmenge erstellt ACTS eine Tabelle mit Testfällen, wie sie in Tabelle 3.8 beispielhaft dargestellt ist. Die Anzahl der Testfälle und die Ausführungszeit der Erstellung der Testmenge unter Anwendung des allgemeinen IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) können Tabelle 3.13 in Abschnitt 3.3.5 entnommen werden: Demnach benötigt ACTS für alle Interaktionsparameter  $t \in \{2, 3, 4, 5, 6\}$  weniger als 0,14 Sekunden zur Erzeugung der Testmengen für das Basis-System. Die Anzahl der Testfälle reicht von 49 für  $t = 2$  bis zu

### 3. Anwendungsfall

458 für  $t = 6$ . Die  $(t + 1)$ -Kombinationsabdeckung, die  $(t + 1)$ -Variablen-Wert-Abdeckung und die  $(0,75 - (t + 1))$ -Vollständigkeit wachsen jeweils mit der Höhe des Interaktionsparameters  $t$ : Die  $(t + 1)$ -Kombinationsabdeckung besitzt eine Spannweite von 14,3% ( $t = 2$ ) bis 71,4% ( $t = 6$ ), die  $(t + 1)$ -Variablen-Wert-Abdeckung von 69,6% ( $t = 2$ ) bis 84,1% ( $t = 6$ ) und die  $(0,75 - (t + 1))$ -Vollständigkeit von 48,6% ( $t = 2$ ) bis 100% ( $t = 6$ ).

Durchführungs-weg	Art der Rückdeckung	Tarif	Beitrag	Zuzahlung zu Beginn	Verw. Beitrag §40 b	Verw. Beitrag § 3(63)
Pensionskasse	keine	PK10	600	0	0	2.068
Direktzusage	Rückdeckungs-versicherung	RVx	0	299	0	0
Direkt-versicherung	keine	AREven	599	2.000	2.149	4.068
...	...	...	...	...	...	...

Tabelle 3.8.: Beispielhafte Menge an Testfälle für das Basis-System

#### 3.3.2. Vergleich Combinatorial Testing / Random Testing

Die Erstellung von Testmengen anhand des Zufallsprinzips ergibt für alle gewählten Werte  $N \in \{50, 100, 200, 300\}$  eine unvollständige  $t$ -fache Kombinationsabdeckung. Dies wurde bei allen 20 Wiederholungen des Random Testing-Ansatzes für jeden Wert von  $N$  beobachtet. Tabelle 3.9 zeigt die weiteren Resultate der Untersuchungen zum Ansatz des Random Testing.

Über alle Interaktionsparameter hinweg ist die Effizienz der zufällig erzeugten Testfälle gering: Der Anteil der erfolgreichen Iterationen, also derjenigen Testfälle, welche die Bedingungen des Basis-Systems nicht verletzen und kein Duplikat darstellen, beträgt durchschnittlich rund 2%. Die Ausführungszeit liegt für  $N \in \{50, 100\}$  im Durchschnitt bei weniger als 10 Sekunden und wächst für  $N \in \{200, 300\}$  auf die Werte von durchschnittlich 311,77 Sekunden ( $N = 200$ ) und 668,45 Sekunden ( $N = 300$ ) an.

Für den Interaktionsparameter  $t = 2$  beträgt die mittlere  $t$ -fache Kombinationsabdeckung bei 50 zufällig generierten Testfällen 39,0% und die Variablen-Wert-Abdeckung

### 3. Anwendungsfall

N	$\varnothing$ Ausführungszeit in s	$\varnothing$ Anzahl Iteratio- nen	$\varnothing$ Anteil erfolgrei- cher Iteratio- nen	t=2			t=3		
				$\varnothing$ t- Abdeckung	$\varnothing$ Variablen- Wert- Abdeckung	$\varnothing$ (0,75- t)-Voll- ständig- keit	$\varnothing$ t- Abdeckung	$\varnothing$ Variablen- Wert- Abdeckung	$\varnothing$ (0,75- t)-Voll- ständig- keit
50	3,72	2.487	2,2%	39,0%	78,0%	74,5%	9,0%	60,1%	42,4%
100	7,15	4.742	2,4%	64,8%	88,0%	91,4%	25,4%	77,0%	79,0%
200	311,77	11.626	2,3%	80,0%	94,5%	98,1%	48,1%	89,0%	93,0%
300	668,45	21.798	2,2%	88,6%	97,4%	100,0%	69,6%	94,5%	99,3%

Tabelle 3.9.: Ergebnisse Random Testing

78,0%. Im Vergleich dazu erzeugt ein systematisches Vorgehen via ACTS und dem IPOG-Algorithmus eine Testfallmenge mit 49 Testfällen mit vollständiger 2-fachen Kombinations- und Variablen-Wert-Abdeckung. Die Ausführungszeit beträgt dabei 0,125 Sekunden.

Eine Erhöhung der vorgegebenen Anzahl an Testfälle sorgt beim Random Testing-Ansatz für eine Steigerung der  $t$ -fachen Variablen-Wert-Abdeckung und der  $t$ -fachen-Kombinationsabdeckung. Bei einer Menge von 300 zufällig erzeugten Testfällen liegt die Variablen-Wert-Abdeckung für  $t = 2$  im Mittelwert bei 97,4%, die 2-fach Abdeckung beträgt dann durchschnittlich 88,6%. Alle Parameterpaare werden im Fall von  $N = 300$  mit mindestens 75% aller möglichen Variablen-Wert-Kombinationen abgedeckt – die (0,75-2)-Vollständigkeit beträgt folglich 100%.

Für eine vollständige Kombinationsabdeckung des Interaktionsparameters  $t = 3$  benötigt der IPOG-Algorithmus via ACTS 144 Testfälle (vgl. Tabelle 3.13). Im Vergleich dazu erreicht der zufallsbasierte Ansatz bei 100 Testfällen durchschnittlich eine 3-fache Kombinationsabdeckung von 25,4%, bei 200 Testfällen von 48,1% und bei 300 Testfällen von 69,6%. Die Variablen-Wert-Abdeckung in Bezug auf  $t = 3$  liegt in diesen Fällen bei 77,0% ( $N = 100$ ), 89,0% ( $N = 200$ ) und 99,3% ( $N = 300$ ).

### 3.3.3. Erweitertes System

Für das Erweiterte System (vgl. Abschnitt 3.2.3) erstellt ACTS Testmengen, welche die zusätzlichen Parameter Finanzierung, Arbeitnehmer-Beitrag (AN-Beitrag) und Arbeitgeber-Beitrag (AG-Beitrag) berücksichtigen. Tabelle 3.10 zeigt eine beispielhafte Menge an Testfällen.

### 3. Anwendungsfall

Durchführungs-weg	Art der Rückdeckung	Tarif	Finanzierungs-	AN-Beitrag	AG-Beitrag	Zuzahlung zu Beginn	Beitrag 40b	Beitrag 363
Direkt-versicherung	keine	AREven	AN	599	0	2.000	0	2.068
Direktzusage	Rückdeckungs-versicherung	RVx	AG	0	1.500	299	0	0
Pensionskasse	keine	PK10	Misch	600	600	2.000	0	2.068
...	...	...	...	...	...	...	...	...

Tabelle 3.10.: Beispielhafte Menge an Testfälle für das Erweiterte System

Die Resultate im Vergleich zur Erstellung der Testmengen des Basis-Systems sind in Tabelle 3.11 dargestellt. Durch die Einführung der zusätzlichen Parameter steigt die Anzahl der abzudeckenden Variablen-Wert-Konfigurationen: Für  $t = 2$  erfordert eine vollständige Kombinationsabdeckung die Berücksichtigung 293 verschiedener Variablen-Wert-Kombinationen, für das Erweiterte System müssen 522 Variablen-Wert-Kombinationen abgedeckt werden. Dies entspricht dem 1,78-fachen Wert des Basis-Systems. Mit steigendem Interaktionsparameter wächst das Verhältnis der abzudeckenden Variablen-Werte des Erweiterten Systems im Vergleich zum Basis-System in exponentiellem Maße: Das Verhältnis liegt für  $t = 3$  bei 2,68, für  $t = 4$  bei 4,48, für  $t = 5$  bei 8,54 und für  $t = 6$  bei 19,86. Im letzteren Fall  $t = 6$  müssen beim Basis-System 1.873 Variablen-Wert-Kombinationen abgedeckt werden, das Erweiterte System erfordert 37.196 Kombinationen.

t	Basis-System			Erweitertes System		
	Anzahl an Testfälle	Ausführungszeit in s	Anzahl an Variablen-Wert-Kombinationen	Anzahl an Testfälle	Ausführungszeit in s	Anzahl an Variablen-Wert-Kombinationen
<b>2</b>	49	0,125	293	67	0,187	522
<b>3</b>	144	0,109	1.214	214	0,150	3.253
<b>4</b>	271	0,110	2.617	559	0,243	11.723
<b>5</b>	447	0,167	3.081	1.339	0,456	26.318
<b>6</b>	458	0,140	1.873	2.630	0,919	37.196

Tabelle 3.11.: Vergleich der Ergebnisse des Basis-Systems und des Erweiterten Systems

Die erhöhte Anzahl an abzudeckenden Variablen-Wert-Kombinationen macht sich in der Ausführungszeit und der Anzahl benötigter Testfälle bemerkbar, wie Abbildung 3.5 auf-

### 3. Anwendungsfall

zeigt. In einer Gegenüberstellung werden in Abbildung 3.5 beide Systeme in Bezug auf die Anzahl der Testfälle (linke Seite) und der Ausführungszeit (rechte Seite) miteinander verglichen.

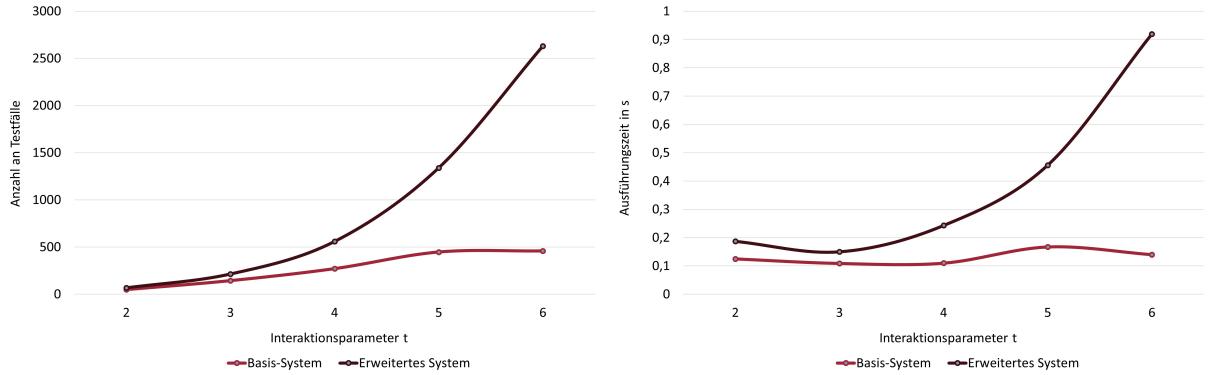


Abbildung 3.5.: Gegenüberstellung des Basis-Systems und des Erweiterten Systems in Bezug auf die Anzahl der Testfälle (linke Seite) und die Ausführungszeit (rechte Seite) bei der Erstellung von Testmengen via ACTS und allgemeinem IPOG-Algorithmus

Für  $t = 2$  erstellt ACTS via IPOG-Algorithmus 49 Testfälle für das Basis-System und 67 Testfälle für das Erweiterte System. Diese Anzahl steigt mit erhöhtem Interaktionsparameter für beide Systeme bis  $t = 5$  an, im Falle des Erweiterten Systems jedoch mit deutlich größerer Wachstumsrate. Während für das Basis-System im Zuge der abnehmenden Anzahl an Variablen-Wert-Kombinationen für  $t = 6$  auch die Anzahl der benötigten Testfälle nur geringfügig im Vergleich zum Interaktionsparameter  $t = 5$  wächst, zeigt sich beim Erweiterten System auch für  $t = 6$  ein exponentielles Wachstum im Vergleich zu  $t = 5$ . Für  $t = 6$  stehen insgesamt 2.630 Testfälle des Erweiterten System 458 Testfällen des Basis-Systems gegenüber.

Ähnliche Entwicklungen lassen sich auch in Bezug auf die Ausführungszeit beobachten: Für das Basis-System sind nur unwesentliche Unterschiede in der Ausführungszeit in Abhängigkeit des Interaktionsparameters  $t$  zu erkennen. Für das Erweiterte System lässt sich analog zur Anzahl der Testfälle ein exponentielles Wachstum in Bezug auf die Ausführungszeit beobachten. Die Spannweite der Ausführungszeit beträgt in diesem Fall 0,187 Sekunden für  $t = 2$  bis 0,919 Sekunden für  $t = 6$ .

### 3. Anwendungsfall

#### 3.3.4. $t$ -fache Kombinatorik für jeden Durchführungsweg

Der Ansatz der Erstellung von Testmengen mit  $t$ -facher Kombinationsabdeckung für jede Konstellation aus Durchführungsweg und Art der Rückdeckung auf Basis des Erweiterten Systems (vgl. Abschnitt 3.2.4) sorgt bei allen Interaktionsparametern  $t \in \{2, 3, 4, 5, 6\}$  für eine höhere Anzahl an erzeugten Testfällen im Vergleich zum klassischen Vorgehen des Erweiterten Systems aus dem vorhergehenden Abschnitt. Tabelle 3.12 zeigt in der Übersicht die benötigte Anzahl an Testfällen der verschiedenen Durchführungswege / Arten der Rückdeckung und deren Summe bei der Erstellung von Testmengen mit den Interaktionsparametern  $t \in \{2, 3, 4, 5, 6\}$ .

Durchführungsweg	Art der Rückdeckung	Anzahl der Testfälle				
		$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$
Direktversicherung	keine	23	79	264	677	1.648
Pensionskasse	keine	27	102	312	809	945
Direktzusage	Rückdeckungsversicherung	30	56	56	56	56
Direktzusage	Unterstützungskasse	26	32	32	32	32
<b>Gesamt</b>		<b>106</b>	<b>269</b>	<b>664</b>	<b>1.574</b>	<b>2.681</b>
<i>Vergleich Erweitertes System</i>		67	214	559	1.339	2.630

Tabelle 3.12.: Anzahl der Testfälle bei der Erstellung von separaten Testmengen für jeden Durchführungsweg und jede Art der Rückdeckung: Als Vergleichsgröße wird zudem die Anzahl der Testfälle des Vorgehens des Erweiterten Systems aufgeführt.

Für den Interaktionsparameter  $t = 2$  beträgt die Anzahl der erzeugten Testfälle des Ansatzes der  $t$ -fachen Kombinatorik für jeden Durchführungsweg und jede Art der Rückdeckung 106 Stück. Im Vergleich dazu erstellt ACTS für das Erweiterte System rund 37% weniger Testfälle, im konkreten 67 Stück.

Mit steigendem Interaktionsparameter  $t$  wächst die Anzahl der Testfälle sowohl für den Ansatz der  $t$ -fachen Kombinatorik für jeden Durchführungsweg und jede Art der Rückdeckung als auch für den Ansatz des Erweiterten Systems exponentiell an (vgl. dazu Abschnitt 3.3.3). Parallel dazu sinken die Unterschiede zwischen beiden Herangehensweisen: Bereits für  $t = 3$  besitzt die erzeugte Testmenge des Erweiterten Systems nur noch 21%

### 3. Anwendungsfall

weniger Testfälle als jene des Ansatzes der  $t$ -fache Kombinatorik für jeden Durchführungs-weg und jede Art der Rückdeckung. Für  $t = 4$  beträgt der Unterschied 16%, für  $t = 5$  noch 15% und für  $t = 6$  knappe 2%.

In Bezug auf die jeweiligen Durchführungswege / Arten der Rückdeckung lässt sich ein exponen-tielles Wachstum der Testfälle bei den Durchführungswegen Direktversicherung und Pensionskasse beobachten, die Anzahl der Testfälle für die beiden Varianten des Durch-führungswegs Direktzusage bleibt hingegen für  $t \geq 3$  konstant. Restriktionen in Bezug auf die Finanzierungsart, Zuzahlung zu Beginn und die nicht vorhandenen Werte für die Pa-rameter Verwendeter Beitrag § 40 b und Verwendeter Beitrag § 3 Nr. 63 (vgl. Tabelle 3.2) und die daraus resultierende, reduzierte Kombinatorik sind dafür ursächlich.

#### 3.3.5. Vergleich verschiedener Combinatorial Testing Algorithmen

Im Folgenden werden die Resultate der vier untersuchten Algorithmen IPOG (vgl. Ab-schnitt 2.2.3.2), IPOG-F (vgl. Abschnitt 2.2.3.2), PICT (vgl. Abschnitt 2.2.4.2) und CASA (vgl. Abschnitt 2.2.3.3) zur Beantwortung der letzten Teilfragestellung dieser Arbeit vor-gestellt: Tabelle 3.13 zeigt die Analyse der Ergebnisse für IPOG, Tabelle 3.14 für IPOG-F, Tabelle 3.15 für PICT und Tabelle 3.16 für CASA.

Als hauptsächliche Kenngrößen zum Vergleich der erstellten Testmengen dienten die Aus-führungszeit der Algorithmen und die Anzahl der erstellten Testfälle zur  $t$ -fachen Kombi-nationsabdeckung: IPOG, IPOG-F und PICT liegen im Hinblick auf die Ausführungszeit eng beisammen und erzeugen Testmengen für das Basis-System in weniger als einer Se-kunde. Schnellster Algorithmus für  $t \in \{2, 3, 4, 5\}$  ist der IPOG-F-Algorithmus, der 0,062

t	Anzahl an Testfälle	Ausführungs-zeit in s	t+1		
			(t + 1)-Abdeckung	Variablen-Wert-Abdeckung	(0,75-(t + 1))-Vollständigkeit
2	49	0,125	14,3%	69,6%	48,6%
3	144	0,109	28,6%	84,1%	82,9%
4	271	0,11	47,6%	91,8%	95,2%
5	447	0,167	71,4%	98,8%	100,0%
6	458	0,140			

Tabelle 3.13.: Ergebnisse IPOG-Algorithmus

### 3. Anwendungsfall

t	Anzahl an Testfälle	Ausführungszeit in s	t+1		
			(t + 1)-Abdeckung	Variablen-Wert-Abdeckung	(0,75-(t + 1))-Vollständigkeit
2	63	0,062	25,7%	77,4%	71,4%
3	191	0,062	34,3%	88,4%	91,4%
4	408	0,109	57,1%	94,2%	95,2%
5	718	0,141	71,4%	98,6%	100,0%
6	848	0,143			

Tabelle 3.14.: Ergebnisse IPOG-F-Algorithmus

t	Anzahl an Testfälle	Ausführungszeit in s	t+1		
			(t + 1)-Abdeckung	Variablen-Wert-Abdeckung	(0,75-(t + 1))-Vollständigkeit
2	51	0,105	14,3%	69,9%	48,6%
3	148	0,117	28,6%	83,9%	85,7%
4	280	0,178	47,6%	92,2%	95,2%
5	445	0,333	71,4%	98,6%	100,0%
6	458	0,443			

Tabelle 3.15.: Ergebnisse PICT-Algorithmus

t	Median Anzahl an Testfälle	Median Ausführungszeit in s	t+1		
			Median (t + 1)-Abdeckung	Median Variablen-Wert-Abdeckung	Median (0,75-(t + 1))-Vollständigkeit
2	47,0	3,23	14,3%	68,2%	48,6%
3	138,5	9,35	28,6%	82,6%	82,9%
4	256,0	25,48	47,6%	90,6%	95,2%
5	503,5	109,34	71,4%	98,5%	100,0%
6	714,0	285,89			

Tabelle 3.16.: Ergebnisse CASA-Algorithmus

Sekunden für die Erzeugung einer Testmenge zur 2-fach- und 3-fach-Abdeckung, 0,109 Sekunden zur 4-fach-Abdeckung und 0,141 Sekunden zur 5-fach-Abdeckung benötigt. Für  $t = 6$  ist der allgemeine IPOG-Algorithmus minimal schneller als der IPOG-F-Algorithmus

### 3. Anwendungsfall

mit einer Ausführungszeit von 0,140 Sekunden (IPOG-F-Algorithmus: 0,143 Sekunden). PICT rechnet durchschnittlich 2,13 Mal so lange wie IPOG-F und 1,74 Mal so lange wie IPOG. Für  $t = 2$  ist PICT mit einer Ausführungszeit von 0,105 Sekunden sogar etwas schneller als IPOG.

CASA rechnet insbesondere für größere Interaktionsparameter wesentlich länger als die drei anderen Algorithmen: Für  $t = 2$  beträgt die durchschnittliche Ausführungszeit der 30 Iterationen des Algorithmus 3,23 Sekunden, bereits für  $t = 4$  liegt dieser Wert bei 25,48 Sekunden. Für  $t = 5$  wächst die durchschnittliche Ausführungszeit auf fast zwei Minuten (109,34 Sekunden) an, für  $t = 6$  beträgt sie annähernd fünf Minuten (285,89 Sekunden).

In Bezug auf die Anzahl der erzeugten Testfälle zeigt sich ein differierendes Bild im Vergleich der Algorithmen: Tabelle 3.17 stellt die Anzahl der generierten Testfälle der verschiedenen Algorithmen gegenüber, Abbildung 3.6 visualisiert diese Gegenüberstellung. Dabei gilt es zu berücksichtigen, dass für CASA in Abbildung 3.6 das Minimum und in Tabelle 3.17 zusätzlich der Median der 30 durchgeföhrten Wiederholungen je Interaktionsparameter als Vergleichswert herangezogen wurden.

t	Anzahl an Testfälle				
	IPOG	IPOG-F	PICT	Minimum CASA	Median CASA
<b>2</b>	49	63	51	<b>45</b>	47
<b>3</b>	144	191	148	<b>136</b>	138,5
<b>4</b>	271	408	280	<b>251</b>	256
<b>5</b>	447	718	<b>445</b>	451	503,5
<b>6</b>	<b>458</b>	848	<b>458</b>	519	714

Tabelle 3.17.: Vergleich aller Algorithmen / Tools in Bezug auf die minimale Anzahl an Testfälle

Für  $t \in \{2, 3, 4\}$  generiert CASA die kleinste Testmenge zur  $t$ -fachen Kombinationsabdeckung: Für  $t = 2$  entspricht dies einer Anzahl von 45 Stück, für  $t = 3$  von 136 Stück und für  $t = 4$  von 251 Stück. Bei  $t = 2$  wurde die geringste Anzahl von 45 Testfällen bei 2 der 30 durchgeföhrten Iterationen erreicht, die 136 Stück bei  $t = 3$  und 251 Stück bei  $t = 4$  jeweils nur ein einziges Mal. Auch im Median erzeugt CASA auch für  $t \in \{2, 3, 4\}$  die geringsten Testmengen: Für  $t = 2$  liegt der Median der 30 durchgeföhrten Iterationen des Algorithmus bei 47 Testfällen, für  $t = 3$  bei 138,5 Testfällen und für  $t = 4$  bei 256 Testfällen.

### 3. Anwendungsfall

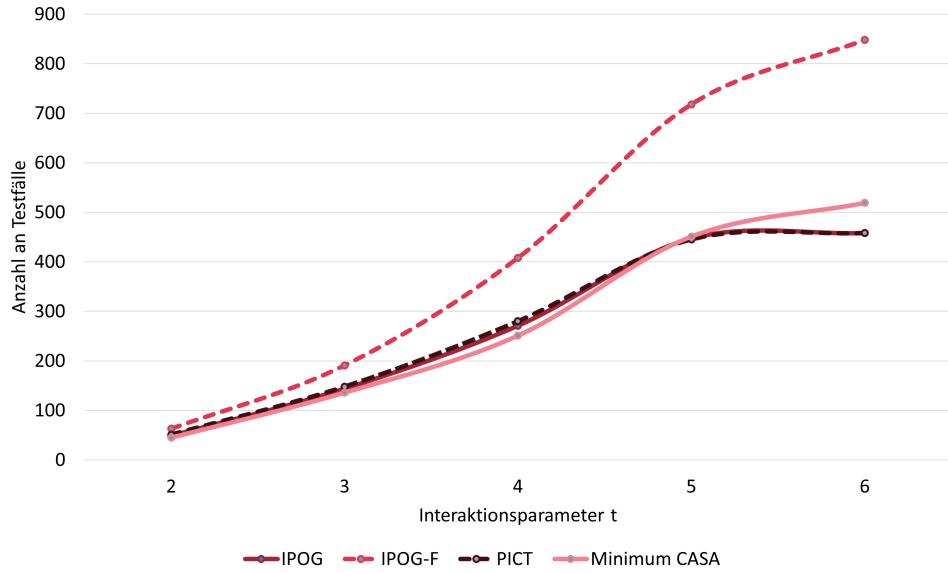


Abbildung 3.6.: Grafische Visualisierung der Anzahl erzeugter Testfälle der verschiedenen Algorithmen

IPOG und PICT erstellen für  $t \in \{2, 3, 4\}$  in geringem Maße mehr Testfälle als CASA, für  $t \in \{5, 6\}$  ist das Gegenteil der Fall: Mit 49, 144 und 271 Testfällen für  $t \in \{2, 3, 4\}$  liegt die Anzahl der Testfälle bei IPOG im Schnitt 7,5% über der minimalen Anzahl der Testfälle bei CASA für diese drei Interaktionsparameter. PICT erzeugt mit 51, 148 und 280 Stück für  $t \in \{2, 3, 4\}$  noch geringfügig mehr Testfälle, im Mittel liegt die Anzahl dort 11,2% über dem Wert von CASA. Für  $t = 5$  besitzt PICT die minimale Anzahl an Testfällen mit 445 Stück. IPOG erstellt an dieser Stelle 447 Testfälle, CASA im Minimum 451 Testfälle. Bei  $t = 6$  erstellen PICT und IPOG beide Testmengen mit der minimalen Anzahl an Testfällen von 458 Stück, CASA besitzt im Minimum 519 Testfälle und im Median 714.

Der IPOG-F-Algorithmus wurde bisher nicht in den Vergleich der Anzahl der Testfälle einbezogen: Der Grund dafür ist, dass IPOG-F für alle Interaktionsparameter  $t$  wesentlich mehr Testfälle generiert als die anderen Algorithmen. Für  $t = 2$  erstellt der IPOG-F-Algorithmus 63 Testfälle und liegt damit 18 Testfälle beziehungsweise 40% über dem Minimalwert des CASA-Algorithmus. Über alle Interaktionsparameter hinweg erzeugt IPOG-F im Mittel 58% mehr Testfälle als das Minimum der drei anderen Algorithmen. Im Fall von  $t = 5$  ist der Unterschied zwischen IPOG-F-Algorithmus mit 718 Testfällen und 445 Testfällen des PICT-Algorithmus am größten (85%).

### *3. Anwendungsfall*

In Bezug auf die weiteren untersuchten Metriken der  $(t + 1)$ -Kombinationsabdeckung, der  $(t + 1)$ -Variablen-Wert-Abdeckung und der  $(0,75-(t + 1))$ -Vollständigkeit lässt sich beobachten, dass über alle Algorithmen hinweg eine erhöhte Anzahl an Testfällen eine Steigerung der Abdeckung aller  $(t + 1)$ -Metriken bewirkt: So liegen die Werte der  $(t + 1)$ -Kombinationsabdeckung, der  $(t + 1)$ -Variablen-Wert-Abdeckung und der  $(0,75-(t + 1))$ -Vollständigkeit des IPOG-F-Algorithmus für  $t = 2, 3, 4$  bei höherer Anzahl an Testfällen auch stets über den Werten der anderen Algorithmen. Für  $t = 5$  besitzen alle vier Algorithmen die identische  $(t + 1)$ -Kombinationsabdeckung von 71,4% und  $(0,75-(t + 1))$ -Vollständigkeit von 100%, in Bezug auf die  $(t + 1)$ -Variablen-Wert-Abdeckung liegt der IPOG-Algorithmus mit 98,8% vorne.

## 4. Diskussion

Die Ergebnisse aller verschiedenen Teilfragestellungen zeigen auf, dass mittels Combinatorial Testing bei komplexen Systemen wie Easy Web Testfallmengen auf sinnvolle Art und Weise erstellt werden können. Dies gilt unabhängig vom konkreten Ansatz und des verwendeten Algorithmus beziehungsweise des verwendeten Tools: Alle verschiedenen Methoden führen in geringem Zeitaufwand zu geeigneten Ergebnissen im Sinne einer hohen praxisrelevanten Nutzbarkeit.

Diese grundlegende Erkenntnis spiegelt jedoch nur ein grobes Gesamtbild der Ergebnisse der verschiedenen Untersuchungen dieser Arbeit wider: Aus diesem Grund sollen im Folgenden die einzelnen Resultate der Teilfragestellungen aus Abschnitt 3.2 und Abschnitt 3.3 zusammengefasst und interpretiert werden.

1. *Ist es überhaupt möglich Testfälle für ein System wie Easy Web mittels Combinatorial Testing zu erstellen?*

ACTS (vgl. Abschnitt 2.2.4.1) erstellt via IPOG-Algorithmus (vgl. Abschnitt 2.2.3.2) Testmengen zur vollständigen  $t$ -fachen Kombinationsabdeckung für das Basis-System in weniger als 0,14 Sekunden (vgl. Abschnitt 3.2.1) und für das Erweiterte System (vgl. Abschnitt 3.2.3) in weniger als 0,92 Sekunden. Somit zeigt sich, dass sich in geringem zeitlichen Rahmen Testfälle für das Anwendungsbeispiel Easy Web erstellen lassen.

Dabei gilt es im Besonderen zu berücksichtigen, dass innerhalb des Anwendungsfalls Bedingungen integriert sind, die bei der Erstellung der Testmengen berücksichtigt werden mussten. Auch dies scheint angesichts der Ergebnisse des Basis-Systems (vgl. Abschnitt 3.3.1) und der Ergebnisse des Erweiterten Systems (vgl. Abschnitt 3.3.3) keine nennenswerten Auswirkungen auf die Anwendbarkeit der Methoden des Combinatorial Testing in der Praxis zu besitzen.

#### 4. Diskussion

Vielmehr lässt sich beobachten, dass weniger die Ausführung der Algorithmen selbst bei der Erstellung von Testmengen zur  $t$ -fachen Kombinationsabdeckung eine Herausforderung darstellen, sondern eher die vorgelagerten Schritte zur Ermittlung der Parameter, deren Ausprägungen und etwaigen Bedingungen, welche das Testsystem erfüllen sollte.

Die Ausführungen zur Implementierung des Basis-Systems (vgl. Abschnitt 3.2.1) und die des Erweiterten Systems (vgl. Abschnitt 3.2.3) zeigen die Relevanz einer adäquaten Analyse des Testobjekts: Die Kombinationsmöglichkeiten der verschiedenen Parameter, sowie unterschiedliche, zu testende Wertebereiche und Grenzwerte erfordern ein hohes Verständnis über Easy Web. Anschaulich lässt sich dies an den komplexen Strukturen des Fehlerverhaltens von Easy Web in Tabelle 3.2 erkennen: Eine unzureichende Analyse an dieser Stelle würde zu einer ungeeigneten Wahl der Werte der verschiedenen Parameter führen (vgl. Tabelle 3.3 für das Basis-System, Tabelle A.1 für das Erweiterte System) und somit in ungeeigneten Testfällen resultieren. Ähnliche Beobachtungen konnten Mehta und Philip [MP13] in ihrer Arbeit zur Anwendung von Combinatorial Testing im Kontext der Finanz- und Versicherungsbranche machen.

Erfahrungswerte in Bezug auf die Erstellung von Testfällen (vgl. Abschnitt 2.1.2.4) und die geeignete Nutzung systematischer Verfahren wie die Äquivalenzklassenmethode (vgl. Abschnitt 2.1.2.1) und die Grenzwertanalyse (vgl. Abschnitt 2.1.2.2) erwiesen sich im Rahmen dieser Arbeit zur Meisterung der beschriebenen Herausforderung als nützlich.

##### 2. Schafft Combinatorial Testing Vorteile gegenüber einem zufälligen Testverfahren?

Im Kontext der Resultate des Ansatzes des Random Testing zeigt sich, inwiefern die Nutzung der Methoden des Combinatorial Testing Vorteile gegenüber anderen, naiven Vorgehensweisen besitzt: Wie Abschnitt 3.3.2 aufzeigt, besitzt Random Testing eine geringe Effizienz in der Ausführung von 2% und damit verbunden eine lange Ausführungszeit (vgl. Abschnitt 3.3.2). Besonders für eine hohe Anzahl der zu erzeugenden Testfälle ( $N = 200, 300$ ) macht sich dieser Effekt besonders bemerkbar.

Hauptursache dafür sind die verschiedenen Bedingungen des Basis-Systems, welche in jeder Iteration eines zufällig generierten Testfalls geprüft werden müssen und in vielen Fällen nicht mit dem Testfall konsistent sind. Darüber hinaus fallen die Werte der Metriken des Combinatorial Testing ( $t$ -fache Kombinationsabdeckung,

#### 4. Diskussion

Variablen-Wert-Abdeckung und  $(0,75-t)$ -Vollständigkeit) bei den die zufällig generierten Testfallmengen gering aus.

Der Ansatz des Combinatorial Testing ermöglicht im Gegensatz zum Random Testing die Erstellung von Testmengen, die wesentlich kleiner ausfallen und zugleich eine höhere, garantierte Qualitätsgüte im Sinne der Metriken des Combinatorial Testing (vgl. Abschnitt 2.2.2) besitzen. Außerdem sind die Algorithmen und Tools des Combinatorial Testing im Kontext der verschiedenen Bedingungen des Basis-Systems wesentlich schneller in ihrer Ausführung, wie die Ergebnisse des Vergleichs der verschiedenen Algorithmen aufzeigen (vgl. Abschnitt 3.3.5).

Ungeachtet der schlechteren Ergebnisse erfordert die Anwendung des Random Testing-Ansatzes genauso wie der Ansatz des Combinatorial Testing eine umfassende Analyse des Testobjekts, insbesondere müssen abzuprüfende Bedingungen des Testsystems genauso analysiert und implementiert werden. Aus diesem Grund bietet der Random Testing-Ansatz nur in geringem Maße Vorteile in Bezug auf eine einfachere und effizientere Implementierung.

3. *Inwieweit verändert die Einführung komplexerer Parameterstrukturen die Resultate im Vergleich zum einfacheren 'Basis-System'?*

Die Erweiterung des Basis-Systems zum Erweiterten System belegen, dass sich die Erkenntnisse der grundlegenden Anwendbarkeit von Combinatorial Testing im Anwendungsfall auf komplexere Problemstellungen skalieren lassen.

Durch die Einführung zusätzlicher Parameter und zusätzlicher Bedingungen steigt die Komplexität des Testsystems an, was sich auch in den Resultaten aus Abschnitt 3.3.3 widerspiegelt: Über alle Interaktionsparameter hinweg wächst die Ausführungszeit, die Anzahl der abzudeckenden Werte-Kombinationen und die Anzahl der Testfälle beim Erweiterten System exponentiell an, während das Wachstum dieser Kenngrößen des Basis-Systems prinzipiell linear verläuft (vgl. Tabelle 3.11, Abbildung 3.5). Diese Erkenntnis entspricht den Erwartungen, welche aus Gleichung 2.3 über das Wachstumsverhalten der Anzahl der Testfälle hervorgeht.

Nichtsdestotrotz bleibt festzuhalten, dass die Erstellung der Testmengen via ACTS für das Erweiterte System eine geringe Ausführungszeit von weniger als einer Sekunde für alle Interaktionsparameter besitzt (vgl. Tabelle 3.11). Somit lässt sich die Erstellung der Testmengen via Combinatorial Testing mit geringem Zusatzaufwand

#### 4. Diskussion

auf komplexere Systeme skalieren. Dabei gilt wie bereits für das Basis-System, dass die Analyse des Testobjekts und die Ermittlung passender Parameter, Werte und Bedingungen als größere Einflussfaktoren bei der Erstellung von Testmengen einzustufen sind als die Ausführung selbst.

4. *Ist es sinnvoll spezifische Parameter zu extrahieren und für deren Werte jeweils unabhängig voneinander Testmengen mittels Combinatorial Testing zu erstellen?*

Letzterer Aspekt der vorherigen Teilfragestellungen spielt auch für die Beantwortung dieser Teilfragestellungen eine entscheidende Rolle: Das Extrahieren der Parameter Durchführungsweg / Art der Rückdeckung und die Erstellung von Testmengen für jede Kombination dieser beiden Parameter bewirkt eine Steigerung der Anzahl der Testfälle im Vergleich zum Erweiterten Systems (vgl. Abschnitt 3.3.4). Diese fällt jedoch insbesondere für große Interaktionsparameter  $t \in \{4, 5, 6\}$  gering aus, sodass die zusätzlichen Kosten der Erstellung von Testmengen via ACTS und der resultierenden Anzahl der Testfälle auch bei diesem Ansatz nicht in besonderem Maße ins Gewicht fallen.

Wie die Ausführungen aus Abschnitt 3.2.4 aufzeigen, ist der Ansatz der  $t$ -fachen Kombinatorik für jeden Durchführungsweg und jede Art der Rückdeckung besonders praxisrelevant und sinnvoll, da er die Zusammenhänge zwischen den verschiedenen Parametern von Easy Web berücksichtigt und etwaige Probleme einer naiven Anwendung der Methoden des Combinatorial Testing ausräumen kann. Die Umsetzung dieses Ansatzes erfordert jedoch ein tiefes Verständnis über die Zusammenhänge innerhalb von Easy Web und veranschaulicht die Relevanz der Erfahrung und Expertise der testenden Person bei der Anwendung der Methoden des Combinatorial Testing.

5. *Welcher Algorithmus beziehungsweise welches Tool ist für die praktische Anwendung am nützlichsten?*

ACTS (vgl. Abschnitt 2.2.4.1) via IPOG (Abschnitt 2.2.3.2) und PICT (vgl. Abschnitt 2.2.4.2) sind im Kontext der Ergebnisse dieser Arbeit als diejenigen Tools einzustufen, welche für die Praxisanwendung am nützlichsten erscheinen. Dies lässt sich einerseits auf die Resultate der Analyse der unterschiedlichen Algorithmen (vgl. Abschnitt 3.3.5) und andererseits auf die Anwenderfreundlichkeit der Tools zurückführen.

#### 4. Diskussion

In Bezug auf die Erstellung von Testmengen mit möglichst geringer Anzahl an Testfällen und geringer Ausführungszeit zeigt sich, dass IPOG und PICT über alle Interaktionsparameter hinweg eng beisammen liegen und geeignete Resultate liefern. In Bezug auf die Ausführungszeit ist IPOG via ACTS bei größeren Interaktionsparametern marginal schneller als PICT.

Der schnellste Algorithmus der vier untersuchten Alternativen, IPOG-F (vgl. Abschnitt 2.2.3.2), erstellt im Vergleich zu IPOG, PICT und dem vierten verglichenen Algorithmus CASA wesentlich größere Testmengen und ist daher in der praktischen Anwendung eher ungeeignet. Dieses Ergebnis widerspricht in gewisser Weise den erwarteten Ergebnissen, da IPOG-F laut seinen Autoren nicht nur in kürzerer Zeit, sondern auch in geringerer Anzahl Testfälle erstellen sollte (vgl. Abschnitt 2.2.3.2). Eine mögliche Erklärung hierfür könnte sein, dass IPOG-F die Bedingungen des Basis-Systems ineffizienter in seinen Algorithmus integriert als die anderen Algorithmen.

Der CASA-Algorithmus generiert für  $t \in \{2, 3, 4\}$  zwar die kleinste Menge an Testfällen für das Basis-System (vgl. Tabelle 3.17), besitzt jedoch einige Einschränkungen, welche die Anwendung in der Praxis erschweren. Als wesentlicher Aspekt lässt sich diesbezüglich die Volatilität von CASA anführen: Da CASA nicht deterministisch bei der Erstellung der Testmengen vorgeht, fallen die Ergebnisse des Algorithmus bei gleichbleibenden Rahmenbedingungen in verschiedenen Iterationen unterschiedlich aus und liefern nur in seltenen Fällen das optimale Minimum (vgl. Abschnitt 3.3.5). Im konkreten Anwendungsfall wurde die minimale Anzahl der Testfälle des CASA-Algorithmus in lediglich zwei ( $t = 2$ ) und einer ( $t \in \{3, 4, 5, 6\}$ ) von 30 durchgeführten Iterationen ermittelt. Eine weitere Einschränkung des CASA-Algorithmus ergibt sich durch die erhöhte Ausführungszeit im Vergleich zu den anderen Algorithmen (vgl. Tabelle 3.16): Insbesondere für hohe Interaktionsparameter ( $t \in \{4, 5, 6\}$ ) dauert die Ausführung des CASA-Algorithmus wesentlich länger als bei IPOG, IPOG-F und PICT.

Darüber hinaus besitzt CASA – wie in Abschnitt 3.2.5 beschrieben – Probleme bei der Festsetzung einer oberen und unteren Schranke für die Anzahl der benötigten Testfälle bei Ausführung des Algorithmus: Im Speziellen geht CASA im Zusammenhang mit den zu berücksichtigenden Bedingungen des Basis-Systems von einer zu hohen Anzahl benötigter Testfälle aus und terminiert nicht immer. Aus diesem

#### *4. Diskussion*

Grund erfordert die Anwendung von CASA die vorherige Ausführung von mindestens einem der beiden Algorithmen IPOG oder PICT, um sinnvolle Werte für die obere und untere Schranke des CASA-Algorithmus angeben zu können.

Auch die Anwenderfreundlichkeit von CASA bleibt hinter jener der beiden Tools ACTS und PICT zurück: Die Übersetzung der verschiedenen Parameter, deren Ausprägungen und Bedingungen in die interne Logik beziehungsweise Syntax von CASA fällt wesentlich komplexer aus als jene der Tools ACTS und PICT. CASA und PICT werden beide über die Kommandozeile gestartet und erfordern daher gewisse Grundkenntnisse im Umgang mit den Befehlen der Konsole. ACTS ermöglicht im Gegensatz dazu die einfachste Bedienmöglichkeit via Benutzeroberfläche, welche sich unter anderem dank einer ausführlichen Anleitung leicht nutzen lässt. Darüber hinaus besitzt ACTS zusätzlich die Möglichkeit über die Kommandozeile oder Java-API ausgeführt zu werden, was die vielfältige Nutzbarkeit des Tools unterstreicht.

Anhand der verschiedenen Ansätze zur Beantwortung der allgemeinen Fragestellung dieser Arbeit und der Teilfragestellungen ergeben sich einige Limitierungen dieser Arbeit und mögliche Forschungsthemen für die Zukunft, die im Folgenden ausgeführt werden.

Im Allgemeinen beschränken sich die Ergebnisse dieser Arbeit auf einen einzigen Anwendungsfall mit einer reduzierten Anzahl an Parametern des verwendeten Testobjekts. Daher würde eine Ausweitung der verschiedenen Ansätze auf weitere Testobjekte beziehungsweise die Berücksichtigung zusätzlicher Parameter zur Generalisierung der Erkenntnisse beitragen.

Neben dieser grundlegenden Einschränkung der Arbeit existieren in Bezug auf die einzelnen Teilfragestellungen dieser Arbeit weitere, detaillierte Limitierungen und Erweiterungen:

- In Bezug auf den Vergleich zwischen Combinatorial Testing und Random Testing könnte insbesondere der Ansatz des zufallsbasierten Testens durch ein realitätsnahe Erweiterung ergänzt werden. In der praktischen Anwendung würde in den meisten Fällen keine derartig naive zufallsbasierte Testmethode zum Einsatz kommen, sondern vielmehr Methoden des Adaptive Random Testing (vgl. Abschnitt 2.1.2.3) oder des erfahrungsbasierten Testens (vgl. Abschnitt 2.1.2.4). Dementsprechend wäre eine Gegenüberstellung von Combinatorial Testing mit diesen beiden Ansätzen für zukünftige Untersuchungen denkbar. Im speziellen Fall des erfahrungsbasierten

#### 4. Diskussion

Testens könnte man beispielsweise bestehende Testdaten aus der Praxis im Hinblick auf die kombinatorische Abdeckung überprüfen und im Zuge dessen eine mögliche Ersetzung durch Testfälle des Combinatorial Testing in Betracht ziehen.

- Das Erweiterte System vereint im Vergleich zum Basis-System Veränderungen der Anzahl der Parameter, der Anzahl möglicher Ausprägungen je Parameter und der Anzahl der Bedingungen an das Testsystem: Dies führt dazu, dass sich Veränderungen der Resultate (vgl. Tabelle 3.11) nicht eindeutig auf einen der drei Faktoren zurückführen lassen und der jeweilige Effekt der drei unterschiedlichen Faktoren unklar bleibt. Dies könnte durch eine detailliertere Herangehensweise ausgeräumt werden, indem jeweils zwei der drei Faktoren konstant gehalten werden, während der verbleibende Faktor verändert wird. Auch im Rahmen des Vergleichs der verschiedenen Algorithmen könnte eine derartige Herangehensweise hilfreich sein, um Unterschiede der Algorithmen im Umgang mit den Faktoren Anzahl der Parameter, Anzahl möglicher Ausprägungen je Parameter und Anzahl der Bedingungen herausarbeiten zu können.
- Für die beiden Ansätze des Erweiterten Systems und der  $t$ -fachen Kombinatorik für jede/n Durchführungsweg / Art der Rückdeckung ergibt sich eine wesentliche Limitierung durch die Einschränkung des Testsystems auf wenige Parameter und eine daraus resultierende, reduzierte Kombinatorik. Insbesondere im Zusammenhang mit den Parametern Zuzahlung zu Beginn, Verwendeter Beitrag § 40 b und Verwendeter Beitrag § 3 Nr. 63 und deren Einschränkungen in Bezug auf mögliche Durchführungswege (vgl. Tabelle 3.2) zeigt sich eine limitierte Aussagekraft der Ergebnisse: Beim Ansatz der  $t$ -fachen Kombinatorik für jede/n Durchführungsweg / Art der Rückdeckung wird bereits für  $t \geq 3$  eine vollständige Kombinationsabdeckung für den Durchführungsweg Direktzusage erreicht, weshalb für höhere Interaktionsparameter keine weiteren Testfälle für die Direktzusage hinzukommen (vgl. Tabelle 3.12). In ähnlicher Form lässt sich diese Erkenntnis auch auf die Ansätze des Basis-Systems und des Erweiterten Systems übertragen: Die allgemeine Aussagekraft der Resultate der verschiedenen Teilfragestellung wird dadurch in gewisser Weise eingeschränkt.

## 5. Fazit

Anhand des Anwendungsbeispiels der Beratungsplattform für Lebensversicherungsprodukte der ALH-Gruppe, Easy Web, wurde im Rahmen dieser Arbeit untersucht, inwiefern sich Methoden des Combinatorial Testing zur Erzeugung sinnvoller Testfälle anwenden lässt. Die Resultate zeigen auf, dass dies im Zusammenhang mit verschiedenen Parametern und Bedingungen des Testsystems generell möglich ist. Insbesondere im Vergleich zu naiven Methoden des zufallsbasierten Testens konnte dargelegt werden, dass Combinatorial Testing im Kontext des Anwendungsfalls erhebliche Vorteile bietet und zugleich wesentlich effizienter mit Bedingungen des Testobjekts umgehen kann als alternative, naive Herangehensweisen. Dies entspricht den Erkenntnissen verschiedener, aktueller Forschungsarbeiten.

Zwei untersuchte Teilfragestellungen konnten zudem belegen, dass sich die Anwendung von Combinatorial Testing unter geringfügigen, zusätzlichen Kosten skalieren lässt. Eine detaillierte Betrachtung des Einflusses der wesentlichen Einflussfaktoren auf die Skalierbarkeit der Methoden (Anzahl zu testender Parameter, Anzahl der zugehörigen Ausprägungen, Anzahl der Bedingungen des Testsystems) wurde in dieser Arbeit jedoch nicht vorgenommen und könnte Teil zukünftiger Forschung sein.

Die beiden Ansätze zur Prüfung der Skalierbarkeit dienten zudem als Methode zur Untersuchung der praktischen Nutzbarkeit von Combinatorial Testing: Als wesentliche Erkenntnis dieser Arbeit bleibt in diesem Zusammenhang festzuhalten, dass die adäquate Analyse des Testobjekts und Formulierung der Testparameter, ihrer Ausprägungen und der Bedingungen des Testsystems von essentieller Bedeutung bei der Anwendung von Combinatorial Testing ist und fundierte Kenntnisse über das Testobjekt und das Testverfahren erfordert. Dies konnten andere Forschungsarbeiten ebenfalls aufzeigen.

Darüber hinaus wurden in einer empirischen Analyse verschiedene Algorithmen und Tools zur Erstellung von Testmengen nach dem Prinzip des Combinatorial Testing verglichen: Es konnte dargelegt werden, dass die Algorithmen PICT und IPOG zuverlässig geringe

## *5. Fazit*

Mengen an Testfällen bei kurzer Ausführungszeit erstellen und daher in der praktischen Anwendung gegenüber den anderen untersuchten Algorithmen CASA und IPOG-F zu bevorzugen sind. In Bezug auf die Nutzbarkeit stellte sich das Tool ACTS als nützlichstes Hilfsmittel bei der Erstellung von Testfällen heraus, unter anderem da es als einziges der verglichenen Tools über eine grafische Benutzeroberfläche verfügt.

# A. Anhang

## A.1. Bedingungen Basis-System

- Tarif = 'AROdd'  $\Rightarrow$  Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung'
- Tarif = 'FR20'  $\Rightarrow$  Durchführungsweg = 'Direktversicherung' || Art der Rückdeckung = 'Rückdeckungsversicherung'
- Tarif = 'RVx'  $\Rightarrow$  Durchführungsweg = 'Direktzusage'
- Tarif = 'PK10'  $\Rightarrow$  Durchführungsweg = 'Pensionskasse'
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Tarif = 'PK10' && Art der Rückdeckung = 'keine'
- Art der Rückdeckung = 'Unterstützungskasse'  $\Rightarrow$  Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0 && Zuzahlung zu Beginn = 0
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Art der Rückdeckung = 'keine'
- Durchführungsweg = 'Direktzusage'  $\Rightarrow$  (Art der Rückdeckung = 'Unterstützungskasse' || Art der Rückdeckung = 'Rückdeckungsversicherung')
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung'  $\Rightarrow$  Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0
- Art der Rückdeckung = 'Unterstützungskasse'  $\Rightarrow$  Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0 && Zuzahlung zu Beginn = 0
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Beitrag = 0 || Beitrag = 599 || Beitrag = 600
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Beitrag = 0 || Beitrag = 299 || Beitrag = 599 || Beitrag = 600
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && (Tarif = 'AREven' || Tarif = 'HR20')  $\Rightarrow$  Beitrag = 0 || Beitrag = 599 || Beitrag = 600 || Beitrag = 125001

## A. Anhang

- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && Tarif = 'FR20' ⇒ Beitrag = 0 || Beitrag = 299 || Beitrag = 600 || Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && Tarif = 'RVx' ⇒ Beitrag = 0 || Beitrag = 299 || Beitrag = 24 || Beitrag = 600 || Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && (Tarif = 'AREven' || Tarif = 'HR20' || Tarif = 'AROdd') ⇒ Beitrag = 0 || Beitrag = 599 || Beitrag = 600 || Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && Tarif = 'RVx' ⇒ Beitrag = 0 || Beitrag = 299 || Beitrag = 5000 || Beitrag = 125001
- Durchführungsweg = 'Direktversicherung' ⇒ Zuzahlung zu Beginn = 0 || Zuzahlung zu Beginn = 99 || Zuzahlung zu Beginn = 2000
- Durchführungsweg = 'Pensionskasse' ⇒ Zuzahlung zu Beginn = 0 || Zuzahlung zu Beginn = 299 || Zuzahlung zu Beginn = 2000
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && (Tarif = 'AREven' || Tarif = 'HR20' || Tarif = 'AROdd') ⇒ Zuzahlung zu Beginn= 0 || Zuzahlung zu Beginn= 2000 || Zuzahlung zu Beginn= 1000001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && Tarif = 'RVx' ⇒ Zuzahlung zu Beginn= 0 || Zuzahlung zu Beginn= 2000 || Zuzahlung zu Beginn= 299 || Zuzahlung zu Beginn= 5001
- Durchführungsweg = 'Direktversicherung' ⇒ Verw. Beitrag § 40 b= 0 || Verw. Beitrag § 40 b= 2148 || Verw. Beitrag § 40 b= 2149
- Durchführungsweg = 'Pensionskasse' ⇒ Verw. Beitrag § 40 b= 0 || Verw. Beitrag § 40 b= 2148 || Verw. Beitrag § 40 b= 2149
- Durchführungsweg = 'Direktversicherung' ⇒ Verw. Beitrag § 3 Nr.63= 0 || Verw. Beitrag § 3 Nr.63= 2068 || Verw. Beitrag § 3 Nr.63= 4068
- Durchführungsweg = 'Pensionskasse' ⇒ Verw. Beitrag § 3 Nr.63= 0 || Verw. Beitrag § 3 Nr.63= 2068 || Verw. Beitrag § 3 Nr.63= 4068

## A.2. Bedingungen Erweitertes System

- Tarif = 'AROdd' ⇒ Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung'

## A. Anhang

- Tarif = 'FR20'  $\Rightarrow$  Durchführungsweg = 'Direktversicherung' || Art der Rückdeckung = 'Rückdeckungsversicherung'
- Tarif = 'RVx'  $\Rightarrow$  Durchführungsweg = 'Direktzusage'
- Tarif = 'PK10'  $\Rightarrow$  Durchführungsweg = 'Pensionskasse'
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Tarif = 'PK10' && Art der Rückdeckung = 'keine'
- Art der Rückdeckung = 'Unterstützungskasse'  $\Rightarrow$  Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0 && Zuzahlung zu Beginn = 0
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Art der Rückdeckung = 'keine'
- Durchführungsweg = 'Direktzusage'  $\Rightarrow$  (Art der Rückdeckung = 'Unterstützungskasse' || Art der Rückdeckung = 'Rückdeckungsversicherung')
- Finanzierungsart = 'AG-finanziert'  $\Rightarrow$  AN-Beitrag = 0
- Finanzierungsart = 'AN-finanziert'  $\Rightarrow$  AG-Beitrag = 0
- Art der Rückdeckung = 'Unterstützungskasse'  $\Rightarrow$  Finanzierungsart != 'Mischfinanziert' && Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0 && Zuzahlung zu Beginn = 0
- Art der Rückdeckung = 'Rückdeckungsversicherung'  $\Rightarrow$  Finanzierungsart = 'AG-finanziert' && Verw. Beitrag § 40 b = 0 && Verw. Beitrag § 3 Nr.63 = 0
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  AN-Beitrag = 0 || AN-Beitrag = 599 || AN-Beitrag = 600 || AN-Beitrag = 1500
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  AN-Beitrag = 0 || AN-Beitrag = 299 || AN-Beitrag = 599 || AN-Beitrag = 600 || AN-Beitrag = 1500
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && (Tarif = 'AREven' || Tarif = 'HR20')  $\Rightarrow$  AN-Beitrag = 0 || AN-Beitrag = 599 || AN-Beitrag = 600 || AN-Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && Tarif = 'RVx'  $\Rightarrow$  AN-Beitrag = 0 || AN-Beitrag = 299 || AN-Beitrag = 1500 || AN-Beitrag = 125001 || AG-Beitrag = 24
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 599 || AG-Beitrag = 600 || AG-Beitrag = 1500
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 299 || AG-Beitrag = 599 || AG-Beitrag = 600 || AG-Beitrag = 1500

## A. Anhang

- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && (Tarif = 'AREven' || Tarif = 'HR20')  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 599 || AG-Beitrag = 600 || AG-Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && Tarif = 'FR20'  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 299 || AG-Beitrag = 600 || AG-Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Unterstützungskasse' && Tarif = 'RVx'  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 299 || AG-Beitrag = 1500 || AG-Beitrag = 125001 || AG-Beitrag = 24
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && (Tarif = 'AREven' || Tarif = 'HR20' || Tarif = 'AROdd')  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 599 || AG-Beitrag = 600 || AG-Beitrag = 125001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && Tarif = 'RVx'  $\Rightarrow$  AG-Beitrag = 0 || AG-Beitrag = 299 || AG-Beitrag = 1500 || AG-Beitrag = 125001
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Zuzahlung zu Beginn = 0 || Zuzahlung zu Beginn = 99 || Zuzahlung zu Beginn = 2000
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Zuzahlung zu Beginn = 0 || Zuzahlung zu Beginn = 299 || Zuzahlung zu Beginn = 2000
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && (Tarif = 'AREven' || Tarif = 'HR20' || Tarif = 'AROdd')  $\Rightarrow$  Zuzahlung zu Beginn= 0 || Zuzahlung zu Beginn= 2000 || Zuzahlung zu Beginn= 1000001
- Durchführungsweg = 'Direktzusage' && Art der Rückdeckung = 'Rückdeckungsversicherung' && Tarif = 'RVx'  $\Rightarrow$  Zuzahlung zu Beginn= 0 || Zuzahlung zu Beginn= 299 || Zuzahlung zu Beginn= 2000 || Zuzahlung zu Beginn= 5001
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Verw. Beitrag § 40 b= 0 || Verw. Beitrag § 40 b= 2148 || Verw. Beitrag § 40 b= 2149
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Verw. Beitrag § 40 b= 0 || Verw. Beitrag § 40 b= 2148 || Verw. Beitrag § 40 b= 2149
- Durchführungsweg = 'Direktversicherung'  $\Rightarrow$  Verw. Beitrag § 3 Nr.63= 0 || Verw. Beitrag § 3 Nr.63= 2068 || Verw. Beitrag § 3 Nr.63= 4068
- Durchführungsweg = 'Pensionskasse'  $\Rightarrow$  Verw. Beitrag § 3 Nr.63= 0 || Verw. Beitrag § 3 Nr.63= 2068 || Verw. Beitrag § 3 Nr.63= 4068

## A. Anhang

### A.3. Mögliche Werte Erweitertes System

Durchführungs-weg / Art der Rückdeckung	Finanz-ierung	Tarife	AN-Beitrag	AG-Beitrag	Zuzahlung zu Beginn	Verw. Beitrag § 40 b	Verw. Beitrag § 3(63)
Direkt-versicherung	AN, AG, Misch	AREven, HR20, FR20	0, 599, 600, 1.500	0, 599, 600, 1.500	0, 99, 2.000	0, 2.148, 2.149	0, 2.068, 4.068
Pensionskasse	AN, AG, Misch	PK10	0, 299, 599, 600, 1.500	0, 299, 599, 600, 1.500	0, 299, 2000	0, 2.148, 2.149	0, 2.068, 4.068
Direktzusage / Unterstützungs-kasse	AN, AG	AREven, HR20	0, 599, 600, 125.001	0, 599, 600, 125.001	–		
		FR20	0, 299, 600, 125.001	0, 299, 600, 125.001	–		
		RVx	0, 24, 299, 1.500, 125.001	0, 24, 299, 1.500, 125.001	–		
Direktzusage / Rückdeckungs-kasse	AG	ARx, HR20	–	0, 599, 600, 125.001	0, 2.000, 1.000.001	–	
		RVx		0, 299, 1.500, 125.001	0, 299, 2.000, 5.001	–	

Tabelle A.1.: Mögliche Werte für die verschiedenen Beitragsparameter für das Erweiterte System in Abhängigkeit des Durchführungswegs, der Art der Rückdeckung und des Tarifs.

# Literaturverzeichnis

- [ABC<sup>+</sup>13] ANAND, Saswat ; BURKE, Edmund K. ; CHEN, Tsong Y. ; CLARK, John ; COHEN, Myra B. ; GRIESKAMP, Wolfgang ; HARMAN, Mark ; HARROLD, Mary J. ; MCMINN, Phil ; BERTOLINO, Antonia u. a.: An orchestrated survey of methodologies for automated software test case generation. In: *Journal of Systems and Software* 86 (2013), Nr. 8, S. 1978–2001
- [ALHa] ALH GRUPPE: *Betriebliche Altersvorsorge*. <https://www.alte-leipziger.de/geschaeftskunden/betriebliche-altersversorgung>. – Zuletzt aufgerufen: 07.12.2021
- [ALHb] ALH GRUPPE: *E@sy Web Leben*. <https://www.al-h.de/appserver/easyweb/>. – Zuletzt aufgerufen: 06.12.2021
- [ALHc] ALH GRUPPE: *Vermittlerportal der ALH Gruppe*. <https://www.vermittlerportal.de/>. – Zuletzt aufgerufen: 07.12.2021
- [AO08] AMMANN, Paul ; OFFUT, Jeff: *Introduction to software testing*. New York : Cambridge University Press, 2008. – ISBN 9780521880381
- [Bac12] BACH, J: *Allpairs test case generation tool*. <https://www.satisfice.com/download/allpairs>. Version: 2012. – Zuletzt aufgerufen: 26.10.2021
- [BK17] BUTTLER, Andreas ; KELLER, Markus: *Einführung in die betriebliche Altersversorgung*. Karlsruhe : VVW, 2017. – ISBN 9783899529791
- [Bun21] BUNDESMINISTERIUM DER JUSTIZ UND FÜR VERBRAUCHERSCHUTZ: *Verordnung über maßgebende Rechengrößen der Sozialversicherung für 2021 (Sozialversicherungs-Rechengrößenverordnung 2021)*. [https://www.gesetze-im-internet.de/svbezgrv\\_2021/BJNR261200020.html](https://www.gesetze-im-internet.de/svbezgrv_2021/BJNR261200020.html). Version: 2021. – Zuletzt aufgerufen: 07.12.2021

## Literaturverzeichnis

- [CCK99] CHATEAUNEUF, MA ; COLBOURN, Charles J. ; KREHER, Donald L.: Covering arrays of strength three. In: *Designs, Codes and Cryptography* 16 (1999), Nr. 3, S. 235–242
- [CDFP97] COHEN, David M. ; DALAL, Siddhartha R. ; FREDMAN, Michael L. ; PATTON, Gardner C.: The AETG system: An approach to testing based on combinatorial design. In: *IEEE Transactions on Software Engineering* 23 (1997), Nr. 7, S. 437–444
- [CJ02] CRAIG, Rick ; JASKIEL, Stefan: *Systematic software testing*. Boston : Artech House, 2002. – ISBN 9781580535083
- [CLM04] CHEN, Tsong Y. ; LEUNG, Hing ; MAK, IK: Adaptive random testing. In: *Annual Asian Computing Science Conference* Springer, 2004, S. 320–329
- [Czea] CZERWONKA, Jacek: *Microsoft/PICT: Pairwise Independent Combinatorial Tool*. <https://github.com/microsoft/pict>. – Zuletzt aufgerufen: 03.11.2021
- [Czeb] CZERWONKA, Jacek: *Pairwise testing*. <https://www.pairwise.org/>. – Zuletzt aufgerufen: 26.10.2021
- [Cze06] CZERWONKA, Jacek: Pairwise testing in real world. In: *24th Pacific Northwest Software Quality Conference* Bd. 200 Citeseer, 2006
- [DDH72] DAHL, Ole-Johan ; DIJKSTRA, Edsger W. ; HOARE, Charles Antony R.: *Structured programming*. Academic Press Ltd., 1972
- [Dev21] DEVA, Prashant: *Method Size Limit in Java - DZone Java*. <https://dzone.com/articles/method-size-limit-java>. Version: Feb 2021. – Zuletzt aufgerufen: 29.10.2021
- [DKS14] DHADYALLA, Gunwant ; KUMARI, Neelu ; SNELL, Timothy: Combinatorial testing for an automotive hybrid electric vehicle control system: a case study. In: *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops* IEEE, 2014, S. 51–57
- [DL19] DUAN, Feng ; LEI, Jeff: User Guide for ACTS. (2019)

## Literaturverzeichnis

- [FLL<sup>+</sup>08] FORBES, Michael ; LAWRENCE, Jim ; LEI, Yu ; KACKER, Raghu N. ; KUHN, D R.: Refining the in-parameter-order strategy for constructing covering arrays. In: *Journal of Research of the National Institute of Standards and Technology* 113 (2008), Nr. 5, S. 287
- [GCD09] GARVIN, Brady J. ; COHEN, Myra B. ; DWYER, Matthew B.: An improved meta-heuristic search for constrained interaction testing. In: *2009 1st International Symposium on Search Based Software Engineering* IEEE, 2009, S. 13–22
- [GCD11] GARVIN, Brady J. ; COHEN, Myra B. ; DWYER, Matthew B.: Evaluating improvements to a meta-heuristic search for constrained interaction testing. In: *Empirical Software Engineering* 16 (2011), Nr. 1, S. 61–102
- [HWKK15] HAGAR, Jon D. ; WISSINK, Thomas L. ; KUHN, D R. ; KACKER, Raghu N.: Introducing combinatorial testing in a large organization. In: *Computer* 48 (2015), Nr. 4, S. 64–72
- [HXCL12] HUANG, Rubing ; XIE, Xiaodong ; CHEN, Tsong Y. ; LU, Yansheng: Adaptive random test case generation for combinatorial testing. In: *2012 IEEE 36th Annual Computer Software and Applications Conference* IEEE, 2012, S. 52–61
- [Int11] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, International Electrotechnical Commission: Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software quality models / International Organization for Standardization. Geneva, CH, März 2011. – Standard
- [KKL<sup>+</sup>10] KUHN, D R. ; KACKER, Raghu N. ; LEI, Yu u. a.: Practical combinatorial testing. In: *NIST special Publication* 800 (2010), Nr. 142, S. 142
- [KL14] KHALSA, Sunint K. ; LABICHE, Yvan: An orchestrated survey of available algorithms and tools for combinatorial testing. In: *2014 IEEE 25th International Symposium on Software Reliability Engineering* IEEE, 2014, S. 323–334
- [Kra21] KRASNER, Herb: The cost of poor software quality in the US: A 2020 report. In: *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, 2021

## Literaturverzeichnis

- [KWG04] KUHN, D R. ; WALLACE, Dolores R. ; GALLO, Albert M.: Software fault interactions and implications for software testing. In: *IEEE transactions on software engineering* 30 (2004), Nr. 6, S. 418–421
- [LGW<sup>+</sup>16] LI, Xuelin ; GAO, Ruizhi ; WONG, W E. ; YANG, Chunhui ; LI, Dong: Applying combinatorial testing in industrial settings. In: *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)* IEEE, 2016, S. 53–60
- [LKK<sup>+</sup>07] LEI, Yu ; KACKER, Raghu ; KUHN, D R. ; OKUN, Vadim ; LAWRENCE, James: IPOG: A general strategy for t-way software testing. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)* IEEE, 2007, S. 549–556
- [LKK<sup>+</sup>08] LEI, Yu ; KACKER, Raghu ; KUHN, D R. ; OKUN, Vadim ; LAWRENCE, James: IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. In: *Software Testing, Verification and Reliability* 18 (2008), Nr. 3, S. 125–148
- [LL10] LUDEWIG, Jochen ; LICHTER, Horst: *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. Heidelberg : Dpunktverlag, 2010. – ISBN 9783898646628
- [LP16] LANGOHR-PLATO, Uwe: *Betriebliche Altersversorgung*. Köln : ZAP, 2016. – ISBN 9783896558404
- [LT98] LEI, Yu ; TAI, Kuo-Chung: In-parameter-order: A test generation strategy for pairwise testing. In: *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No. 98EX231)* IEEE, 1998, S. 254–261
- [MP13] MEHTA, Manish ; PHILIP, Roji: Applications of combinatorial testing methods for breakthrough results in software testing. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* IEEE, 2013, S. 348–351
- [Ozc17] OZCAN, Murat: Applications of practical combinatorial testing methods at siemens industry inc., building technologies division. In: *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* IEEE, 2017, S. 208–215

## Literaturverzeichnis

- [RKK17] RAUNAK, Mohammad S. ; KUHN, D R. ; KACKER, Raghu: Combinatorial testing of full text search in web applications. In: *2017 IEEE international conference on software quality, reliability and security companion (QRS-c)* IEEE, 2017, S. 100–107
- [Roy87] ROYCE, Winston W.: Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering*, 1987, S. 328–338
- [Sch01] SCHÖNING, Uwe: *Algorithmik*. Heidelberg, Berlin : Spektrum, Akad. Verl, 2001. – ISBN 3827410924
- [Sch12] SCHNEIDER, Kurt: *Abenteuer Softwarequalität : Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*. Heidelberg : Dpunkt.verlag, 2012. – ISBN 9783898647847
- [Sin] SINDLINGER, Johannes Gabriel: *Bachelorarbeit: Kombinatorische Testmethoden zur Analyse aktuarieller Software*. <https://github.com/gsindlinger/Combinatorial-Testing-Use-Case>. – Zuletzt aufgerufen: 07.12.2021
- [SJB<sup>+</sup>19] SMITH, Riley ; JARMAN, Darryl ; BELLOWS, Jared ; KUHN, Richard ; KACKER, Raghu ; SIMOS, Dimitris: Measuring Combinatorial Coverage at Adobe. In: *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* IEEE, 2019, S. 194–197
- [SL19] SPILLNER, Andreas ; LINZ, Tilo: *Basiswissen Softwaretest : Aus- und Weiterbildung zum Certified Tester Foundation Level nach ISTQB-Standard*. Heidelberg : dunkt.verlag, 2019. – ISBN 9783960885016
- [SLS11] SPILLNER, Andreas ; LINZ, Tilo ; SCHAEFER, Hans: *Software Testing Foundations : A Study Guide for the Certified Tester Exam : Foundation Level, ISTQB compliant*. Santa Barbara, CA : Rocky Nook, 2011. – ISBN 9781933952789
- [sol] Choco Solver. <https://choco-solver.org/>. – Zuletzt aufgerufen: 03.11.2021
- [Som12] SOMMERVILLE, Ian: *Software-Engineering*. München, Harlow, u.a. : Pearson, Higher Education, 2012. – ISBN 9783868940992

## *Literaturverzeichnis*

- [Ste99] STEVENS, Brett: *Transversal covers and packings*. University of Toronto, 1999
- [uni] *Unicode® Version 14.0 Character Counts.* [https://www.unicode.org/versions/stats/charcountv14\\_0.html](https://www.unicode.org/versions/stats/charcountv14_0.html). – Zuletzt aufgerufen: 29.10.2021
- [Ver] VERBRAUCHERZENTRALE: *Betriebsrente: Mit dem Arbeitgeber für die Rente sparen.* <https://www.verbraucherzentrale.de/wissen/geld-versicherungen/altersvorsorge/betriebsrente-mit-dem-arbeitgeber-fuer-die-rente-sparen-7675>. – Zuletzt aufgerufen: 07.12.2021
- [WC80] WHITE, Lee J. ; COHEN, Edward I.: A domain strategy for computer program testing. In: *IEEE transactions on software engineering* (1980), Nr. 3, S. 247–257
- [YLKK13] YU, Linbin ; LEI, Yu ; KACKER, Raghu N. ; KUHN, D R.: Acts: A combinatorial test generation tool. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation* IEEE, 2013, S. 370–375

## **Ehrenwörtliche Erklärung**

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

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

Ulm, den 25.03.2020

---

(Unterschrift)