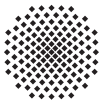
Beiträge zum Stuttgarter Maschinenbau

Karl Kübler

Methodik für eine ganzheitliche Testautomatisierung beim Systemtest von automatisierten Fertigungssystemen

stuttgarter
maschinenbau
interdisziplinär und vielfältig

Universität Stuttgart
Institut für Steuerungstechnik
der Werkzeugmaschinen und
Fertigungseinrichtungen (ISW)



Universität Stuttgart

Beiträge zum Stuttgarter Maschinenbau **Band 9**

Herausgeber: Prof. Dr.-Ing. Oliver Riedel
Prof. Dr.-Ing. Alexander Verl
Jun.-Prof. Dr. rer. nat. Andreas Wortmann

Karl Kübler

Methodik für eine ganzheitliche Testautomatisierung beim Systemtest von automatisierten Fertigungssystemen

Fraunhofer Verlag

Kontaktadresse:
Institut für Steuerungstechnik der Werkzeugmaschinen
und Fertigungseinrichtungen ISW
Seidenstr. 36
70174 Stuttgart
info@isw.uni-stuttgart.de
https://www.isw.uni-stuttgart.de

Titelbild: Karl Kübler

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de abrufbar.

ISSN: 2750-655X
ISBN: 978-3-8396-1817-2

D 93

Zugl.: Stuttgart, Univ., Diss., 2021

Druck und Weiterverarbeitung:
Fraunhofer Verlag, Mediendienstleistungen

Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

© Fraunhofer Verlag, 2022
Nobelstraße 12
70569 Stuttgart
verlag@fraunhofer.de
www.verlag.fraunhofer.de

als rechtlich nicht selbständige Einheit der

Fraunhofer-Gesellschaft zur Förderung
der angewandten Forschung e.V.
Hansastraße 27 c
80686 München
www.fraunhofer.de

Alle Rechte vorbehalten

Dieses Werk ist einschließlich aller seiner Teile urheberrechtlich geschützt. Jede Verwertung, die über die engen Grenzen des Urheberrechtsgesetzes hinausgeht, ist ohne schriftliche Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Speicherung in elektronischen Systemen.
Die Wiedergabe von Warenbezeichnungen und Handelsnamen in diesem Buch berechtigt nicht zu der Annahme, dass solche Bezeichnungen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und deshalb von jedermann benutzt werden dürften.
Soweit in diesem Werk direkt oder indirekt auf Gesetze, Vorschriften oder Richtlinien (z.B. DIN, VDI) Bezug genommen oder aus ihnen zitiert worden ist, kann der Verlag keine Gewähr für Richtigkeit, Vollständigkeit oder Aktualität übernehmen.

Geleitwort

Die deutsche Wirtschaft ist weltweit bekannt für ihren Anlagen- und Maschinenbau. Dabei ist die Universität Stuttgart mit ihren beiden Maschinenbaufakultäten – unter deren Dach sich 42 Institute befinden – die größte universitäre Einrichtung für den Maschinenbau in Deutschland. Unsere wissenschaftliche Exzellenz stützt sich dabei auf unsere zahlreichen Promovierenden und ihre hervorragenden Dissertationen. Viele dieser Dissertationen entstehen in lokaler, nationaler und internationaler Zusammenarbeit mit renommierten Universitäten und außeruniversitären Einrichtungen wie dem Deutschen Zentrum für Luft- und Raumfahrt, der Fraunhofer-Gesellschaft und der Max-Planck-Gesellschaft. Dabei reicht das inhaltliche Spektrum der Dissertationen von Biotechnik, Energietechnik, Fahrzeugtechnik, Kybernetik und Systemtechnik, Produktentwicklung und Konstruktionstechnik, Produktionstechnik bis hin zur Verfahrenstechnik und stützt sich auf die sechs Forschungsschwerpunkte Advanced Systems Engineering, Autonome Produktion, Software-Defined Manufacturing, Resiliente Versorgung, Biointelligenz und Dekarbonisierung der Industrie. Die Ergebnisse aus den Dissertationen zielen darauf ab, kunden-, produkt-, prozess- und mitarbeiterorientierte Technologie zielgerichtet und zeitnah zu entwickeln und anzuwenden.
Viele der im Rahmen der Forschungsarbeiten an den Instituten entstandenen Dissertationen werden in diesen »Beiträgen zum Stuttgarter Maschinenbau« veröffentlicht. Die beiden Fakultäten des Stuttgarter Maschinenbaus wünschen den Promovierenden, dass ihre Dissertationen aus dem Bereich des Maschinenbaus in der breiten Fachwelt als maßgebliche Beiträge wahrgenommen werden und so den Wissensstand auf ein neues Niveau heben.

Für den Stuttgarter Maschinenbau

Stefan Weihe
Prodekan Fakultät 4

Oliver Riedel
Prodekan Fakultät 7

Vorwort der Herausgeber

Innerhalb der Reihe »Beiträge zum Stuttgarter Maschinenbau« berichtet das Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen an der Universität Stuttgart (ISW) über seine Forschungsergebnisse. Das Institut beschäftigt sich in vielfältiger Form mit Steuerungs- und Automatisierungstechnik sowie dem Einsatz von modernen Methoden des Informationsmanagements. Dabei stehen Grundlagenforschung und anwendungsorientierte Entwicklung in einem stetigen Austausch, wodurch ein kontinuierlicher Technologietransfer in die Praxis sichergestellt wird.

Die am ISW entstandenen Dissertationen werden damit unter erweitertem Namen und inzwischen in vierter Generation in der bewährten Konzeption, die der Gründer des ISW Prof. Stute und sein Nachfolger Prof. Pritschow 1972 begonnen haben, durch die heutige Institutsleitung fortgesetzt.

Herrn Dipl.-Ing. Karl Kübler möchten wir für die geleistete Arbeit danken, dem Verlag für die Aufnahme dieser Schriftenreihe in sein Angebot und der Druckerei für die saubere und zügige Ausführung. Möge das Buch von der Fachwelt gut aufgenommen werden.

Alexander Verl

Oliver Riedel

Methodik für eine ganzheitliche Testautomatisierung beim Systemtest von automatisierten Fertigungssystemen

Von der **Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik der Universität Stuttgart**
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

Vorgelegt von

Dipl.-Ing. Karl Simon Johannes Kübler
aus Stuttgart

Hauptberichter: Univ.-Prof. Dr.-Ing. Dr. h.c. mult. Alexander Verl

Mitberichter: Univ.-Prof. Dr.-Ing. Steffen Ihlenfeldt

Tag der mündlichen Prüfung: 15.12.2021

Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart

2022

„Remember that the goal of software testing is to find problems (because we know there will be problems!).“

Myers, Badgett, Sandler 2012 – The art of software testing

Vorwort

Der Weg zu einer Promotion verläuft selten linear, daher möchte ich mich bei folgenden Personen bedanken, die auf verschiedene Weise meinen Weg unterstützt haben.

Zunächst gebührt mein Dank Herr Prof. Verl der meine Promotion als Doktorvater immer wohlwollend unterstützt hat und durch seine Erfahrung immer einen Tipp für das große Ganze parat hatte. Herr Prof. Ihlenfeldt möchte ich für die unkomplizierte Übernahme des Mitberichts danken.

Zum Gelingen dieser Dissertation haben Dr. Armin Lechler, Dr. Jan Schlechtendahl und Dr. Gerhard Krebser durch Korrekturlesen des Manuskripts beigetragen. Vielen Dank für die Anregungen, auf die man irgendwann nicht mehr selbst kommt.

Am Institut möchte ich mich bei meinen Zimmerkollegen Adrian Neyrinck und Florian Jaensch für die gemeinsame Zeit bedanken. Der Gruppe für Virtuelle Methoden in der Produktionstechnik danke ich für die tolle Zusammenarbeit und die erreichten Meilensteine. Den weiteren Kolleginnen und Kollegen des Instituts danke ich für die hilfsbereite und offene Zusammenarbeit, die gegenseitige Motivation und die Möglichkeit auch über die Arbeit hinaus eine tolle Zeit verbracht zu haben. Alexander Blank, Oliver Kabierschke und Elmar Schwarz möchte ich für die Arbeiten als studentische Hilfskräfte danken, die Teile der Realisierung meiner Arbeit unterstützt haben. Den Kollegen der Industriellen Steuerungstechnik GmbH danke ich für die enge Zusammenarbeit und den gewinnbringenden Austausch in mehreren gemeinsamen Projekten, deren Ergebnisse in diese Arbeit eingeflossen sind.

Privat gebührt mein Dank meinen Eltern, die den Weg meiner Ausbildung immer unterstützt haben. Bei meiner Freundin möchte ich mich für das entgegengebrachte Vertrauen, die Unterstützung und die Geduld bedanken.

Mönsheim, 06.03.2022

Inhalt

Vorwort V
Inhalt VII
Abkürzungsverzeichnis XI
Abbildungsverzeichnis XIII
Kurzinhalt XIX
Short Summary XXI
1 Einleitung 1
1.1 Wissenschaftliche Fragestellung der Arbeit 5
1.2 Einordnung und Abgrenzung der wissenschaftlichen Arbeit 6
1.3 Überblick zum Inhalt der nachfolgenden Kapitel 7
2 Grundlagen und Festlegungen für die Arbeit 9
2.1 Grundlagen zur industriellen Steuerungstechnik 9
2.1.1 Architekturen 10
2.1.2 Programmierung 12
2.1.3 Beteiligte Rollen 14
2.2 Hardware-in-the-Loop Systeme und Virtuelle Inbetriebnahme 15
2.2.1 Echtzeit-HiL Simulation 18
2.2.2 Methoden und Modelle der Virtuellen Inbetriebnahme 19
2.3 Grundlagen zum Begriff Fehler 22
2.3.1 Fehlerklassifikation 23
2.3.2 Fehlerentstehung 25
2.3.3 Fehlerkosten 26
2.4 Qualitätsbegriff und Softwaretest 28
2.4.1 Techniken für den Softwaretest 30

2.4.2 Testprozess 33
2.4.3 Testautomatisierung 34
2.4.4 Softwarequalitätsmetriken 36
2.5 Zusammenfassung 37
3 Anforderungen an eine neue Testmethodik 39
3.1 Defizite des aktuellen manuellen Testprozesses 40
3.2 Anforderungen zur Automatisierung 41
3.3 Anforderungen der Entwicklung von Fertigungssystemen 42
3.4 Anforderungen für einen effizienten Testprozess 43
3.5 Zusammenfassung 44
4 Stand der Wissenschaft und Technik und Zielsetzung der Arbeit 47
4.1 Forschungsarbeiten zu Test und Testautomatisierung im Kontext des Maschinen- und Anlagenbaus 47
4.2 Kommerzielle Lösungen für Testwerkzeuge zur Testautomatisierung 56
4.3 Zusammenfassung, Handlungsbedarf und Ziele dieser Arbeit 58
5 Voruntersuchungen zur ganzheitlichen Testautomatisierung und Grobkonzept 61
5.1 Integration von ganzheitlicher Testautomatisierung in das Vorgehensmodell im Maschinen- und Anlagenbau 61
5.2 Auswahl von Testtechniken für die ganzheitliche Testautomatisierung 64
5.3 Auswahl von Typen von Testwerkzeugen 67
5.4 Betrachtung von Diversität von Steuerungsschnittstellen 72
5.5 Entwurf der Architektur eines Werkzeugs für die ganzheitliche Testautomatisierung 74
5.6 Zusammenfassung 78
6 Methodik zur ganzheitlichen Testautomatisierung 81

6.1 Verifikation des Grobkonzepts und Vorgehen bei der Erarbeitung der Methodik 82

6.2 Methodik für die ganzheitliche Testautomatisierung 84

6.2.1 Fehlerbasierte Erweiterung des Simulationsmodells 84

6.2.2 Entwurf der Testtreiber für das RT-HiL System 89

6.2.3 Entwicklung eines Komparators für den automatisierten Testablauf 91

6.2.4 Erreichen einer Herstellerunabhängigkeit für Steuerungssysteme 93

6.2.5 Virtueller Bediener zur Automatisierung der Mensch-Maschine-Schnittstelle 98

6.2.6 Entwicklung einer Sprache für die Erstellung von Testfällen 102

6.2.7 Wiederverwendung von Testfällen durch Modularisierung 110

6.3 Zusammenfassung 116

7 Implementierung und praktische Validierung 121

7.1 Entwicklung eines eigenen Testwerkzeugs 122

7.2 Realisierung mit einem bestehenden Testwerkzeug 124

7.3 Validierung der ganzheitlichen Testautomatisierung und Diskussion zu Vorteilen und Grenzen der Methodik 127

7.3.1 Praktische Validierungen durch Anwendung auf ein Testszenario 128

7.3.2 Einsatz von ganzheitlicher Testautomatisierung zum Systemtest bei Messzyklen einer NC-basierten Steuerung 133

7.3.3 Vorteile und Grenzen einer ganzheitlichen Testautomatisierung 138

7.4 Zusammenfassung 141

8 Zusammenfassung und Ausblick 143

Anhang 147

Literaturverzeichnis 149

Abkürzungsverzeichnis

CAST ..... Computer-Aided Software Testing

CNC ..... Computerized Numerical Control

FOCAS ..... FANUC Open CNC API Specifications

GUI ..... Graphical User Interface

HiL ..... Hardware-in-the-Loop

HMI ..... Human Machine Interface

IBN ..... Inbetriebnahme

IP ..... Internet Protocol

KDT ..... Keyword-Driven Testing

MC ..... Motion Control

MiL ..... Model-in-the-Loop

NC ..... Numerical Control

OEM ..... Original Equipment Manufacturer

OPC UA ..... Open Platform Communications Unified Architecture

PC ..... Personal Computer

QM ..... Qualitätsmanagement

QS ..... Qualitätssicherung

RC ..... Robot Control

RT-HiL ..... Real-Time Hardware-in-the-Loop

SADT ..... Structured Analysis and Design Technique

SiL ..... Software-in-the-Loop

SPS ..... Speicherprogrammierbare Steuerung

SuT ..... System-under-Test

TCP ..... Transfer Communication Protocol

TDW.....Testdurchführungswerkzeug

TMW .....Testmanagementwerkzeug

TSW..... Testspezifikationswerkzeug

VDE .....Verband der Elektrotechnik Elektronik Informationstechnik e.V.

VDI.....Verein Deutscher Ingenieure e.V.

VIBN ..... Virtuelle Inbetriebnahme

ViST ..... Virtuelle Steuerungstestbench

ViTau .....Virtuelle Testautomatisierung für CNC-Systeme

VNC ..... Virtual Network Computing

XML .....Extensible Markup Language

Abbildungsverzeichnis

Abbildung 1 Erhebungen zum Produktionswert des Maschinenbaus (links) und einzelner Sektoren (rechts) im deutschen Maschinenbau (Statistisches Bundesamt et al. 2021; Statistisches Bundesamt 2021) 1

Abbildung 2 Verschiebung der Kerndisziplinen im Maschinen- und Anlagenbau (Eigner et al. 2014, S. 43) 2

Abbildung 3 Übersicht und Zusammenhang der Begriffs- und Grundlagendefinitionen in dieser Arbeit 9

Abbildung 4 Die gängigsten Architekturen von kommerziellen Steuerungssystemen und Antriebsperipherie (Laible 2005, S. 20f) 11

Abbildung 5 Irrtümer, welche bei der Programmierung von Fertigungssystemen auftreten können, nach (Spath et al. 2000, Kormann et al. 2012b) 13

Abbildung 6 Beteiligte Rollen und Ablauf einer Auftragsfertigung von der Bestellung bis zur Entwicklung des Fertigungssystems. 14

Abbildung 7 Die Virtuelle Inbetriebnahme eingebettet im Lebenszyklus eines Fertigungssystems. (Eigene Darstellung, mit Angaben aus (Royce 1970; Bender 1999, S. 6f; Zäh et al. 2006; Verband Deutscher Maschinen- und Anlagenbau e.V. 2020) 16

Abbildung 8 Chronologische Übersicht zu Forschungsarbeiten und kommerziellen Produkten im Bereich der HiL Simulation für Fertigungssysteme 18

Abbildung 9 Echtzeit-Hardware-in-the-Loop nach (Pritschow et al. 2004) 19

Abbildung 10 Sinngemäße Darstellung der Konkretisierung des Steuerungssystems in verschiedenen virtuellen Testumgebungen (Weiterentwicklung von Scheifele et al. 2018) 21

Abbildung 11 Teil einer virtuellen Schalttafel für den Test am HiL System (Quelle: Chiron-Werke GmbH & Co. KG) 22

Abbildung 12 Klassifizierung von Fehlern in Steuerungssystemen nach Art, Zeitpunkt und Ort der Entstehung (nach Laible 2005, S. 54) 24

Abbildung 13 Fehlerentstehung, -erkennung und -kosten in Softwareprojekten (Liggesmeyer 2009, S. 33) 26

Abbildung 14 Verteilung von Softwareausfallraten und Softwarekosten in fehlertoleranten Systemen (rechte Seite nach Oestreicher 1986, S. 112; Montenegro 2000) 27

Abbildung 15 Aufteilung von Softwarekosten (aus Frühauf et al. 2007, S. 18) 28

Abbildung 16 V-Modell zur Softwareentwicklung (Bröhl et al. 1995, S. 24; Dornseiff et al. 2001, S. 351; Frühauf et al. 2007, S. 19; Balzert 2008, S. 554; Spillner et al. 2012, S. 42; Schaefer 2012) 31

Abbildung 17 Testprozess nach ISTQB-Lehrplan (Spillner et al. 2014, S. 14) erweitert um Artefakte (blau) 34

Abbildung 18 Vor- und Nachteile von Testautomatisierung (Galin 2004, S. 242–244) 35

Abbildung 19 Zusammenfassung und Clusterung der Anforderungen 45

Abbildung 20 Schnellübersicht der Bewertung mittels Harvey Balls von Forschungsarbeiten gegen die Anforderung 56

Abbildung 21 Einordnung der bekannten Forschungsarbeiten zwischen Aufwand und Aussagekraft der Testmethodik; Identifikation der Forschungslücke 59

Abbildung 22 Vergleich beim manuellen Vorgehen in der Testphase mit der Methodik zur ganzheitlichen Testautomatisierung 63

Abbildung 23 Übersicht zu Testtechniken: Testarten, Testmethoden und Testverfahren (eigene Darstellung von Informationen aus (Helfen et al. 2006; Balzert 2008; Liggesmeyer 2009; Myers et al. 2012; Witte 2016)) 65

Abbildung 24 Eignung von Werkzeugtypen für das CAST Werkzeug der ganzheitlichen Testautomatisierung basierend auf den Anforderungen 70

Abbildung 25 Diversität und fehlende Middleware zwischen den Schnittstellen von gängigen Steuerungssystemen eines Fertigungssystems und dem CAST Werkzeug 73

Abbildung 26 Beispielhafte Testfälle von Werkzeugmaschinenherstellern, eigene Erhebung 76

Abbildung 27 Architektur der virtuellen Steuerungstestbench aus CAST Werkzeug und RT-HiL System 77

Abbildung 28 Umfang der neuen Testmethodik, eingeteilt nach digitalen Methoden, Modellen und Werkzeugen 81

Abbildung 29 In Anlehnung an die universelle Testsystemarchitektur für mechatronische Produkte nach (Korotkiy 2010) wurde das Konzept für das Werkzeug für die ganzheitliche Testautomatisierung als blaue Flächen eingezeichnet 83

Abbildung 30 Auflistung und Gegenüberstellung von Versagen innerhalb eines Fertigungssystems aus Literaturquellen 86

Abbildung 31 Auflistung der identifizierten Signaltypen für Fehlermodellierung mit Erläuterung 88

Abbildung 32 Modellerweiterung einer Schwenkbrücke (oben links) eines CNC-Bearbeitungszentrums um fehlerhafte Aktorsignale (unten links) und eine manuelle, virtuelle Bedientafel zum Auslösen (rechts) 89

Abbildung 33 Einbettung und Funktionsprinzipien des Komparators 92

Abbildung 34 Definierte Schlüsselworte auf der Endnutzerebene 95

Abbildung 35 Definierte Schlüsselworte auf der Domänenebene 96

Abbildung 36 Definierte Schlüsselworte auf der Testschnittstellenebene 97

Abbildung 37 Funktionsweise der herstellerunabhängigen Definition und Ausführung von Testfällen an der Steuerungsschnittstelle 98

Abbildung 38 Übersicht zu drei verschiedenen grafischen Elementen (Zeilen) und deren möglichen Zuständen (Spalten) 100

Abbildung 39 Beispielhafte Bedienfolge GuiSelectNcProgram mit allen Schaltflächen und Knöpfen zur Auswahl des NC-Programms „Kreisfahrt“ 101

Abbildung 40 Phasen und Artefakte der ganzheitlichen Test-automatisierung 104

Abbildung 41 Auswahl an grafischen Programmier- und Modellierungssprachen im Umfeld der Fertigungstechnik 105

Abbildung 42 Darstellung verschiedener Abläufe in der Testdurchführung (erweitert nach (Frühauf et al. 2007, S. 58)) 106

Abbildung 43 Modellierung eines Testschritts als Blackbox mit seiner Umgebung 108

Abbildung 44 Zeilen des Testprotokolls für den Testschritt "PauseExecution" 110

Abbildung 45 Prinzip zum Aufbau der Bibliothek: I Basisebene II Testprozessebene III Testfallebene mit horizontalen Anwendungsfeldern, grau: Testarten aus der Entwicklung; hellblau: spezifische Testarten; hellgrün: nach Baukasten 113

Abbildung 46 Gesamtarchitektur und Lösungen für die Methodik der ganzheitlichen Testautomatisierung 118

Abbildung 47 Visualisierung der im Rahmen dieser Arbeit erstellten Demonstratoren als Derivate der Virtuellen Steuerungs-testbench 122

Abbildung 48 Die GUI des entwickelten Testwerkzeugs ViTau mit Erläuterungen zu einzelnen Funktionen 123

Abbildung 49 Aufnahmen des Demonstrators zur ganzheitlichen Testautomatisierung unter Nutzung eines RT-HiL Systems 124

Abbildung 50 Übersicht zu den erstellten Bibliotheken für die Automatisierung mittels expecco 126

Abbildung 51 Ablaufdiagramm für den Testfall "Emergency stop" 129

Abbildung 52 Aufbau der Variante1 (links) und Variante 2 (rechts) des Demonstrators zum direkten Vergleich der CAST Implementierungen; oben: Screenshots der erstellten Testszenarien sowie die von den CAST Werkzeugen generierten Testberichte; unten: RT-HiL System aus HMI, Steuerung und virtuellem Bearbeitungszentrum. 130

Abbildung 53 Testszenario "Emergency stop"; oben: die Reaktion am RT-HiL System, NC HMI zeigt den Fehler "Not-Halt"; mittig: die virtuelle Steuertafel zeigt in den blauen Kreisen an, dass der Nothalt ausgelöst ist, die Freigaben zurückgesetzt wurden und die Antriebe stromlos geschalten wurden; unten: das modellierte Testszenario in den beiden Werkzeugen ViTau und expecco. 131

Abbildung 54 Vergleich der Zeitdauer bei der Durchführung und Auswertung im Testszenario „Emergency stop“ an den drei Testkonfigurationen 132

Abbildung 55 Überblick und Beschreibung der Parameter für den Messzyklus (ISG Industrielle Steuerungstechnik GmbH 2021, S. 110) 135

Abbildung 56 Screenshot bei laufender Testdurchführung an der virtuellen Testumgebung zum Test von Messzyklus SysMeasWcs8 (mit freundlicher Genehmigung der ISG Steuerungstechnik GmbH) 137

Abbildung 57 Vergleich des manuellen Testprozesses mit ganzheitlicher Testautomatisierung anhand Kriterien entlang der Einführung, Verwendung und Qualitätsverbesserung. Der Text je Kriterium liefert die Begründung für die Bewertung. 139

Abbildung 58 Erster Teil der identifizierten Signalverläufe für fehlerbasierte Erweiterung von Simulationsmodellen 147

Abbildung 59 Zweiter Teil der identifizierten Signalverläufe für fehlerbasierte Erweiterung von Simulationsmodellen 148

Kurzinhalt

Diese Arbeit stellt einen Beitrag zur Qualitätssicherung bei der Entwicklung von automatisierten Fertigungssystemen dar. Im Speziellen wird eine ganzheitliche Testautomatisierung für den steigenden Softwareanteil in Fertigungssystemen motiviert, entwickelt und validiert.

Seit Jahrzehnten wird daran entwickelt und geforscht, den Automatisierungsgrad von Fertigungssystemen zu erhöhen. Im Trend der steigenden und intelligenteren Automatisierung ergibt sich zeitgleich der Effekt von deutlich mehr softwarebasierten Lösungen gegenüber der bisherigen Ausführung in Elektronik oder Mechanik. Der Einsatz von Software bietet einige Vorteile: Schnellere Entwicklungszyklen, kostengünstigere Nachbesserungen am Produkt und bessere Skalierbarkeit beim Ausrollen. Der höheren Dynamik beim Entwickeln und Nachbessern am Softwareanteil steht ein steigender Aufwand zur Qualitätssicherung gegenüber: Komplexere Testabläufe und größere Datenmengen bedeuten einen gestiegenen Testumfang bei gleichzeitig höherer Frequenz an Testdurchführungen. Diese Tendenzen machen eine hohe Testabdeckung und eine hohe Testtiefe mittels der aktuellen (manuellen) Testprozesse unmöglich. Zunächst wird in dieser Arbeit der Testprozess für Fertigungssysteme untersucht und dessen Defizite herausgearbeitet: Die neue Testdynamik führt zu Zeitproblemen, fehlerhaften Ergebnissen aufgrund manueller Ausführung und insgesamt zu einem unkoordinierten Testprozess mit wenig Unterstützung durch Werkzeuge.

In dieser Arbeit wird im Folgenden eine ganzheitliche Testautomatisierung betrachtet, um den Testprozess für Fertigungssysteme in den Testphasen Systemtest, Abnahmetest und Regressionstest zu verbessern. Zielgröße ist die Schnittmenge aus hoher Aussagekraft der Testergebnisse und geringem Aufwand bei der Anwendung der neuen Testmethodik.

Die Lösung ist die Kombination aus Methoden der Softwaretechnik, einer virtuellen Testumgebung wie bei der Virtuellen Inbetriebnahme und der ganzheitlichen Betrachtung des Testprozesses bestehend aus Testentwurf, Testrealisierung, Testdurchführung und Testauswertung. Kernmethoden der Methodik sind die

werkzeugunterstützte, automatische Testdurchführung und -auswertung. Dazu werden bekannte Testwerkzeugtypen analysiert und eine Auswahl getroffen. Als Testumgebung, in Kopplung mit dem Testwerkzeug wird ein Echtzeit-Hardware-in-the-Loop System vorgeschlagen, um eine frühe und risikolose Testumgebung zu ermöglichen, bei gleichzeitig gegebener Übertragbarkeit der Testergebnisse auf das reale Fertigungssystem. Die in der Testumgebung verwendeten Simulationsmodelle zur Abbildung des Fertigungssystems werden um fehlerbasiertes Verhalten ergänzt, um die Testabdeckung zu erhöhen. Für den Testentwurf wird erarbeitet, wie Abläufe für den Test definiert werden müssen und wie die Wiederverwendung den Aufwand im Testentwurf reduziert. Um die Testrealisierung zu vereinfachen, wird mittels schlüsselwortgetriebenem Testen eine herstellerunabhängige Definition von Testfällen erarbeitet. Für die Definition von oberflächenbasierter Automation wird ein modellbasierter Ansatz entwickelt, der die Bedienhandlungen für die automatisierte Testdurchführung generiert.

Anhand zweier Validierungsszenarien werden konkrete Verbesserungen zum bisher bestehenden Testprozess ermittelt. Quantitativ verkürzt die Methodik die Zeitdauer bei der reinen Testdurchführung mit einem ausgereiften Testwerkzeug um ca. 45 %. In einem Systemtestszenario wurde der Gesamtaufwand mit Testentwurf und -abschluss um ca. 95 % gesenkt. Bei der Durchführung wird zudem kein Personal mehr eingebunden. Die Spannweite der Zeitdauer zwischen manueller und automatischer Testdurchführung wird um den Faktor 6 gesenkt. Die Zeitdauer bei der Erstellung von Testberichten konnte um den Faktor 20 gesenkt werden.

Qualitativ konnten weitere Verbesserungen identifiziert werden: Die Rate an ausgeführten Testfällen je Regressionsschleife im Systemtest kann ohne Mehraufwand auf 100 % gesteigert werden. Das ganzheitliche Vorgehen unterstützt die Reproduzierbarkeit der Testergebnisse, da Schritte des Testentwurfes, der Testdurchführung und der Testauswertung immer auf die gleiche Art und Weise durchgeführt werden. Der Aufwand der Verifikation und Validierung im Entwicklungsprozess von Fertigungssystemen wird mittels der ganzheitlichen Testautomatisierung reduziert bzw. entzerrt, sodass mehr Zeit für die Optimierung aufgewandt werden kann.

Short Summary

This work represents a contribution to quality assurance in the development of automated manufacturing systems. Specifically, it motivates, develops and validates holistic test automation for the increasing amount of software in manufacturing systems.

For decades, there has been development and research to improve the level of automation in manufacturing systems. Included in the trend of increasing and more intelligent automation, there is a simultaneous effect for the use of significantly more software-based solutions, compared to the previous design in electronics or mechanics. The use of software offers several advantages: faster development cycles, more cost-effective rework of the product and better scalability when rolling out. The higher dynamics in the development and reworking of the software part are offset by an increasing effort for quality assurance: More complex test sequences and larger data volumes mean an increased scope of testing with a simultaneous higher frequency of test executions. These tendencies make a high test coverage and a high test depth impossible because of the current (manual) test processes. First, this paper examines the test process for manufacturing systems and identifies its shortcomings: The new test dynamics lead to timing problems, erroneous results due to manual execution, and an overall uncoordinated test process with little support from tools.

This thesis, therefore, considers a holistic test automation to improve the test process for manufacturing systems in the phases system test, acceptance test and regression test. The objective is the combination of high informative value and low effort for the new test methodology.

The solution is the combination of software engineering methods, a virtual test environment as in Virtual Commissioning and the holistic view of the test process consisting of test design, test realization, test execution and test evaluation. The core method is a tool-supported, automatic test execution and evaluation. For this purpose, known test tool types are analyzed and a selection is made. As test

environment, in coupling with the test tool, a real-time hardware-in-the-loop system is proposed to enable an early and risk-free test environment, while at the same time providing transferability of the test results to the real manufacturing system. Fault-based behavior is added to the simulation models used in the test environment to represent the manufacturing system for increasing the test coverage. For the test design, it is worked out how procedures for the test have to be defined and how reuse reduces the effort in the test design. In order to simplify the test realization, a vendor-independent definition of test cases is developed by means of keyword-driven testing. For the definition of interface-based automation, a model-based approach is developed that generates the operator actions for automated test execution.

Based on two validation scenarios, concrete improvements to the existing test process are identified. Quantitatively, the methodology reduces the time required for pure test execution by about 45 %, with a mature test tool. In a system test scenario, the total effort for test preparation and follow-up steps was reduced by approx. 95 %. In addition, personnel is no longer involved in the execution. The span of time between manual and automatic test execution is reduced by a factor of 6. The time taken to prepare test reports was reduced by a factor of 20.

Qualitatively, further improvements could be identified: The rate of executed test cases per regression loop in the system test can be increased to 100 %, without additional effort. The holistic approach supports the reproducibility of the test results, since the steps of test preparation, test execution and test logging are always performed in the same way. The effort of verification and validation in the development process of manufacturing systems is reduced or equalized by means of holistic test automation so that more time can be spent on optimization.

1 Einleitung

Der Standort Deutschland besitzt seit jeher eine sehr starke Stellung im Maschinenbau. Bereits seit 1985 kann Deutschland diese Stellung behaupten (Weck et al. 2005, S. 3). Innerhalb des Maschinenbaus ist die Branche für Werkzeugmaschinen, gemessen am Produktionswert, am bedeutendsten, siehe Abbildung 1.

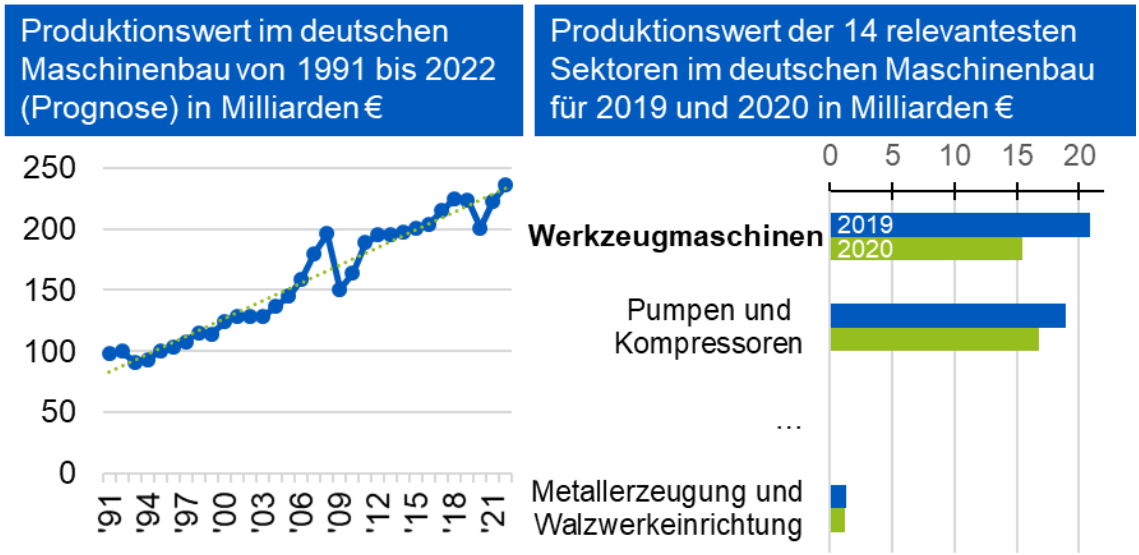


Abbildung 1 Erhebungen zum Produktionswert des Maschinenbaus (links) und einzelner Sektoren (rechts) im deutschen Maschinenbau (Statistisches Bundesamt et al. 2021; Statistisches Bundesamt 2021)

Mit der Digitalisierung in Privat- und Berufsleben halten immer mehr softwarebasierte Systeme Einzug in unsere unmittelbare Umgebung. Nach den Smartphones sind mittlerweile auch die Uhren „smart“ und beinhalten ein softwarebasiertes Betriebssystem (Porteck 2018). Der Maschinen- und Anlagenbau erlebt seit Jahren eine ähnliche Entwicklung, weg von seiner bisher stark mechanischen Prägung. Die ersten Softwaresysteme hielten mit der digitalen Steuerungstechnik Einzug in Maschinen und Anlagen (Pritschow 2006, S. 26–33). Für die Fertigung von Produkten bis heute am wichtigsten sind numerische Steuerungen (NC[1]) und speicherprogrammierbare Steuerungen (SPS). Durch die Ergänzung von

[1] Im Sinne einer besseren Lesbarkeit wird für Begriffe für die eine Abkürzung benutzt wird für die Mehrzahl dieselbe Abkürzung benutzt.

Sensoren und Aktoren spricht man bei den entstehenden automatisierten Fertigungssystemen von mechatronischen Systemen: „Mechatronische Systeme zeichnen sich durch die funktionale und/oder räumliche Integration von Sensoren, Aktoren (auch Aktuatoren genannt), Informationsverarbeitung und einem Grundsystem aus. Das Grundsystem kann aus mechanischen, fluidtechnischen, chemischen oder biologischen Strukturen bestehen.“ (Norm VDI 2206 2004, S. 10). Abbildung 2 visualisiert die Verschiebung hin zu mechatronischen Systemen im Maschinen- und Anlagenbau.

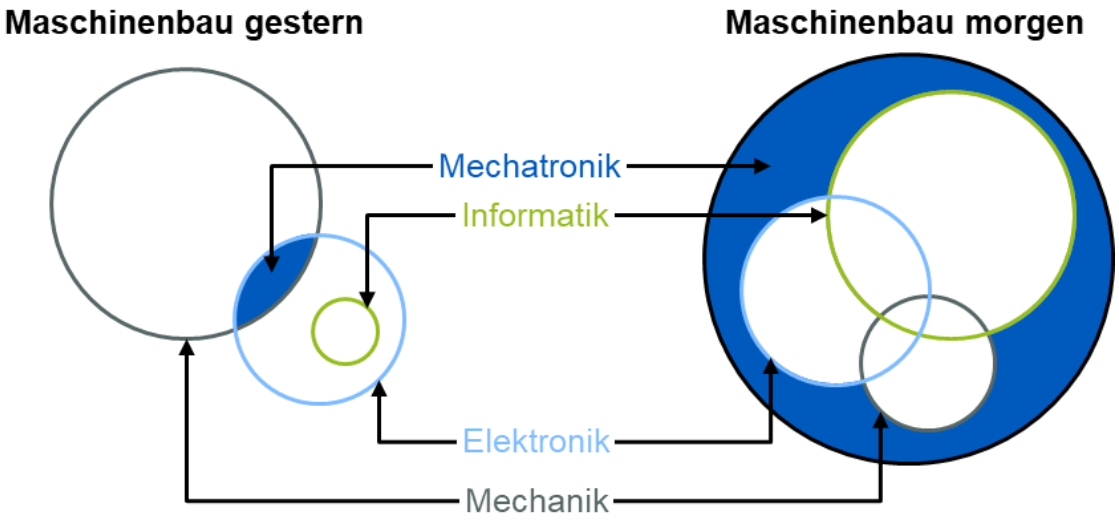


Abbildung 2 Verschiebung der Kerndisziplinen im Maschinen- und Anlagenbau (Eigner et al. 2014, S. 43)

Der Wandel von industriellen Steuerungssystemen hin zu einer Softwareplattform sorgte dafür, dass heute mehr Produktfunktionalitäten der Maschinen und Anlagen in Software realisiert werden (Spiegelberger 2014). Offene Steuerungssysteme bieten insbesondere dem Sondermaschinenbau die Möglichkeit, seine Spezialfunktionen und Algorithmik in Software umzusetzen (Brecher et al. 2010). Neben den Steuerungsalgorithmen werden auch die Peripherie des Steuerungssystems und die zur Anbindung notwendigen Schnittstellen in Software realisiert; beispielhaft seien die folgenden Entwicklungen genannt: Innovative Bedienkonzepte (Herkommer 2015; Business Wire 2016; Sommer 2018), neue Konnektivitätslösungen zwischen Anlagenkomponenten (Keinert et al. 2015; Kretschmer et al. 2016) sowie Systeme zur cloudbasierten Steuerung von Prozessen (Schlechtendahl et al. 2014; Coupek et al. 2016).

Wissenschaftliche Erhebungen zeigen, dass Experten bereits in den früheren Jahren der Software diese gegenüber Hardware bei der Technologieselektion bevorzugt haben (Bauer 1995, Kapitel 5.2.4). Ein Treiber dieser Tendenz dürfte die Möglichkeit der Realisierung von kürzeren Produktlebenszyklen und schnelleren Lieferzeiten, bei gleichzeitiger Erhöhung von Varianten und Artikeln in Unternehmen, sein (Prefi 2003, S. 42), ohne die aus der Mechanik bekannten hohen Investitionskosten. Auch im Maschinen- und Anlagenbau sind Softwarekomponenten einer höheren Änderungsrate ausgesetzt als mechanische Komponenten, müssen aber eine genauso lange Verfügbarkeit gewährleisten wie das Gesamtsystem (Vogel-Heuser et al. 2014). Die Integration der Softwareentwicklung in das gesamtmechatronische Vorgehen bei der Entwicklung und eine Inbetriebnahme aller mechatronischen Disziplinen werden jedoch weiterhin vernachlässigt (Spiegelberger 2014). Eine Studie im Auftrag des Bundesministerium für Bildung und Forschung der Bundesrepublik Deutschland (GfK Marktforschung et al. 2000) zeigt, dass in den sekundären Softwarebranchen, wozu der Maschinenbau gezählt wird, sich die Software zu einem „wettbewerbsentscheidenden Instrument“ zwischen den Anbietern entwickelt.

Auf technischer Ebene entstehen durch die neuen softwarebasierten Funktionalitäten mehr Varietät und mehr Konnektivität innerhalb und zwischen Maschinen und Anlagen. Zum einen vergrößert sich die Anzahl an Subsystemen, zum anderen erhöht sich die Anzahl an Schnittstellen zwischen eben diesen. Eine Studie der Universität Paderborn in Zusammenarbeit mit dem Fraunhofer IPT (Institut für Produktionstechnologie) und der Unity AG bestätigt diesen Zusammenhang (Gausemeier et al. 2013, S. 16f): Bei der Befragung nach den Herausforderungen für den Produktentstehungsprozess von morgen waren die beiden Topnennungen der Unternehmen (mehrfach Nennungen möglich):

- Wachsende Interdisziplinarität (73 %) in der Entwicklung
- Hohe Schnittstellenvielfalt (33 %) in der Entwicklung.

Nach der Definition von (Ulrich et al. 1995) werden Systeme mit hoher Varietät und hoher Konnektivität als *komplexe Systeme* bezeichnet. Damit unterliegen Maschinen und Anlagen in zunehmendem Maße der Betrachtungsweise und

dem Verhalten komplexer Systeme. Auch auf den Entwicklungsprozess der Systeme hat die steigende Komplexität einen Einfluss: Steigt der Grad an Komplexität in einem System, sinken nach (Montenegro 2001) zum einen dessen Sicherheit und Verlässlichkeit, und zum anderen wird die Entwicklung des Systems erschwert. Für sicherheitskritische Maschinen und Anlagen heißt das: Bei steigendem Automatisierungsgrad und steigender Zahl an Subsystemen entsteht eine erhöhte Ausfallrate, bzw. die Kosten zur Sicherstellung der Verfügbarkeit steigen (Seifert 1992, S. 8–10). Für automatisierte Maschinen und Anlagen entfallen nach (Bender 2005, S. 8) zwischen 25 und 75 Prozent der Entwicklungsaufwendungen auf die Softwareentwicklung.

Um die Entwicklung der komplexen Maschinen und Anlagen beherrschbar zu machen, werden neue Vorgehensmodelle à la Systems Engineering (Haberfellner 2015) und Model-Based Systems Engineering (Eigner et al. 2014) erarbeitet und weiterentwickelt (Kaiser 2013; Bursac 2016; Eigner et al. 2017; Pohlmann 2018; Kübler et al. 2020). Weitere Vorgehensmodelle sind ein mechatronisches Engineering durch Variation des aus der Softwareentwicklung bekannten V-Modells (Norm VDI 2206 2004; Bender 2005) mittels agilem Vorgehen (Verband Deutscher Maschinen- und Anlagenbau e.V. 2015) oder mittels baukastenbasiertem Entwickeln (Verband Deutscher Maschinen- und Anlagenbau e.V. 2004; Verband Deutscher Maschinen- und Anlagenbau e.V. 2010; Herrman et al. 2015).

Aktuell werden Maschinen und Anlagen zur Gewährleistung ihrer Funktionsweise, Sicherheit und Verlässlichkeit immer häufiger vor ihrer Auslieferung virtuell in Betrieb genommen. Den Begriff der Virtuellen Inbetriebnahme (VIBN) definieren VDI und VDE in (Norm VDI/VDE 3693 Blatt 1 2016) dabei als „[…] das entwicklungsbegleitende Testen einzelner Komponenten und Teilfunktionen des Automatisierungssystems mithilfe von auf die jeweilige Aufgabenstellung abgestimmten Simulationsmethoden und -modellen […]“. Somit ermöglicht die VIBN die frühe Fehlererkennung in der Software von Maschinen und Anlagen (Verband Deutscher Maschinen- und Anlagenbau e.V. 2020, S. 5). Der VDI/VDE-Fachausschuss hinter der Richtlinie 3693 hat zum Ziel „[…] die VIBN im Bereich der Automatisierungstechnik zu etablieren.“ und trifft damit die Erwartungshaltung der Industrie: Ausgehend von einer Quote von unter 50 % für den Einsatz von VIBN

in 2011 (Linke 2011) geht die Mehrheit der Industrie (fast 60 % der 58 befragten Unternehmen) in 2017 laut einem Report des Verband der Elektrotechnik Elektronik Informationstechnik e.V. (Reimann 2017; Verband Deutscher Maschinen- und Anlagenbau e.V. 2017) mittlerweile von Wachstum oder sogar starkem Wachstum für die VIBN aus. Über die letzten Jahre kann man feststellen, dass in der Industrie vermehrt VIBN eingesetzt wird, in der Hauptsache, um die reale Inbetriebnahmezeit vor Ort beim Kunden drastisch zu reduzieren (Preiml 2013; Freyer 2017; Daniel 2018; Zetzsche 2019). Untersuchungen hierzu zeigen, die Zeit der realen Inbetriebnahme (IBN) verkürzt sich durch Nutzung der VIBN um bis zu 80 % (Zäh et al. 2006; Daniel 2015).

Zusammenfassend kann nach der Einleitung festgestellt werden, dass zur Qualitätssicherung von Maschinen und Anlagen, und dabei insbesondere der Software, vermehrt VIBN für den System- und Abnahmetest genutzt wird. Ein gravierender Nachteil des Testprozesses mit oder ohne VIBN ergibt sich durch die manuelle Ausführung des Prozesses. Die manuelle Ausführung von Tests bindet in der Entwicklung viele Ressourcen (Spiegelberger 2014), stellt meist keinen systematischen Testprozess dar (Kormann et al. 2012) und lässt bei steigendem Testaufwand keine umfassende Aussage mehr über Fehlerfreiheit oder Qualität der untersuchten Software zu (Denkena et al. 2007, S. 124). Die vorliegende Arbeit greift diese Erkenntnis auf und leistet einen Beitrag in Form einer ganzheitlichen Methodik zur Verbesserung des Testprozesses für den Maschinen- und Anlagenbau.

1.1 Wissenschaftliche Fragestellung der Arbeit

Die **Hypothese dieser Arbeit** lautet: Die aktuelle Form der Testprozesse im Maschinen- und Anlagenbau ist nicht ausreichend, um den kommenden Herausforderungen bei der Entwicklung von softwarelastigen, automatisierten Fertigungssystemen[2] zu begegnen.

[2] Im Rahmen dieser Arbeit, siehe Kapitel 2, definiert als der Verbund von einer oder mehreren Werkzeugmaschinen und deren Peripherie.

Daher stellt sich für den weiteren Verlauf dieser Arbeit die **wissenschaftliche Fragestellung**:

> Wie muss eine neue Methodik aussehen, welche den Testprozess von automatisierten Fertigungssystemen verbessert?

Ein optimaler Testprozess zeichnet sich durch ein definiertes Vorgehen, eine kurze Zeitdauer, Korrektheit der Ergebnisse und aufwandsarme Durchführung aus. Die Vorteile einer früher verfügbaren, virtuellen Testumgebung für den Systemtest macht eine genauere Betrachtung von VIBN notwendig.

Der Beantwortung der wissenschaftlichen Fragestellung vorausgehend muss zunächst die Frage beantwortet werden: Wie sieht der aktuelle Testprozess für automatisierte Fertigungssysteme aus und worin liegen dessen genaue Defizite?

1.2 Einordnung und Abgrenzung der wissenschaftlichen Arbeit

Die Adressaten dieser Arbeit sind, neben dem wissenschaftlichen Publikum, Leser aus dem Maschinen- und Anlagenbau, welche Informationen und Impulse für einen verbesserten und effizienteren Testprozess auch unter Einsatz einer Virtuellen Inbetriebnahme suchen.

Der Beitrag kann in das Digital Engineering verortet werden, da Digital Engineering auf die durchgängige Nutzung digitaler Methoden und Werkzeuge über den Lebenszyklus hinweg abzielt, um die Planungsqualität sowie die Prozessbeherrschung zu verbessern (Schumann et al. 2011). Die Arbeit leistet weiterhin einen Beitrag für ein konsequentes Simultaneous Engineering[3] in der Entwicklung von Fertigungssystemen. Die dabei angewandten, adaptierten und erweiterten Methoden sind die dynamische und rechnergestützte Testdurchführung, der systematische Testentwurf und die rechnergestützte Testauswertung. Die Methodik

[3] Nach (Norm VDI 2221 Blatt 1 2019) die parallele oder überlappende Ausführung von Aufgaben bei Produkt- oder Produktionsentwicklung

dieser Arbeit ermöglicht die Steigerung der Qualität von automatisierten Fertigungssystemen, mit Fokus auf deren Automatisierungssoftware. Mittels eines realisierten Demonstrators zeigt die Arbeit die Anwendbarkeit der Methodik.

Außerhalb der Fragestellung dieser Arbeit liegt die automatische Generierung von Testfällen oder Testdaten im Testprozess, dazu passende Arbeiten finden sich bspw. bei (Linder 2008; Kormann 2016; Rösch 2016).

1.3 Überblick zum Inhalt der nachfolgenden Kapitel

In Kapitel 2 werden begriffliche Definitionen und thematische Grundlagen für das Verständnis dieser Arbeit gegeben. Es werden die Themenkomplexe *Industrielle Steuerungstechnik*, *Virtuelle Inbetriebnahme*, der Begriff des *Fehlers*, sowie *Qualitätsbegriff und Softwaretest* behandelt.

Aus den Grundlagen, weiteren Quellen sowie eigenen Untersuchungen werden in Kapitel 3 die Anforderungen an die zu entwerfende neue Methodik für den Testprozess für Software in automatisierten Fertigungssystemen erarbeitet. Explizit wird dafür der Testprozess mittels VIBN betrachtet.

Kapitel 4 legt den aktuellen Stand der Wissenschaft und Technik zu Testautomatisierung und Testwerkzeugen für Automatisierungssoftware dar. Dabei werden relevante wissenschaftliche Arbeiten und am Markt verfügbare Werkzeuge und Lösungen betrachtet, beschrieben und bewertet. Abgeschlossen wird das Kapitel mit der Betrachtung, wie die wissenschaftliche Fragestellung in dieser Arbeit beantwortet werden kann und welche Schritte im Folgenden dafür notwendig sind.

In Kapitel 5 werden Voruntersuchungen und das daraus hergeleitete Grobkonzept für die Lösung beschrieben.

In Kapitel 6 wird der im Rahmen dieser Arbeit durchgeführte Feinentwurf einer ganzheitlichen Testautomatisierung für die Virtuelle Inbetriebnahme ausgearbeitet und vorgestellt.

Das Kapitel 7 befasst sich mit der Implementierung und praktischen Validierung der ganzheitlichen Testautomatisierung.

Kapitel 8 fasst die Inhalte der Arbeit retroperspektiv zusammen. Weiterhin gibt das Kapitel einen Ausblick auf mögliche weiterführende Fragestellungen.

2 Grundlagen und Festlegungen für die Arbeit

Dieses Kapitel verschafft einen Überblick zu den in dieser Arbeit verwendeten Begrifflichkeiten. Dabei werden Begriffe zu den Themenfeldern „Automatisierte Fertigungssysteme“, „Softwaretechnik“ und „Qualitätsmanagement“ definiert und erläutert. Wenn Begriffe aus der Literatur bekannt sind, werden die vorhandenen Definitionen genutzt, andernfalls wird eine eigene Definition gegeben. Abbildung 3 zeigt die im Folgenden relevanten Begriffe und deren Zusammenhang.

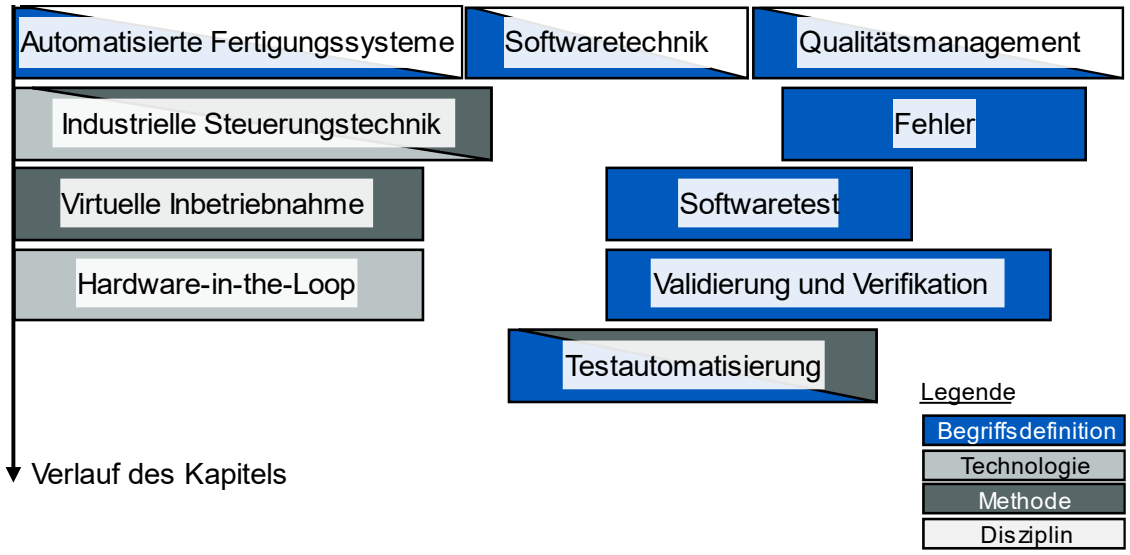


Abbildung 3 Übersicht und Zusammenhang der Begriffs- und Grundlagendefinitionen in dieser Arbeit

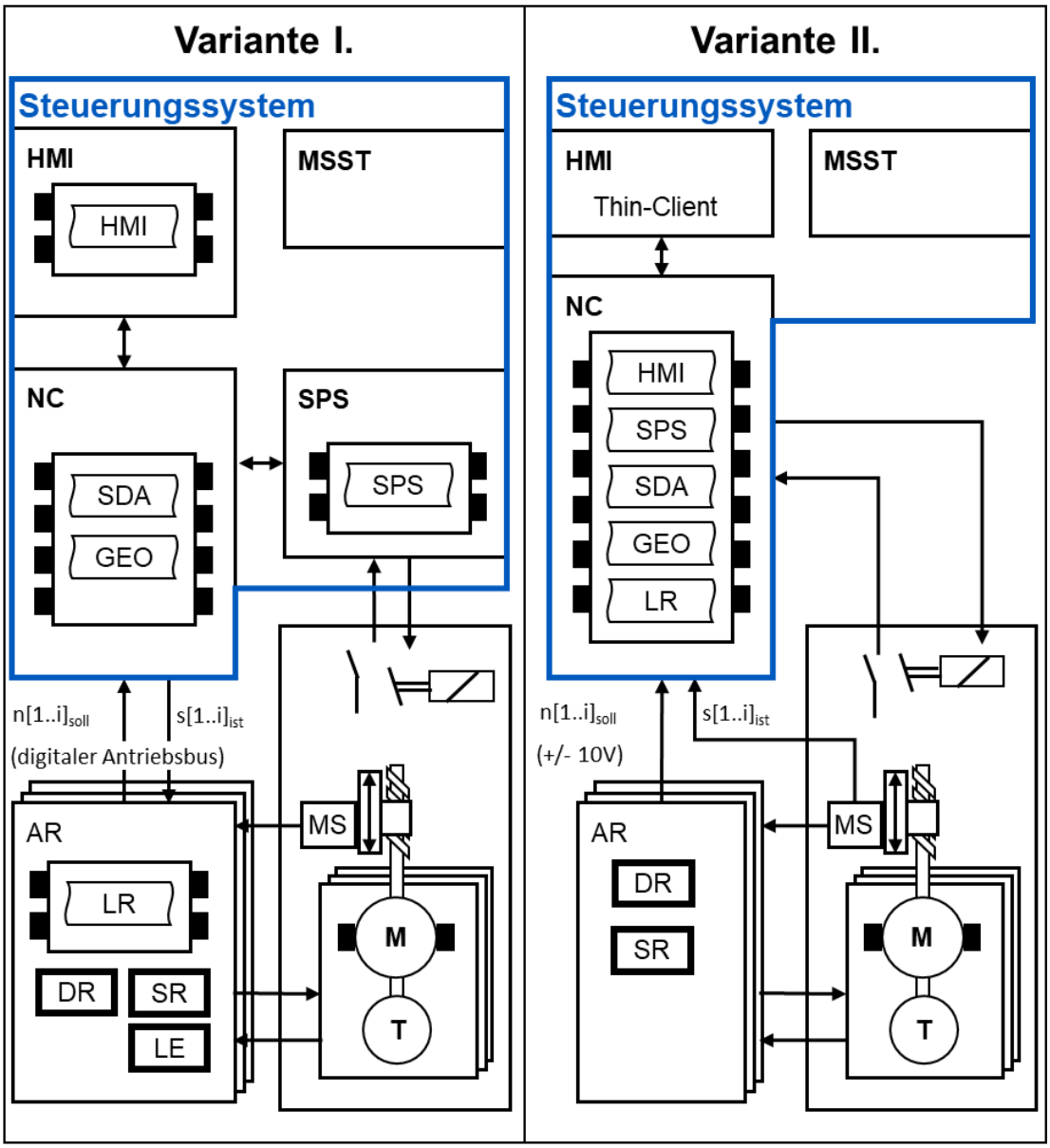
2.1 Grundlagen zur industriellen Steuerungstechnik

Im weiteren Verlauf der Arbeit wird von automatisierten Fertigungssystemen gesprochen. Als Fertigungssystem wird der Verbund von einer oder mehreren Werkzeugmaschinen und deren Peripherie verstanden. Die Peripherie besteht dabei aus Systemen zum Materialtransport, zur Maschinenbeschickung und zum Werkzeughandling. Die Hauptkomponente eines Fertigungssystems bildet die Werkzeugmaschine, welche durch ihr beherrschtes Fertigungsverfahren dem System seine fertigungstechnische Zuordnung gibt, eine Einteilung dazu zeigt (Weck et al. 2005, S. 16f). Der Automatisierungsgrad wird bestimmt durch die Peripherie und teilt die Fertigungssysteme ein in Fertigungsmaschine, Fertigungszentrum, Fertigungszelle, (flexibles) Fertigungssystem und Transferstraße

(Weck et al. 2005, S. 17f). Im weiteren Verlauf der Arbeit werden die verschiedenen Ausbaustufen von automatisierten Fertigungssystemen unter der Bezeichnung *Fertigungssystem* zusammengefasst, von nicht automatisierten Systemen wird nicht ausgegangen.

2.1.1 Architekturen

Die Automatisierung der Fertigungssysteme findet heute durch programmierbare industrielle Steuerungssysteme statt. Je nach Ausbaustufe des Fertigungssystems existieren eine Reihe von unterschiedlichen Steuerungssystemen, die jeweils einem speziellen Zweck dienen: SPS, NC, Robotersteuerungen (RC) und Bewegungssteuerungen (MC). Zur Steuerung von Werkzeugmaschinen in Fertigungssystemen wird ein Verbund aus SPS und NC genutzt (Pritschow 2006, S. 314; Weck et al. 2006, S. 155). Durch die Realisierung der numerischen Steuerungstechnik auf Hardware von Computern spricht man auch von CNC-Steuerungssystemen (Computerized Numerical Control) (Weck et al. 2006, S. 152). Abbildung 4 zeigt die zwei gängigsten Architekturen von Steuerungen mit den weiteren Bestandteilen, Benutzerschnittstelle (HMI) und der Antriebstechnik. Im weiteren Verlauf wird der Verbund aus NC und SPS und Bedienteil (HMI mit Eingabegeräten und Steuertafel) als *Steuerungssystem* bezeichnet. Die HMI besteht aus einer grafischen Benutzeroberfläche (GUI) und Eingabegeräten, ähnlich einer Maus/Mauspad und einer Tastatur an einem Personal Computer (PC) sowie einer Maschinensteuertafel (Pritschow 2006, S. 323f).



Legende

… Prozessor
… Gerät
… Hardwarefunktion
… Softwarefunktion

NC … Numerical Control
SPS … Speicherprogrammierbare Steuerung
HMI … Human Machine Interface
SDA … Steuerdatenverarbeitung
GEO … Geometriedatenverarbeitung
MSST … Maschinensteuertafel

LR … Lageregler
DR … Drehzahlregler
SR … Stromregler
LE … Leistungselektronik
AR … Antriebsregelgerät
M … Antrieb
T … Tacho
MS … Messsystem
n … Drehzahl Soll-/Istwert
s … Position Soll-/Istwert

Abbildung 4 Die gängigsten Architekturen von kommerziellen Steuerungssystemen und Antriebsperipherie (Laible 2005, S. 20f)

Die HMI wird genutzt, um den Status von Alarmen und Meldungen der Steuerungen und des Fertigungssystems einzusehen. Die NC-Programme können per USB-Stick oder Netzwerkverbindung auf die Steuerung übertragen werden und an der HMI mittels der Eingabegeräte verändert werden (Weck et al. 2006, S. 154). Je nach Umfang des Fertigungssystems kann die Werkzeug- und Werkstückverwaltung über die HMI eingesehen und verändert werden. Die HMI läuft, wie in Abbildung 4 gezeigt, entweder als eigenständiger Prozess auf der Hardware des NC-Kerns oder auf einer dedizierten Hardware mit eigenem Prozessor. Die erste Variante wird verwendet, wenn zusätzliche Anwendungen auf der HMI benötigt werden, welche aber aus Ressourcen oder Echtzeitgründen nicht mit der NC zusammenlaufen können. Die zweite Variante vereint alle Softwarekomponenten auf einer Hardware. Sie ist somit die kostengünstigere Variante. Mit dem Einzug leistungsstärkerer Rechnertechnik seit Mitte der 1990er wurden alle Komponenten eines Steuerungssystems nur noch als Softwarekomponenten implementiert und auf einem Hardwaresystem integriert. Man spricht von einer aufgaben-/funktionsorientierten Sichtweise (Pritschow 2006, S. 315). Steuerungssysteme werden heute als *offene Steuerungssysteme* entworfen. (Wälde 2002, S. 15f) und (Pritschow 2006, S. 315–320) beschreiben deren Eigenschaften. Grundlegend sind die Funktionen der Komponenten in Softwaremodule gekapselt. Die Module wiederum kommunizieren über standardisierte Schnittstellen miteinander. Das Grundsystem als Laufzeitumgebung ist eine rechnerbasierte Systemplattform, welche den Modulen einheitlich Dienste zur Verfügung stellt. Dieser Aufbau eines Steuerungssystems ermöglicht es verschiedenen Anbietern, eigene Softwaremodule für die Plattform Steuerungssystem bereitzustellen.

2.1.2 Programmierung

Zur Programmierung von speicherprogrammierbaren Steuerungen definiert (Norm DIN EN 61131-3 2013) mehrere Programmiersprachen. Die fünf Programmiersprachen werden unterteilt in textuelle und grafische Sprachen. Mit der Vielfalt an Sprachen wird der Tatsache Rechnung getragen, dass die Programmierung der SPS nicht von Softwaretechnikern oder Informatikern vorgenommen

wird, sondern von den Technikern aus den Bereichen Mechanik- und Elektronikentwicklung. Erhebungen aus der Industrie zeigen, dass alle Sprachen der Norm Verwendung finden (Vogel-Heuser et al. 2017, S. 45). Für die Programmierung von numerischen Steuerungen definiert die Norm DIN 66025 (Norm DIN 66025-1 1983), wie NC-Sätze aus Worten gebildet werden und welche Wegbedingungen und Zusatzfunktionen mit sogenannten G-bzw. M-Befehlen zur Verfügung stehen (Norm DIN 66025-2 1988). Eigene Zyklen zur Bearbeitung (bspw. Siemens AG 2008) und Vermessung von Werkstücken (bspw. Siemens AG 2013), können vom Steuerungshersteller angeboten werden. Die Nutzung der Zyklen erleichtert es dem Benutzer der Steuerung, wiederkehrende Bearbeitungs- oder Messaufgaben zu parametrieren anstatt zu programmieren.

Abbildung 5 zeigt, welche Irrtümer bei der Programmierung von Steuerungen eines Fertigungssystems gemacht werden. Ziel der Inbetriebnahme und besonders der VIBN ist es, die in den Steuerungsprogrammen begangenen Irrtümer zu finden, um diese vor dem Start der Produktion zu beheben.

Irrtümer bei der Programmierung von Fertigungssystemen
• Falsche oder fehlende Verriegelungen
• Fehlerhafte Prozesse/Fehlende Reaktion in undefinierten Maschinenzuständen
• Fehlerhaftes Setzen/Rücksetzen von Schritten, inkorrekte Sprünge
• Fehlerhaftes Setzen/Rücksetzen von Merkern/Vorzeichen (Negative Logik)
• Mehrfache Belegung gleicher Variablen
• Laufzeitfehler (Zykluszeitüberschreitungen)
• Andere Fehler (z. B. Tippfehler)
• Falsche I/O Adressen
• Fehlende Benachrichtigung auf HMI

Abbildung 5 Irrtümer, welche bei der Programmierung von Fertigungssystemen auftreten können, nach (Spath et al. 2000, Kormann et al. 2012b)

2.1.3 Beteiligte Rollen

Abbildung 6 zeigt die beteiligten Rollen, vom Kunden bis zum Steuerungshersteller, einer Auftragsfertigung und den Ablauf zur Bereitstellung und Nutzung eines Fertigungssystems. Die Abbildung zeigt einen möglichen Zusammenhang unter den Rollen, abweichend kann bspw. der Kunde gleichzeitig Nutzer des Fertigungssystems sein, wenn er für sich selbst fertigt. Weiterhin kann der OEM (Original Equipment Manufacturer – Hersteller des Fertigungssystems) das Steuerungssystem selbst entwickeln und kauft dafür Softwaremodule zu, der Steuerungshersteller entfällt in diesem Fall.

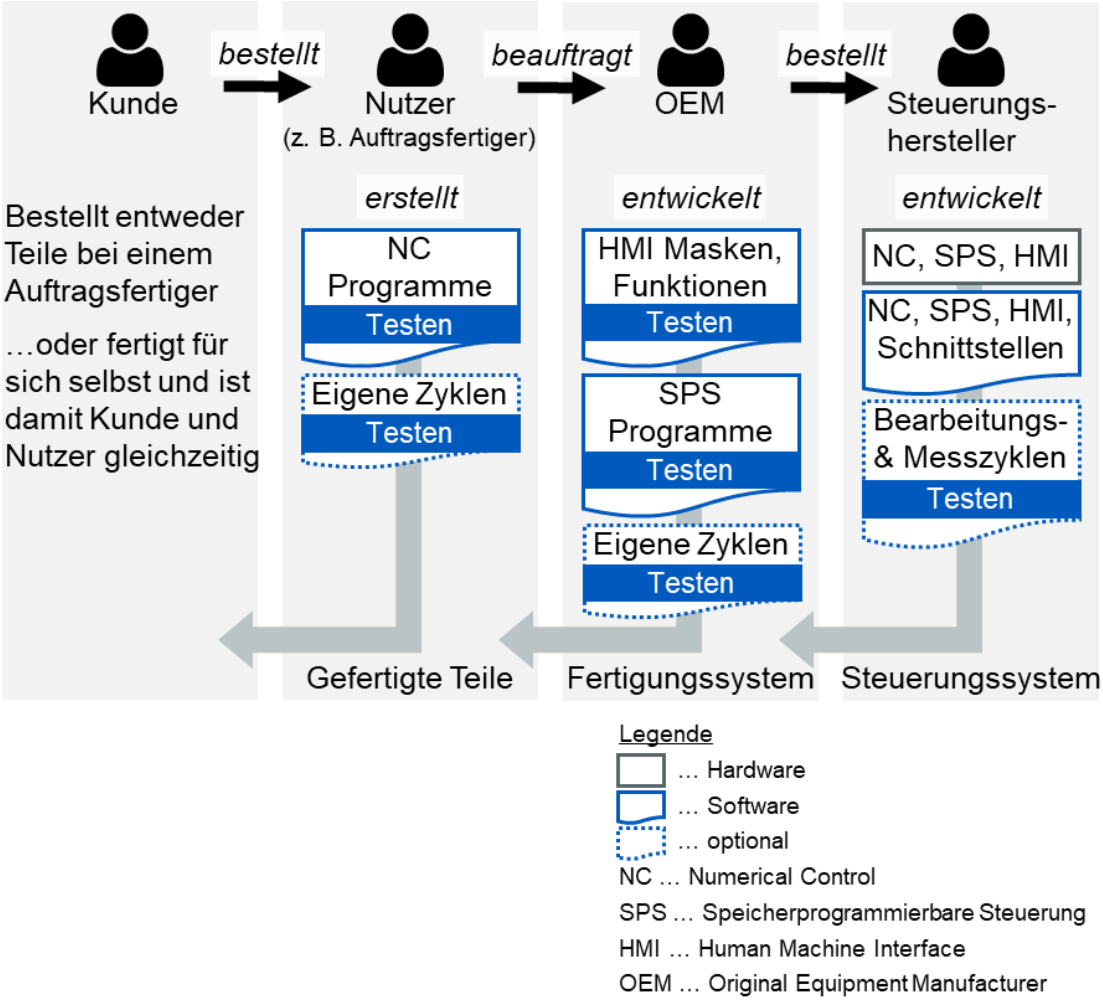


Abbildung 6 Beteiligte Rollen und Ablauf einer Auftragsfertigung von der Bestellung bis zur Entwicklung des Fertigungssystems.

Üblicherweise kauft oder least der Nutzer ein Fertigungssystem vom OEM. Die SPS im Steuerungssystem wird für die jeweiligen Nutzeranforderungen vom

OEM programmiert. Die NC wird vom OEM parametriert und der Nutzer erstellt seine NC-Programme zur Fertigung der Werkstücke. Bei der Programmierung der Konturen und Bewegungsbahnen kann der Kunde auf vorliegende Zyklen zurückgreifen oder eigene erstellen. Für das Vermessen von Werkstücken stehen Messzyklen zur Verfügung. Die vom Steuerungshersteller mitausgelieferte Software wird als *Steuerungssoftware* bezeichnet und kann mit einem Betriebssystem auf einem PC verglichen werden. Die *Steuerungsprogramme* (NC und SPS) zur Automatisierung von Fertigungs-, Handhabungs- oder Montageschritten sind vergleichbar mit Applikationen auf einem PC. In Abbildung 6 ist markiert, an welchen Stellen und von welcher Rolle jeweils Softwaretests durchgeführt werden müssen. Für die neue Testmethodik werden in der vorliegenden Arbeit die Steuerungsprogramme für NC, SPS und die von Nutzer und OEM erstellten Zyklen betrachtet.

2.2 Hardware-in-the-Loop Systeme und Virtuelle Inbetriebnahme

Die VIBN ist eine Simulationsmethode, bei der das Steuerungssystem mit einem Simulationssystem verbunden wird, welches das Verhalten des realen Fertigungssystems nachbildet (Röck 2007, S. 29). Der Zusammenschluss aus Simulation und Steuerung wird als *virtuelle Testumgebung* bezeichnet (Verband Deutscher Maschinen- und Anlagenbau e.V. 2020, S. 10). Der Einsatzzweck für die virtuelle Testumgebung, und damit für die VIBN, ist die Test- und Optimierungsphase am Gesamtsystem. (Kufner 2012, S. 31) definiert folgende Ziele beim Test mittels der VIBN: Aufdecken von Programmierfehlern, Erkennen von Kollisionen, Test der Reaktion der Steuerung im Fehlerfall. Ohne VIBN musste bisher auf die Montage des Fertigungssystems und damit auf die reale IBN gewartet werden. Abbildung 7 zeigt das aktuelle Vorgehen bei der Entwicklung von Fertigungssystemen (z. B. Zetzsche 2019).

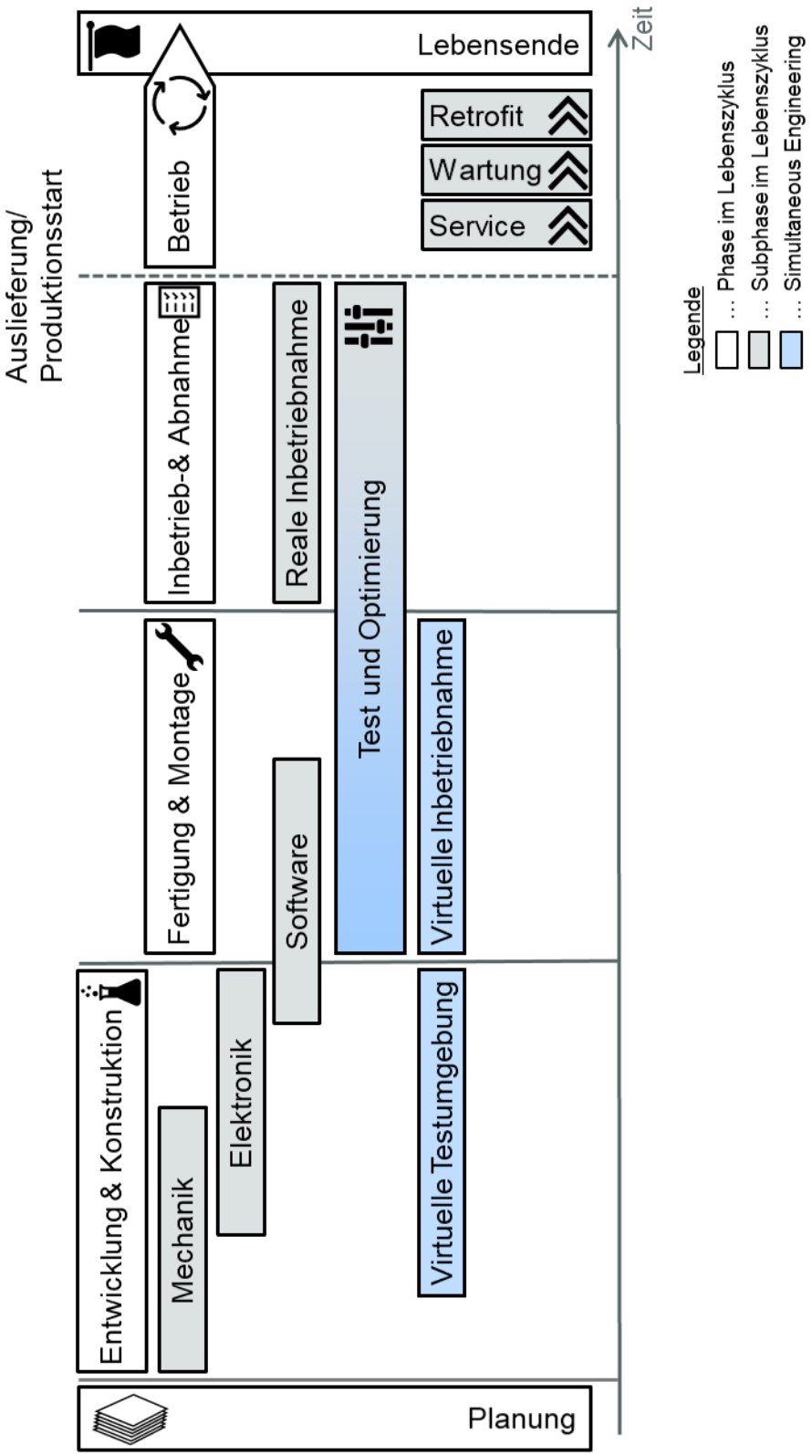


Abbildung 7 Die Virtuelle Inbetriebnahme eingebettet im Lebenszyklus eines Fertigungssystems. (Eigene Darstellung, mit Angaben aus (Royce 1970; Bender 1999, S. 6f; Zäh et al. 2006; Verband Deutscher Maschinen- und Anlagenbau e.V. 2020)

Mittels VIBN kann die Test- und Optimierungsphase früher und parallel, als Simultaneous Engineering, begonnen werden. Nach der VIBN beschränken sich die notwendigen Tätigkeiten bei der realen IBN und Abnahme auf die Prüfung der korrekten Verdrahtung von Sensorik und Aktorik (Seipel 2012), u. U. die Parametrierung der Prozesse, die Fertigung von Testwerkstücken und den Test der Sicherheitskomponenten (Gamblin 2014, S. 153f). Dadurch, dass einige der Tätigkeiten der realen IBN zeitlich nach vorne verlagert werden (sog. *Frontloading*[4]), wird der Zeitpunkt für den Produktionsstart früher erreicht (Zäh et al. 2006).

Zur Vorabinbetriebnahme der ersten SPS Programme wurden sogenannte Schalterkästen beim Testen genutzt (Vollmer 1994). Der Aufbau eines Schalterkastens wurde mit den Ein- und Ausgängen der SPS verbunden. Schalter und Potenziometer wurden entsprechend eines Testvorgangs betätigt, um Eingangssignale für die Steuerung zu erzeugen. Lampen zeigten den Status von Ausgängen an (Weck et al. 2006, S. 135). Komplexere und dynamische Signalverläufe wurden mit Hardwareschaltkreisen erzeugt (Bender 1999, S. 14). Da der Tester bei dieser Methode selbst das (Prozess-)Verhalten der Maschine nachbilden musste, stieß diese Methode schnell an ihre Grenzen, was den Umfang und das korrekte Zeitverhalten anging. Die Lösung des Problems bestand darin, das logische und zeitliche Verhalten durch eine rechnergestützte Simulation zu ermöglichen. (Weck et al. 2006, S. 136)

Bereits in den 1990er Jahren wurde in der Forschung (bspw. in Kohring 1993; Bender 1999) und in der Industrie (Vollmer 1994; Mewes 1996) an der Möglichkeit zur Vorabinbetriebnahme der industriellen Steuerungstechnik mittels Maschinensimulationen gearbeitet. Anschließend an weitere Arbeiten (bspw. Kreusch 2002; Mewes 2005; Zäh et al. 2005; Röck 2007; Wünsch 2008) konnten sich kommerzielle Produkte (Simulationswerkzeuge) am Markt platzieren. Abbildung 8 zeigt eine Übersicht von gängigen Simulationswerkzeugen für die Hardware-in-the-Loop (HiL) Simulation mit Steuerungssystemen und eine Auswahl der wichtigsten Beiträge aus der Forschung.

[4] Duden (2020): „Form der Produktentwicklung, bei der in einer frühen Phase mit digitalen Modellen die wesentlichen Grundstrukturen simuliert, analysiert und festgelegt werden."

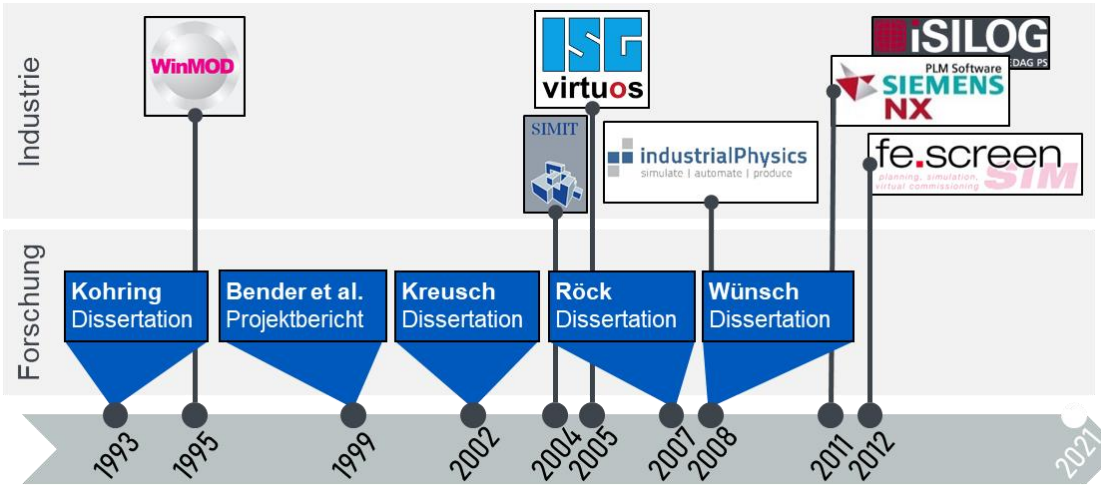
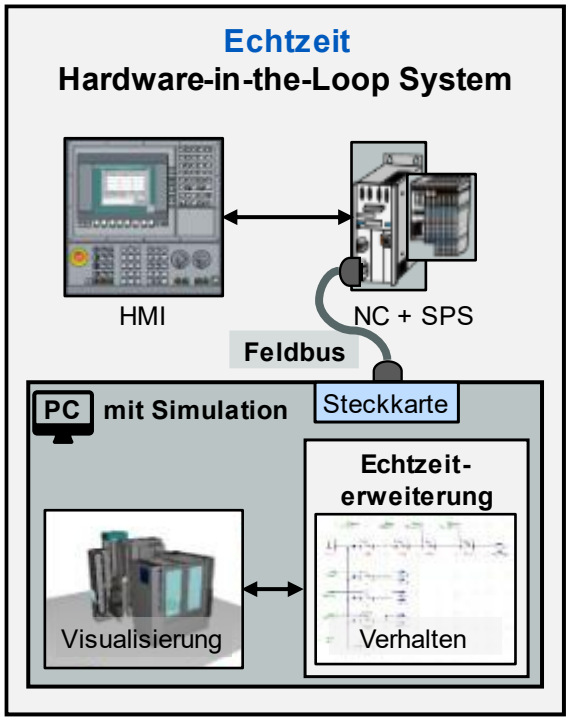


Abbildung 8 Chronologische Übersicht zu Forschungsarbeiten und kommerziellen Produkten im Bereich der HiL Simulation für Fertigungssysteme

2.2.1 Echtzeit-HiL Simulation

Der Ansatz nach (Röck 2007) mündete in dem kommerziellen Simulationswerkzeug *ISG-virtuos* der Fa. *ISG Industrielle Steuerungstechnik GmbH*. Abbildung 9 zeigt schematisch den Aufbau einer RT-HiL nach Röck, erstmals formuliert in (Pritschow et al. 2004). Im oberen Abschnitt der Abbildung ist das Steuerungssystem, bestehend aus HMI, NC und SPS, im unteren Abschnitt die Simulation auf einem PC-System zu sehen. Die Echtzeiterweiterung auf dem PC-System und eine Einsteckkarte mit Bustreiber machen die direkte und verlustfreie Kommunikation über einen Feldbus möglich.



Legende
HMI … Human Machine Interface
NC … Numerical Control
SPS … Speicherprogrammierbare Steuerung

Abbildung 9 Echtzeit-Hardware-in-the-Loop nach (Pritschow et al. 2004)

Dabei ist der Ansatz nach Röck in der Forschung einer von mehreren Echtzeitsimulationsansätzen gewesen, welche Zykluszeiten von < 1ms bei der Modellberechnung ermöglichen (Wünsch 2008, S. 63). In der Industrie ist ISG-virtuos bis heute das einzige Echtzeitsimulationswerkzeug. Da diese Simulationslösung ohne Verlust von Signalen auf das Steuerungssystem reagieren kann, besitzt sie die höchste Aussagekraft und kann ein Fertigungssystem inklusive Antriebs- und Sicherheitstechnik abbilden. Im weiteren Verlauf der Arbeit wird von einem HiL System nach (Röck 2007) als virtuelle Testumgebung ausgegangen. Um dies zu verdeutlichen, wird der Begriff des Real-Time Hardware-in-the-Loop (RT-HiL) Systems eingeführt.

2.2.2 Methoden und Modelle der Virtuellen Inbetriebnahme

Die VIBN wird durch die VDI/VDE-Richtlinien 4499 und 3693 definiert und standardisiert, und durch den VDMA Leitfaden (Verband Deutscher Maschinen- und Anlagenbau e.V. 2020) die Einführung in Unternehmen fokussiert betrachtet. Die VIBN unterstützt mit passenden Modellen, Methoden und Werkzeugen die Inbetriebnahme und wird damit der Digitalen Fabrik zugeordnet (Norm VDI 4499 Blatt

1 2008, S. 2). Die passenden Werkzeuge sind die beschriebenen Simulationswerkzeuge und virtuellen Testumgebungen. Die Modelle sind ausführbare Simulationsmodelle, welche eine Komponenten-, Maschinen- und Prozesssimulation der Fertigungssysteme ermöglichen (Scheifele et al. 2007). Die Modelle werden in drei Ausprägungen von virtuellen Testumgebungen eingesetzt (Norm VDI/VDE 3693 Blatt 1 2016): Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) und HiL. Abbildung 10 zeigt die Unterschiede der drei Ausprägungen bzgl. ihrer Konkretisierung des Steuerungssystems. Die VIBN mittels HiL, bzw. RT-HiL, erreicht den höchsten Konkretisierungsgrad und wird unmittelbar vor der realen IBN eingesetzt. Tests mittels HiL können in der Entwicklung dann begonnen werden, wenn sowohl die realen Steuerungsprogramme, als auch die reale Hardware des Steuerungssystems zur Verfügung stehen. Das Simulationsmodell wird über die drei Ausprägungen hinweg mit unterschiedlicher Genauigkeit genutzt, erreicht aber nie 100 % Abbildungsgenauigkeit. (Lohmeyer 2013, S. 20) bezeichnet dies als Reduktionsdilemma, da die Anwendbarkeit des Modells im Widerspruch zur Ganzheitlichkeit steht.

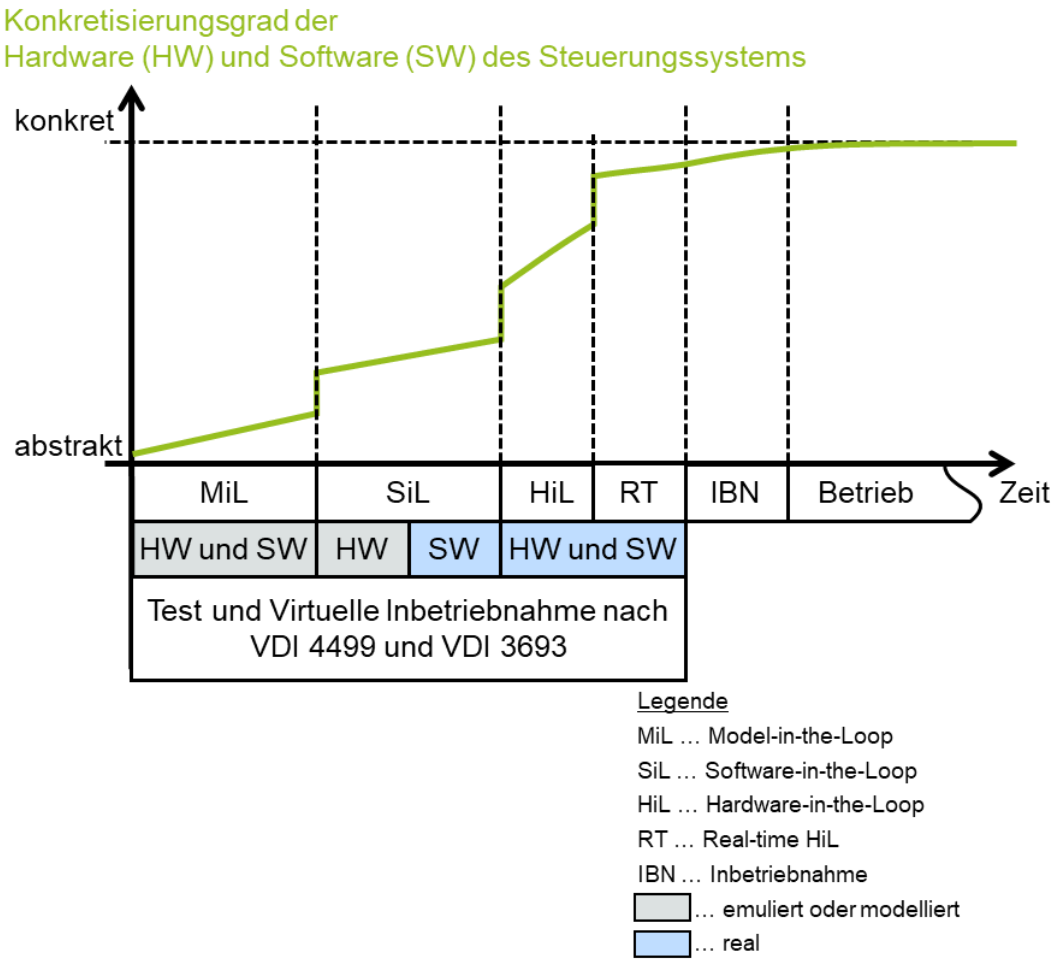


Abbildung 10 Sinngemäße Darstellung der Konkretisierung des Steuerungssystems in verschiedenen virtuellen Testumgebungen (Weiterentwicklung von Scheifele et al. 2018)

(Norm VDI/VDE 3693 Blatt 1 2016, S. 10–14) beschreibt Methoden bei der Testdurchführung an einer virtuellen Testumgebung. Die gängigste Methode, die Signalmanipulation, kommt von den frühen Schalterkästen und wird mittels professioneller Simulationswerkzeuge umgesetzt. Dazu erstellt der Tester im Simulationswerkzeug virtuelle Schaltertafeln. Diese Schaltertafeln bieten eine zentrale Stelle im Simulationsmodell, von wo aus Signale beobachtet, gesteuert oder manipuliert[5] werden können. Abbildung 11 zeigt einen Ausschnitt einer virtuellen Schaltertafel eines Bearbeitungszentrums mit zwei Werkzeugmagazinen. Mittels virtueller Schalter und Knöpfe kann der Tester Werkzeuge manuell entnehmen

[5] Gelegentlich mit „forcen“ bezeichnet (Norm VDI/VDE 3693 Blatt 1 2016), vom Englischen *to force sth.* – etw. erzwingen

und einlegen. Die großen virtuellen LEDs (Light Emitting Diode) zeigen dabei an, ob der aktuelle Platz bereits belegt oder frei zum Einlegen ist.

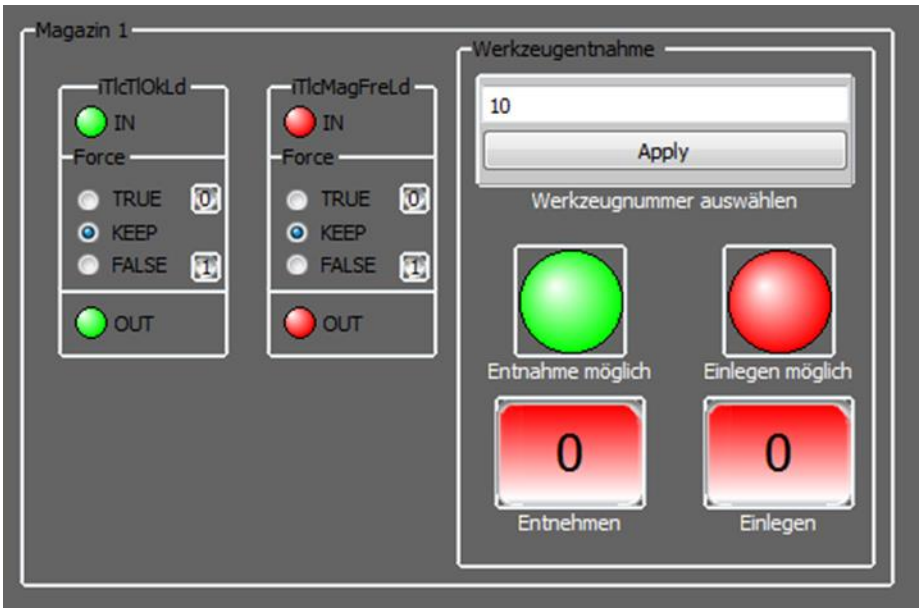


Abbildung 11 Teil einer virtuellen Schalttafel für den Test am HiL System (Quelle: Chiron-Werke GmbH & Co. KG)

2.3 Grundlagen zum Begriff Fehler

Die Quellen (Thaller 2000; Huckle 2015) zeigen je eine Sammlung an publik gewordenen Softwarefehlern mit teils verheerenden Folgen. Bei der Sichtung der Auflistungen wird klar, dass selbst Projekte und Systeme mit hohem Entwicklungsbudget und erfahrenen Entwicklern nicht fehlerfrei sind. Die Fehler kosten im schlimmsten Fall viel Geld oder Menschenleben (Thaller 2000, S. 25f u. 29f). Dieser Exkurs zeigt, dass Fehlern in Software mit einer gewissen Methodik begegnet werden muss, um solch verheerende Folgen zu vermeiden. Das aktuelle Kapitel definiert dazu zunächst den Begriff des *Fehlers*.

Umgangssprachlich wird sowohl das Stoppen eines Fertigungssystems im Betrieb aufgrund einer Störung, als auch ein Tippfehler während der Programmierung oder das falsche Formulieren einer Anforderung im Pflichtenheft als *Fehler* bezeichnet. Die Fachliteratur unterscheidet diese Fälle durch dedizierte Begriffe. Zum erweiterten Verständnis sind die englischen Begriffe mit angegeben (Norm ANSI/IEEE 610.12 (R2002) 1990, S. 29):

1. **Irrtum** (mistake) – eine menschliche Aktion, die ein inkorrektes Ergebnis produziert
2. **Defekt** (fault) – ein inkorrekter Schritt, Prozess oder Datendefinition
3. **Fehler** (error) – die Abweichung zwischen einem berechneten, beobachteten oder gemessenen Wert oder Zustand und dem wahren, spezifizierten oder theoretisch korrekten Wert oder Zustand
4. **Versagen** (failure) – ein inkorrektes Resultat

(Zeller 2009, S. 2–4) veranschaulicht die Begriffe durch Aufstellen einer kausalen Kette: Der Programmierer begeht einen Irrtum (1.) und erzeugt damit einen defekten Programmcode. Der Defekt (2.) kommt zur Laufzeit des Programms zum Tragen. Der Defekt erzeugt dann Fehler (3.) im Zustand oder in Werten des Programms. Der ursprüngliche Defekt und der entstandene Fehler breiten sich unter Umständen in weiteren Teilen des laufenden Programmcodes aus. Am Ende der Kette steht der zu beobachtende Zustand des Versagens (4.), bspw. stürzt die Applikation ab.

Im Kontext der VIBN definiert (Norm VDI/VDE 3693 Blatt 1 2016, S. 4) einen Fehler als: „anormaler Zustand, der eine Einschränkung oder den Verlust der Funktion einer Betrachtungseinheit verursachen kann.“ Damit liest sich diese Definition von Fehler wie eine Mischung aus *Fehler* und *Versagen* nach der obigen Definition. Zur besseren Unterscheidung werden in dieser Arbeit die Definitionen nach (Norm ANSI/IEEE 610.12 (R2002) 1990) verwendet.

2.3.1 Fehlerklassifikation

Defekte und Fehler in Steuerungssystemen können zur Klassifizierung nach ihrer „Art“, ihrem „Entstehungszeitpunkt“ und ihrem „Entstehungsort“ eingeteilt werden. Abbildung 12 zeigt die Klassifizierung von Fehlern nach (Laible 2005).

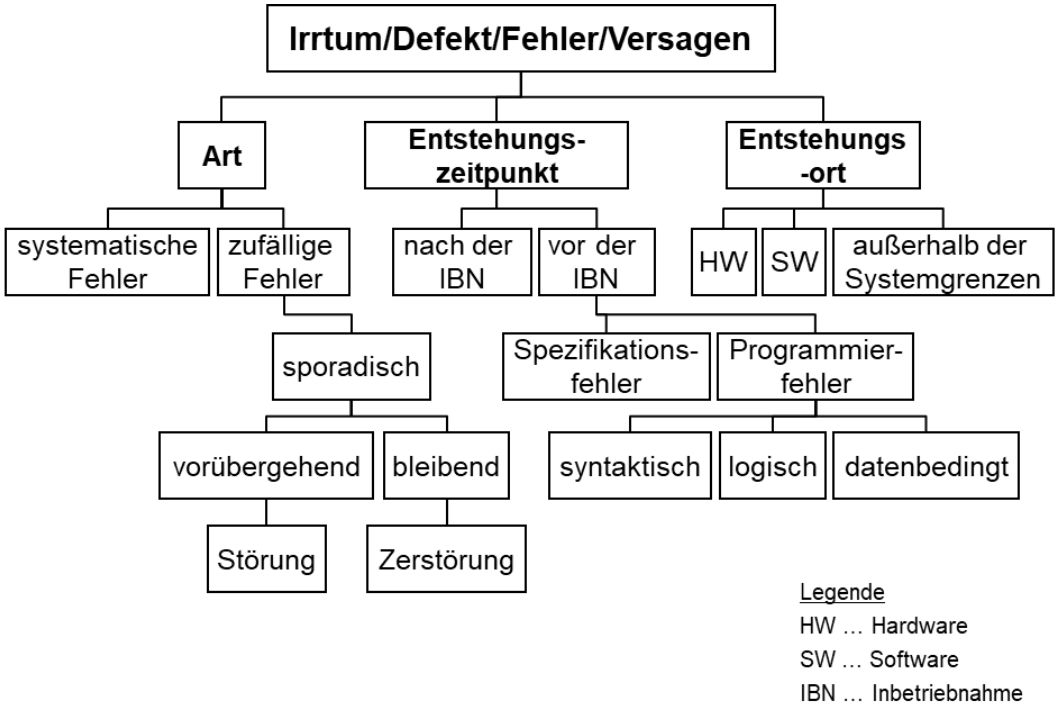


Abbildung 12 Klassifizierung von Fehlern in Steuerungssystemen nach Art, Zeitpunkt und Ort der Entstehung (nach Laible 2005, S. 54)

Noch vor der IBN eines Fertigungssystems können in der Analyse und dem Entwurf „Spezifikationsfehler“ gemacht werden. Bei der Implementierung können „Programmierfehler“ auftreten. Befindet sich der Fehler im Programmcode, kann in „syntaktische“, „logische“ und „datenbedingte“ Fehler unterteilt werden (Bourque et al. 2014, S. 13.8). Die vorab gemachten Fehler werden als „systematisch“ bezeichnet. Nach der IBN entstehen meist nur „zufällige Fehler“, bedingt durch falsche Benutzung oder Änderungen am System (Laible 2005). Da Fehler in Softwaresystemen nicht durch Alterung oder Verschleiß entstehen können, sind Fehler in der „Software“ systematischer Natur und immer seit der Entwicklungsphase im Programmcode (Spillner et al. 2012, S. 7). Bei der „Hardware“ können systematische und zufällige Fehler auftreten. Laible bezeichnet sporadisch vorübergehendes Versagen als „Störung“ und sporadisch bleibendes Versagen als „Zerstörung“.

Die Klassifikation von Fehlern kann für die *Fehlervermeidung* (in der Entwicklungsphase) und *Fehlerbeherrschung* (in der Betriebsphase) genutzt werden. Die Fehlerbeherrschung setzt eine *Fehlererkennung* voraus und definiert eine

Fehlerreaktion. *Fehlersichere* und *fehlertolerante* Systeme sind in der Lage, Fehler zu beherrschen. Steuerungssysteme zählen zu diesen Systemen und sind somit in der Lage, nach Auftreten eines Versagens im Betrieb Schaden zu vermeiden (fehlersicher) oder die Funktionalität beizubehalten (fehlertolerant) (Pritschow 2006, S. 259f). Arbeiten zu fehlersicheren und fehlertoleranten Steuerungssystemen finden sich bei (Laible 2005) und (Bock 2015). In dieser Arbeit liegt der Fokus auf der Fehlervermeidung durch Fehlererkennung, noch vor dem Betrieb des Fertigungssystems.

2.3.2 Fehlerentstehung

Die Software fehlertoleranter Systeme zeigt sich im realen Betrieb für ca. acht Mal mehr Ausfälle des Gesamtsystems verantwortlich als die Hardware (Montenegro 1999). Zudem werden Softwareprojekte nie fehlerfrei ausgeliefert (Liggesmeyer 2009, S. 31). Weiterhin bestehen Zusammenhänge zwischen den einzelnen Phasen der Softwareentwicklung und der *Fehlerentstehung*, der *Fehlererkennung* sowie den *Fehlerkosten*. Abbildung 13 zeigt diese Zusammenhänge, basierend auf (Möller 1996). Die Abbildung und deren Aussagen werden in (Balzert 2008, S. 484; Liggesmeyer 2009, S. 33) besprochen.

Aus der Abbildung 13 können drei Erkenntnisse abgeleitet werden. Erstens, die meisten Defekte entstehen bei der Programmierung (oberste Zeile). Dieser Sachverhalt wurde von Bürger, Grosse und Mayer mit gleicher Tendenz für Steuerungssysteme untersucht (Bürger 2005, S. 53f). Hier entstanden sogar 77 % der Fehler nur während der Programmierung. Damit sind testende Prüfverfahren aufgrund der hohen Fehlerrate in der Programmierungsphase von Steuerungssystemen von hoher Bedeutung.

Zweitens, während des Systemtests können die meisten Fehler gefunden werden (mittlere Zeile). Einen wissenschaftlichen Nachweis dazu liefert (Hampp 2010).

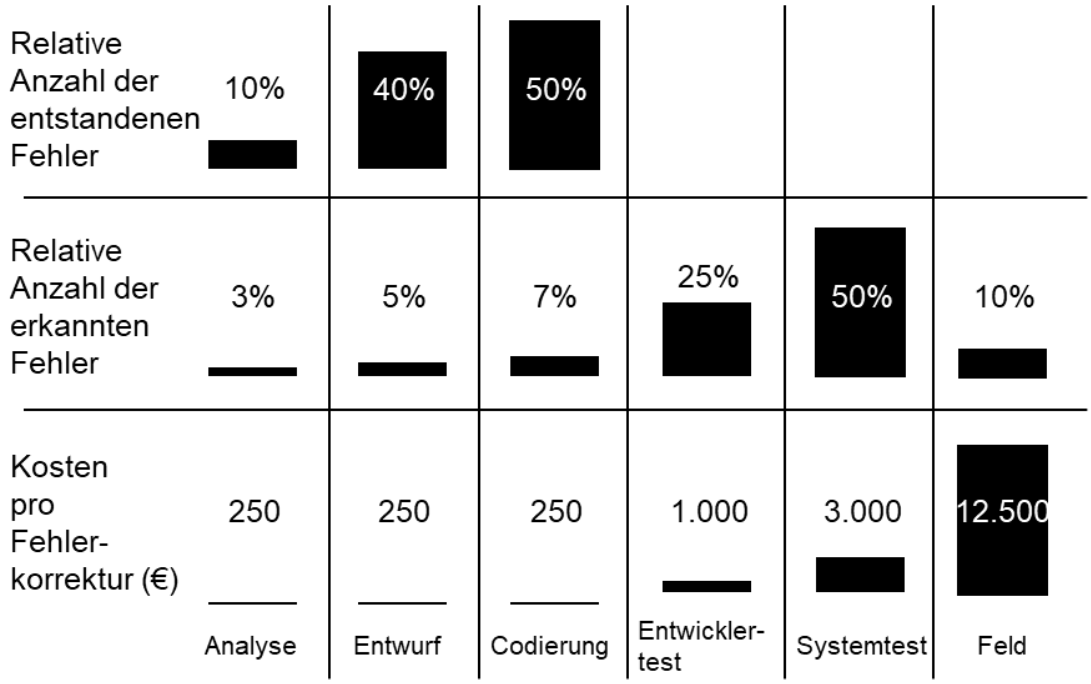


Abbildung 13 Fehlerentstehung, -erkennung und -kosten in Softwareprojekten (Liggesmeyer 2009, S. 33)

Drittens, eine längere Verweildauer von Fehlern im System führt zu einem exponentiellen Anstieg der Kosten einer Fehlerkorrektur in späten Phasen (unterste Zeile). Der dritte Sachverhalt existiert in ähnlicher Form in weiteren Quellen (Frühauf et al. 2007, S. 20; Bertsche et al. 2009, S. 200; Schmitt et al. 2015, S. 3; Witte 2016, S. 88) und wird auch mit *Rule-of-ten* (dt. Zehnerregel) oder *Barry-Boehm Kurve* bezeichnet. Auch die Schwere von Fehlern erhöht die Kosten der Korrektur (Hampp 2010, S. 167f). Aus den dargelegten Sachverhalten wird für diese Arbeit eine möglichst frühe Fehlererkennung motiviert, welche sich auf Programmierirrtümer und Entwurfsirrtümer fokussiert.

2.3.3 Fehlerkosten

Für die SPS Entwicklung zeigt (Oestreicher 1986), dass der größte einzelne Kostenfaktor bei der Erstellung von SPS Software der Test bei der Inbetriebnahme mit den in dieser Phase durchgeführte Korrekturen ist, ca. 40 % (siehe rechte Seite in Abbildung 14). Dieser Wert findet sich auch in der Softwaretechnikliteratur wieder, 40-50 % Testaufwand im Verhältnis zum Gesamtaufwand (Rumpe

2012, S. 172; Spillner et al. 2012, S. 188). Um die hohen Ausfallraten durch Software in fehlertoleranten Systemen, linke Seite Abbildung 14, zu reduzieren, bietet sich ausgiebiges Testen von realen Anwendungsszenarien und besonders Fehlerszenarien an. Das wiederum steht im Widerspruch zu den bereits hohen Kosten für Test und Korrektur der Software.

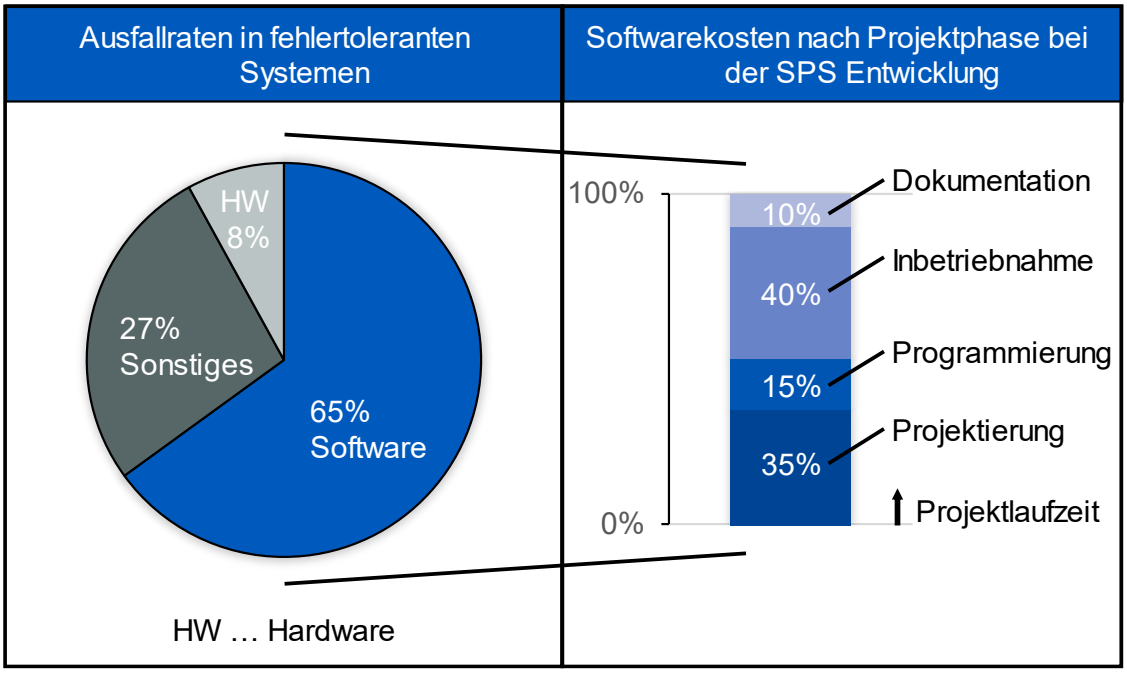


Abbildung 14 Verteilung von Softwareausfallraten und Softwarekosten in fehlertoleranten Systemen (rechte Seite nach Oestreicher 1986, S. 112; Montenegro 2000)

„Fehlerkosten“ sind Teil der Qualitätskosten in einem Projekt und sind aus „Fehlerbehebungskosten“, „Folgekosten“ (im Betrieb) und „Garantiekosten“ (im Betrieb) zusammengesetzt. Die Fehlerbehebungskosten beinhalten auch Fehlersuchkosten (Ludewig et al. 2010, S. 63). Abbildung 15 zeigt den Zusammenhang der Kostenarten, nach (Frühauf et al. 2007). Der Teil der Qualitätskosten kann die restlichen Herstellungskosten schnell übersteigen (Ludewig et al. 2010, S. 62), insbesondere wenn Folgekosten oder Garantiekosten aufgrund gravierender Schäden im Einsatz entstehen. Dabei ist individuell je Projekt und Unternehmen zu prüfen, wie man sein Budget auf die Qualität verteilt (Frühauf et al. 2007, S. 18). Weiterhin sprechen die Autoren (Frühauf et al. 2007, S. 19) von einem optimalen Kosten-Nutzen-Verhältnis, wenn Kosten für die Fehlerverhütung und Prüfung entstehen und nicht für die Fehler und Fehlerfolgen.

Abbildung 15 Aufteilung von Softwarekosten (aus Frühauf et al. 2007, S. 18)

Aus diesem Unterkapitel wird motiviert, das Budget für die Qualität in eine frühe und effiziente Fehlererkennung und -behebung zu planen, gerade für sicherheitskritische Steuerungssysteme.

2.4 Qualitätsbegriff und Softwaretest

In der internationalen Norm (Norm DIN EN ISO 9000 2015) und der daran angeschlossenen Normungsreihe zum Qualitätsmanagement werden die wichtigsten Begriffe zu Qualität als Fachbegriff festgelegt.

Der Begriff der **Qualität** für ein generisches Objekt (Produkt, Dienstleistung, Prozess, System, etc.) ist definiert als der Grad der Erfüllung von inhärenten Merkmalen gegenüber Anforderungen. Aus dieser Definition kann abgeleitet werden, dass Qualität eine messbare Größe ist.

Ein **Qualitätsmerkmal** ist ein „inhärentes Merkmal eines Objekts, das sich auf eine Anforderung bezieht."

Qualitätsmanagement (QM) ist Management bezogen auf Qualität.

Qualitätsanforderungen sind Anforderungen bezogen auf Qualität.

Qualitätssicherung (QS) dient als Teil des Qualitätsmanagements dazu, Vertrauen zu schaffen, dass Qualitätsanforderungen erfüllt werden.

Verifikation[6] bestätigt durch einen objektiven Nachweis, dass festgelegte Anforderungen erfüllt werden. Es wird bestätigt, wie ein Objekt etwas tut, bezogen auf den Qualitätsbegriff, die äußere und innere Qualität eines Objekts.

Validierung bestätigt durch einen objektiven Nachweis die Gebrauchs- oder Anwendungstauglichkeit, bezogen auf eine definierte Anforderung. Es wird bestätigt, was ein Objekt tut, bezogen auf den Qualitätsbegriff, die Gebrauchsqualität eines Objekts.

Test wird als die „Bestimmung entsprechend den Anforderungen an einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung“ definiert. “ Einsatzzweck ist die Validierung. Die Konformität wird durch Betreiben des Objekts nachgewiesen (Norm ANSI/IEEE 610.12 (R2002) 1990).

Prüfung wird als die Bestimmung des Grads der Konformität mit festgelegten Anforderungen definiert. Einsatzzweck ist die Verifikation.

In der Entwicklung von Produkten findet QM im Rahmen von eingesetzten Vorgehensmodellen statt, z. B. V-Modell (Bröhl et al. 1995), V-Modell XT (Verein zur Weiterentwicklung des V-Modell XT e.V. 2006) und Spiralmodell (Thaller 2002, S. 24f; Balzert 2008, S. 556–560). Manche Vorgehensmodelle beschreiben zwar Testprozesse, haben aber keine explizite QS (Balzert 2008, S. 552), z. B. das Wasserfallmodell (Royce 1970). Betrachtet man Fertigungssysteme als Produkt, so steigt bei deren Entwicklung die Abhängigkeit von Informatik und Software immer weiter an. Daher sind die Qualität und damit die Messung und Quantifizierung von Software von immer größerer Wichtigkeit.

Ein umfassendes QM bezieht sich auf alle Bereiche eines Unternehmens (Norm DIN 55350-11 2008) und umfasst, organisatorische, konstruktive und analytische QS-Maßnahmen (Frühauf et al. 2007, S. 20). Bei den analytischen QS-Maßnah-

[6] *Verifizierung, die* und *Verifikation, die* werden im Deutschen als Synonym für *Bestätigung, die* genutzt (Bibliographisches Institut GmbH 2019). Bei der Übersetzung ins Englische bedeutet beides *verification*. Daher werden die Begriffe als identisch angesehen, in der Softwareliteratur finden beide Begriffe Verwendung, aber immer nur einer je Quelle. Aufgrund der besseren phonetischen Unterscheidung zu Validierung wird in dieser Arbeit Verifikation verwendet.

men für Software grenzt sich das Testen von den formalen Methoden ab (im Anhang von Norm ISO/IEC/IEEE 29119-1 2013). (McCall et al. 1977) entwickelten eines der ersten Qualitätsmodelle für Software. Heute wird die Qualität von Software in der Normenreihe ISO/IEC 25000 SQUaRe standardisiert. Das Ziel von QM ist es, mit den vorhandenen Ressourcen an Zeit, Personal und Budget Fehlerraten zu minimieren und die geforderte Qualität zu erreichen. Der Test ist dabei ein wesentliches Hilfsmittel, um Fehler zu erkennen (Rumpe 2012, S. 172). Formale Methoden zur Verifikation von Steuerungssystemen sind nicht Teil dieser Arbeit und werden unter anderem in den Arbeiten von (Mertke et al. 2001; Huuck 2003; Vyatkin et al. 2009; Sünder et al. 2013; Legat et al. 2014; Weißenberger et al. 2015; Biallas 2016) betrachtet.

2.4.1 Techniken für den Softwaretest

„Programmtests können ein sehr effektiver Weg sein, um das Vorhandensein von Fehlern zu zeigen, sind aber hoffnungslos unzureichend, um deren Abwesenheit zu zeigen“[7] (Dijkstra 1972, S. 864). Die Aussage Dijkstras zeigt das Dilemma von Softwaretests beim Verbessern von Softwarequalität und könnte einen Grund darstellen, warum eine Vielzahl von Techniken für den Softwaretest zur Verfügung stehen. Testtechniken können mit Qualitätsmerkmalen verknüpft werden um einen gezielten Einsatz von Testtechniken zu ermöglichen (Norm ISO/IEC/IEEE 29119-1 2013, S. 16f). Zur Einführung der relevantesten Testtechniken werden diese anhand des V-Modells gezeigt, siehe Abbildung 16.

[7] Das Zitat im englischen Originalen lautet: „program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.“

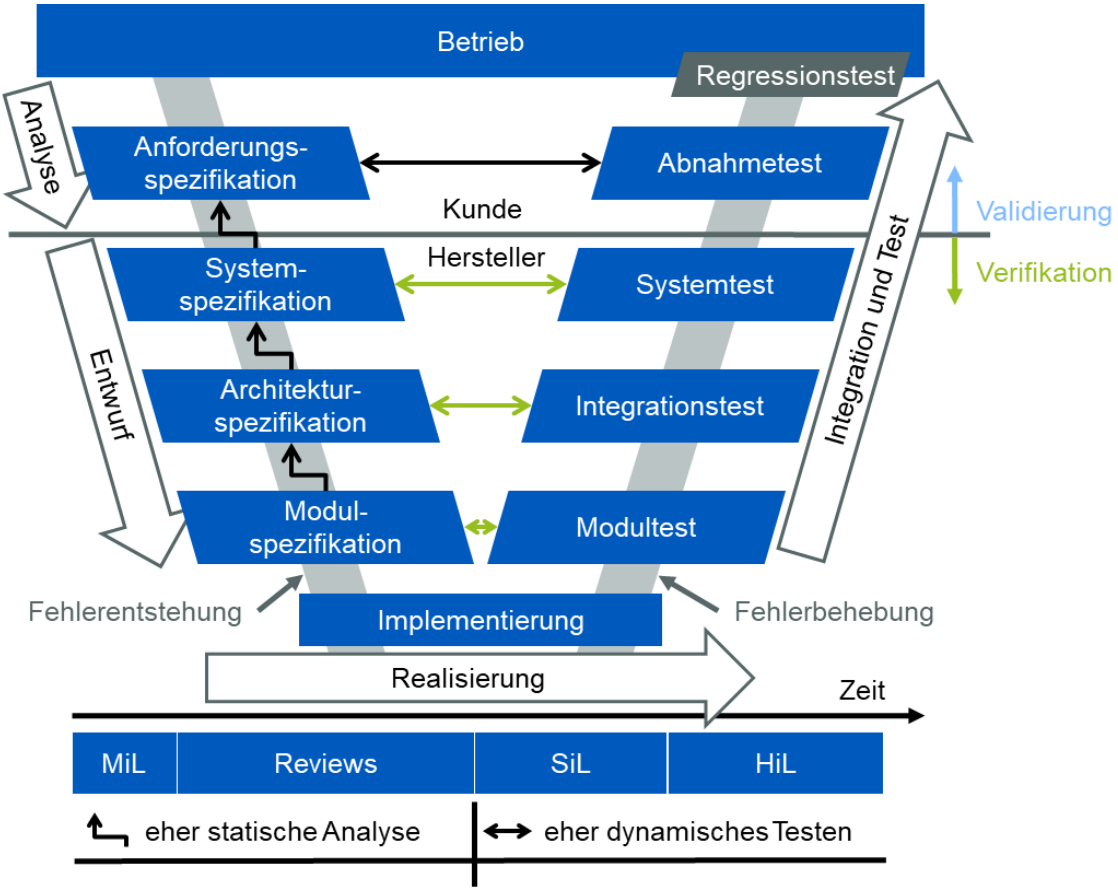


Abbildung 16 V-Modell zur Softwareentwicklung (Bröhl et al. 1995, S. 24; Dornseiff et al. 2001, S. 351; Frühauf et al. 2007, S. 19; Balzert 2008, S. 554; Spillner et al. 2012, S. 42; Schaefer 2012)

Auf der linken Seite des V-Modells entstehen nach Analyse und der ersten Stufe des Entwurfs die Anforderungen und die Systemspezifikation in Form von Lasten- und Pflichtenheft, (Norm VDI/VDE 3694 2014) definiert diese Dokumente für die Automatisierungstechnik. Die Architekturspezifikation und Modulspezifikation stellen den Grob- bzw. den Feinentwurf der Software dar. Die Architekturspezifikation beschreibt dabei wie die Systemspezifikationen im Pflichtenheft mittels Softwaremodulen und Schnittstellen umgesetzt werden. Die Modulspezifikation beschreibt den Aufbau und die Funktionsweise der einzelnen Module. Bei der Realisierung implementiert der Programmierer die entworfene Architektur mit Modulen und führt dabei kleinere Tests an Funktionen (auch Prozeduren oder Methoden genannt) durch. Die rechte Seite des V-Modells enthält nun eine Reihe von Testarten zur Verifikation und Validierung. Anhand der Modulspezifikation

werden Testfälle für den Modultest abgeleitet. Anhand der Architekturspezifikation werden Testfälle für die Integrationstest abgeleitet und so weiter bis hin zur Abnahme, denn Fehler werden auf der Abstraktionsebene entdeckt, auf welcher sie begangen werden (Ludewig et al. 2010, S. 63). Bei der Abnahme ist es im Idealfall der Kunde selbst der Testfälle definiert, da nur er den Gebrauchseinsatz des Systems kennt. Im Folgenden werden die relevanten Testtechniken aus der Literatur beschrieben.

Beim **Modultest***,* auch *Unittest*, werden individuelle Module und Routinen, meist vom Entwickler selbst getestet. Ziel ist es Fehler aus der Programmierung und Analyse zu finden. (Norm ISO/IEC/IEEE 24765 2017) Die Modultests testen gegen die Modulspezifikation aus dem Feinentwurf (Liggesmeyer 2009, S. 370). Ein Vorteil dieser Testtechnik ist die Möglichkeit zur Parallelisierung, da alle Module zeitgleich und unabhängig voneinander getestet werden können (Myers et al. 2012, S. 85).

Der **Integrationstest** findet durch stückweise Integration von Modulen oder von Systemkomponenten statt, wobei der Test deren Zusammenspiels im Fokus steht (Norm ISO/IEC/IEEE 24765 2017, S. 231). Wird daher auch als inkrementelles Testen bezeichnet (Myers et al. 2012, S. 118). Das inkrementelle Vorgehen kann funktionsorientiert oder strukturorientiert stattfinden, nichtinkrementell durch direkte Kopplung des Gesamtprogramms wird als Big-Bang bezeichnet (Hoffmann 2013, S. 163ff). Die innere Struktur oder weiter entfernte Schnittstellen werden bei diesen Test nicht betrachtet (Liggesmeyer 2009, S. 370). In der Praxis werden verschiedene Arten von Integrationstests unterschieden, bspw. der Subsystemintegrationstest, der Systemintegrationstest, der Hardware-Software-Integrationstest. (Liggesmeyer 2009, S. 370).

Beim **Systemtest** wird das fertiggestellte System gegen die festgelegten Funktions-, Leistungs- und Qualitätsanforderungen getestet (Liggesmeyer 2009; Norm ISO/IEC/IEEE 24765 2017, S. 456). Der Systemtest wird vom Hersteller durchgeführt, indem er die definierten Eigenschaften aus dem Pflichtenheft aus Sicht des Kunden testet (Hoffmann 2013, S. 167).

Abnahmetest, auch *Akzeptanztest*, beschreibt den „Test mit dem Ziel festzustellen, ob eine Software akzeptiert werden kann“ (Liggesmeyer 2009, S. 509). Üblicherweise führt der Käufer oder Auftraggeber diese Art von Test durch, um die Abnahme des Systems gegenüber dem Hersteller anhand definierter Kriterien zu bestätigen (Norm ISO/IEC/IEEE 24765 2017, S. 5).

Ein **Regressionstest** wird durchgeführt um die Fehler zu entdecken welche durch Modifikation in Software eingebracht wurden (Norm ISO/IEC/IEEE 24765 2017, S. 372). Regressionstests sind somit notwendig, nachdem eine bestehende Version einer Software durch Weiterentwicklung oder Fehlerkorrektur verändert wurde. Dazu werden die Ergebnisse der Tests mit den Ergebnissen der Vorgängerversion verglichen. (Hoffmann 2013, S. 198–200) Durch mehrmaliges Ausführen derselben Tests und einen zwingenden Vergleich mit der Vorgängerversion bietet sich für diese Testart die Nutzung eines Werkzeugs zur Automatisierung an (Metzner 2020, S. 153).

Whitebox-Testverfahren werden eher auf der Modultestebene, **Blackbox**-Verfahren auf der Systemtest eingesetzt (Myers et al. 2012, S. 119). Whitebox-Testverfahren unterscheiden sich von den Blackbox-Testverfahren weiterhin dadurch, dass sie Informationen über die Programmstruktur beim Testen nutzen, Blackbox-Verfahren hingegen nutzen Informationen zur Funktionsweise bzw. betrachten die Programmstruktur explizit nicht (Liggesmeyer 2009, S. 40). Mittels Blackbox-Tests wird getestet, was die Spezifikation verlangt (Ludewig et al. 2010, S. 504).

2.4.2 Testprozess

Der Testprozess für Software wird z. B. beschrieben in (Ludewig et al. 2010, Kapitel 19.3; Norm ISO/IEC/IEEE 29119-1 2013, Kapitel 5.2.1; Spillner et al. 2014, Kapitel 2; Sommerville 2018, S. 260). Die einzelnen Prozessbeschreibungen unterscheiden sich lediglich in ihrem Fokus nicht aber in ihrer Komposition oder Anordnung weswegen von gleichen Prozessen gesprochen werden kann. Abbildung 17 zeigt das Vorgehen im Testprozess entnommen aus (Spillner et al.

2014), der den Testprozess nach ISTQB[8]-Zertifizierung darstellt. Die originale Darstellung wurde auf der linken Seite um die Ergebnisse der einzelnen Prozessschritte erweitert. Der (dynamische) Testprozess endet mit dem Testbericht und unterscheidet sich damit insofern grundlegend vom statischen Prüfen (z. B. Reviews), als das er die Fehlerbehebung und Nachkontrolle nicht mit einschließt, vgl. (Witte 2016, S. 199; Sommerville 2018, S. 791). Im folgenden Verlauf wird der Prozessschritt „Bewertung von Endkriterien und Bericht“ als *Testauswertung* bezeichnet.

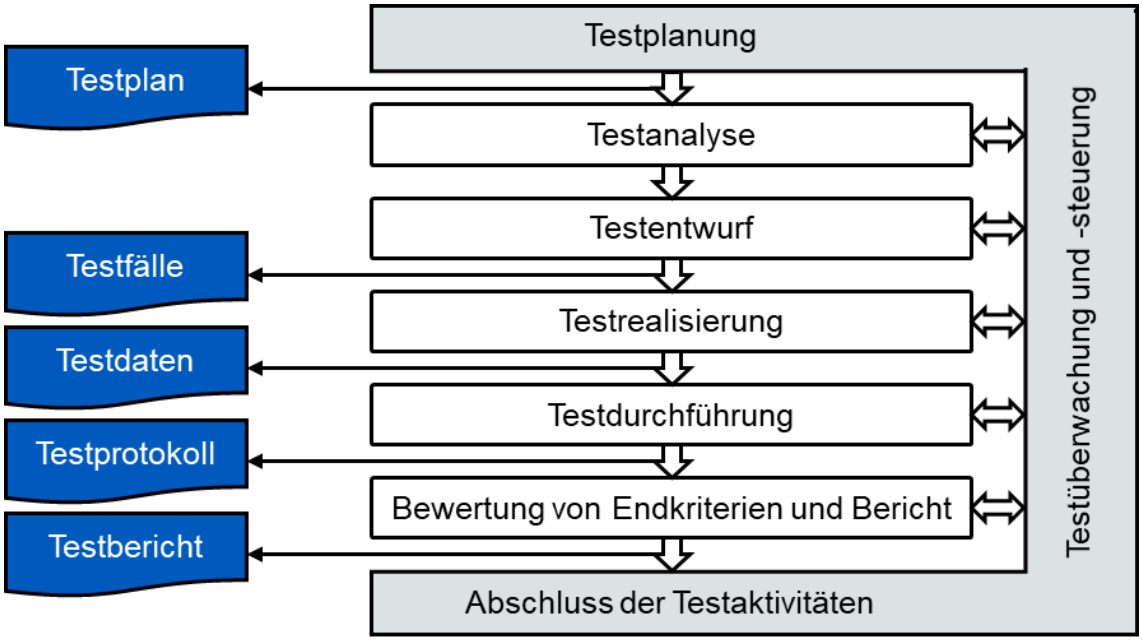


Abbildung 17 Testprozess nach ISTQB-Lehrplan (Spillner et al. 2014, S. 14) erweitert um Artefakte (blau)

2.4.3 Testautomatisierung

Der hohe Kosteneinsatz in der Testphase von Software macht die Automatisierung der Testtätigkeiten zu einem relevanten Thema (Arcuri 2008). Die Testautomatisierung kann dabei im Vergleich zum manuellen Test als die effizientere Prüfung bezeichnet werden (Dornseiff et al. 2001, S. 356). Der Begriff *Testautomatisierung* wird von (Bucsics et al. 2015, S. 7) definiert als „[…] die Durchfüh-

[8] ISTQB – International Software Testing Qualifications Board, www.istqb.org

rung von ansonsten manuellen Testtätigkeiten durch Automaten“. Der Teststandard nach ISTQB (Norm ISTQB/GTB Version 2.3 2014) definiert Testautomatisierung als „Einsatz von Softwarewerkzeugen zur Durchführung oder Unterstützung von Testaktivitäten, z. B. Testmanagement, Testentwurf, Testdurchführung und Soll/Ist-Vergleich.“. Abbildung 18 zeigt Vor- und Nachteile der Testautomatisierung.

Vorteile	Nachteile
• Genauigkeit und Vollständigkeit der Ergebnisse	• Hohe Investitionen beim Kauf von Lösungen und Schulungen
• Genauigkeit des Ergebnisprotokolls und der zusammenfassenden Berichte	• Hohe Investitionskosten für die Entwicklung von Lösungen
• Ausführung von kompletten Regressionstests	• Hohe Personalressourcen für den Testentwurf
• Wenig Personalressourcen für die Testdurchführung	• Erhebliche Testbereiche bleiben unerschlossen
• Kürzere Testzeiten	
• Umfassende Informationen zu Fehlern	
• Ausführung von Testklassen außerhalb des manuellen Tests	

Abbildung 18 Vor- und Nachteile von Testautomatisierung (Galin 2004, S. 242–244)

(Bender 2005) benennt ein Softwarewerkzeug für Testautomatisierung als Testautomat. (Spillner et al. 2012, S. 209–210) nennen weitere Begriffe wie Testframework (dt. Testrahmen) und CAST (Computer-Aided Software Testing) Tool (dt. Werkzeug). Je nachdem welche Aktivität unterstützt wird werden Testwerkzeuge noch spezifischer benannt, z. B. Performancetestwerkzeug für die automatische Ausführung des Performancetests.

Die Softwareentwicklung von Fertigungssystemen ist trotz steigendem Softwareanteil dem Stand der Technik der Softwaretechnik hinterher. In der Softwaretech-

nik wird neben dem Erkennen von Fehlern auch bereits das Auffinden und Beheben automatisiert (bspw. in Renieres et al. 2003; Arcuri 2008; Zeller 2009, S. 14–15; Reiss 2014).

Im Rahmen dieser Arbeit wird der Begriff CAST Werkzeug als Oberbegriff für ein Testwerkzeug benutzt, zusammen mit der Definition nach ISTQB. Der Begriff CAST wird in Anlehnung an Computer-Aided Software Engineering (CASE) verwendet und wurde in den frühen 1990er Jahren geprägt (Graham et al. 1993).

2.4.4 Softwarequalitätsmetriken

Mittels Metriken kann eine quantitative Aussage über den Erfüllungsgrad einer Eigenschaft durch das untersuchte Objekt getroffen werden (in Anlehnung an (Norm ANSI/IEEE 610.12 (R2002) 1990). Metriken können für verschiedene Eigenschaften eines Objekts definiert werden. In der Softwaretechnik werden Metriken zur Messung der Qualität, sogenannte Softwarequalitätsmetriken, genutzt. Der IEEE Standard 1061-1998 (Norm IEEE 1061-1998 2009, S. 3) definiert eine Metrik für Softwarequalität als „eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet, welcher als Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit interpretierbar ist.“[9]. Gängige Metriken sind bspw. Robustheit, Verfügbarkeit, Erweiterbarkeit und Performanz (siehe McCall et al. 1977). Eingesetzt werden Softwarequalitätsmetriken auch, um für die Testphase ein quantitatives Abbruchkriterium zu erhalten (Spillner et al. 2012, S. 30f): ein vereinfachtes Kriterium wäre die „Anzahl durchgeführter Testfälle“ zu quantifizieren. Die Beschreibungen weiterer sogenannter Testendmetriken finden sich in (Thaller 2002, S. 287–291; Ludewig et al. 2010, S. 496f; Myers et al. 2012, S. 135–141; Spillner et al. 2014, Kapitel 7).

[9] Das Zitat im englischen Original lautet „A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.“

2.5 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Begriffe und thematischen Grundlagen für den Themenkomplex in dieser Arbeit erläutert.

Industrielle Steuerungstechnik: Die programmierten Steuerungsprogramme auf Steuerungssystemen sind maßgeblich für die einwandfreie und passende Automatisierung von Fertigungssystemen verantwortlich. Damit ist die Zuverlässigkeit und Verfügbarkeit der Steuerungssysteme und der Steuerungsprogramme von hoher Bedeutung für die Gesamtzuverlässigkeit und -Verfügbarkeit eines Fertigungssystems. Offene Steuerungssysteme ermöglichen es Softwarekomponenten von Drittanbietern zu integrieren. Dies erhöht die Komplexität und den Testaufwand bei der Realisierung und Integration der Steuerungssysteme.

VIBN und **HiL**: VIBN hat es von einem Forschungsthema in die Anwendung für Fertigungssysteme geschafft. VIBN ist eine etablierte Methode für das Simultaneous Engineering geworden. Der Einsatz von VIBN dient dem frühzeitigen Testen des Fertigungssystems. Die Kombination aus Simulationsmodellen, Simulationswerkzeug und standardisierten Methoden macht die VIBN zu einem Teil der Digitalen Fabrik. Als virtuelle Testumgebung für die VIBN erreichen RT-HiL Systeme den höchsten Grad an Konkretisierung.

Fehlerbegriff: Neben dem umgangssprachlichen Fehler wird in der Fachliteratur zwischen den Begriffen Irrtum, Defekt, Fehler und Versagen unterschieden. Die Begriffe stehen bezogen auf die Anwendung im Softwarelebenszyklus in einem kausalen Zusammenhang. Der Großteil an Defekten in Programmcode entsteht während der Programmierung, die Entdeckung und Behebung findet in der Systemtestphase statt.

Qualität und **Softwaretest**: Der Softwaretest ist als QS-Maßnahme Bestandteil von QM für Softwareentwicklung. Aus der Softwaretechnik sind verschiedene Testtechniken für verschiedene Entwicklungsphasen bekannt. Zur Beurteilung der Qualität von Software und der Effektivität von Testen existieren standardisierte Metriken für Softwarequalität. Der Testprozess ist standardisiert und lässt sich mittels CAST Werkzeugen automatisieren.

3 Anforderungen an eine neue Testmethodik

In diesem Kapitel werden Anforderungen an eine neue Testmethodik für Fertigungssysteme erarbeitet. Dazu werden zunächst Erkenntnisse aus den Grundlagen aus Kapitel 1 zusammengefasst. Anschließend werden die Defizite des aktuellen Testprozesses bei Fertigungssystemen aufgearbeitet. Aufbauend zu den Überlegungen aus Grundlagen und aktuellen Defiziten werden die konkreten Anforderungen erarbeitet. Aus dem vorangegangenen Kapitel „Grundlagen und Festlegungen für die Arbeit“ sind folgende Erkenntnisse für die Erarbeitung der Anforderungen relevant:

- Für Fertigungssystemen haben sich komplexe und umfangreiche Softwarelösungen etabliert. Der Test und die Fehlerbehebung in der Software sind somit für die gesamte Qualitätssicherung wichtiger geworden.
- In der Phase der Programmierung der Software entstehen die meisten Irrtümer, daher ist der Test nach dieser Phase wichtig.
- Vor der Auslieferung an den Kunden müssen so viele Fehler wie möglich behoben werden, weil ansonsten teure Serviceeinsätze notwendig sind, u. U. Strafzahlungen fällig sind und zusätzlich Imageschäden entstehen können. Der Test vor dieser Phase ist somit wichtig.
- Nach Programmierung und Integration und noch vor Auslieferung werden heute Systemtests (Verifikation) und Abnahmetests (Validierung) durchgeführt. Dabei werden die meisten Fehler entdeckt, da erstmalig am Gesamtsystem getestet wird. Die beiden Testarten stehen somit im Fokus für die neue Testmethodik.
- Bei (Software-)Änderungen am Fertigungssystems während der Betriebsphase werden Regressionstests notwendig, diese sind von der neuen Testmethodik zu unterstützen.
- System-, Abnahme- und Regressionstest zählen zu den dynamischen Blackbox Testverfahren.
- Die Vorteile einer zeitlich längeren und früheren Test- und Optimierungsphase durch VIBN sollen für die neue Testmethodik genutzt werden.

3.1 Defizite des aktuellen manuellen Testprozesses

Die Literatur in Form von wissenschaftlichen Veröffentlichungen liefert eine nur geringe Anzahl an Informationen zu Defiziten im aktuellen Testprozess (Linke 2011; Kormann et al. 2012). Die Quellen beziehen diese sich zudem nur auf SPS Systeme nicht aber auf ein gesamtes Steuerungssystem. Daher wurde eine eigene Erhebung in der Industrie durchgeführt, die Ergebnisse wurden in (Kübler et al. 2016) veröffentlicht. Folgende Defizite können für den aktuellen Testprozess von Fertigungssysteme identifiziert werden:

1. Die Gesamtzeit für die Tests der IBN wird durch den Einsatz von VIBN nicht verringert (Zäh et al. 2006, S. 597; Frieß 2014, S. 22; Kübler et al. 2016, S. 18).
2. Die Nebenzeiten eines Testfalls, dazu gehört die virtuelle Testumgebung in einen definierten Ausgangs- und Endzustand zu bringen, übersteigen den Zeitaufwand der eigentlichen Testdurchführung (Kübler et al. 2016, S. 18).
3. Die Durchführung der Testabläufe ist bei realer oder virtueller Testumgebung eine manuelle Tätigkeit und damit fehlerbehaftet (Kübler et al. 2016).
4. Vor dem Hintergrund des zunehmenden Softwareanteils in Fertigungssystemen lässt die manuelle Verifikation keine umfassende Aussage mehr über Fehlerfreiheit oder Qualität der untersuchten Software zu (Denkena et al. 2007, S. 124).
5. Explizite Zeiten für die Testphase sind in den Projektplänen nicht berücksichtigt (Kübler et al. 2016, S. 18), zudem fehlen systematische Testprozesse (Kormann et al. 2012).
6. Die notwendige treue gegenüber Lieferterminen und geforderte schnellere Lieferzeiten von Produktionssystemen lässt weniger Zeit zum Testen als benötigt (Zetzsche 2019).
7. Standards für die Entwicklung und den Test der Software fehlen (Kormann et al. 2012, S. 4).
8. Für die Erstellung, Verwaltung und Durchführung der Testabläufe fehlen geeignete Softwarewerkzeuge (Kormann et al. 2012, S. 5).

9. Erhöhter Aufwand durch Regressionstests nach Updates von Drittanbieter am offenen Steuerungssystem, nicht zuletzt die Firmware des Steuerungshersteller, kann durch den rein manuellen Testprozess schlecht bewältigt werden (Kübler et al. 2016, S. 18).

Die Darlegungen, sowohl aus eigener Arbeit als auch aus Quellen Dritter, bestätigen die in Kapitel 1.1 aufgestellte Hypothese, eines unzureichenden Testprozesses bei steigender Wichtigkeit von Software in Fertigungssystemen.

3.2 Anforderungen zur Automatisierung

Aus der vorangegangenen Auflistung lassen sich die identifizierten Defizite 1, 2, 3, 4, 8 und 9 explizit mittels Automatisierung im Testprozess lösen. Implizit können Defizite 5 und 6 durch ein methodisches und unterstütztes Vorgehen bei der Testautomatisierung gelöst werden. Defizit 7, zur Definition von Standards zum Entwickeln und Testen der Software, steht nicht im Fokus dieser Arbeit. Die Notwendigkeit zur Automatisierung von Testabläufen und nach einem methodischen Vorgehen im Testprozess für Fertigungssysteme ist aus anderen Arbeiten hinlänglich bekannt, siehe (Denkena et al. 2007; Wünsch 2008). Weiterhin bleibt das Thema ungelöst und gewinnt erneut an Präsenz, z. B. im Rahmen von in Losgröße 1 produzierter Software (Weyrich et al. 2018, S. 29).

Im Folgenden werden neun Anforderungen definiert:

/A1/ Der Testprozess wird durch Automatisierung mittels eines digitalen Werkzeugs unterstützt (Testautomatisierung mittels CAST Werkzeug)

Von (Simon 2007) wurden Anforderungen an eine optimale Inbetriebnahme-Umgebung für den System- und Abnahmetest für Steuerungssysteme definiert: Realitätsnahe Prozessdarstellung, genaue Zeitberechnung, exakte Bewegungsführung, Unterstützung des Sprachumfangs und originalgetreue Bedienung. Dies kann wie in Kapitel 2.2.1 erläutert mit einem RT-HiL System gewährleistet werden. Zudem soll sich mit der neuen Testmethodik das frühere zur Verfügung stehen einer virtuellen Testumgebung zu Nutze gemacht werden.

/A2/ Neben einer realen Testumgebung muss eine virtuelle Testumgebung mittels RT-HiL unterstützt werden (System-under-Test real/virtuell)

Während der Durchführung von Testfällen müssen Zustände im SuT (System-under-Test) ausgelesen und beeinflusst werden können. Das Steuerungssystem im SuT muss ebenfalls ausgelesen aber nicht beeinflusst werden können. Zusätzlich muss das SuT fernsteuerbar sein um neben der Testdurchführung auch die Nebenzeiten des Tests zu automatisieren. Hervorzuheben ist, dass alle Komponenten des Steuerungssystems (SPS, NC und die HMI) relevant für die Testdurchführung sind:

/A3/ Das CAST Werkzeug muss in der Lage sein, mit den Komponenten des SuT zum Zweck der Testautomatisierung interagieren können (Testtreiber)

3.3 Anforderungen der Entwicklung von Fertigungssystemen

In Kapitel 2.1.3 „Beteiligte Rollen“ wurde erläutert welche Rollen bei der Entstehung eins Fertigungssystems beteiligt sind. Bisher unerwähnt blieb die Tatsache, dass die Auswahl des Steuerungssystems nicht immer durch den OEM, sondern häufig durch den Kunden oder Nutzer vorgeschrieben wird. Dies ist zu begründen mit dem beim Kunden/Nutzer vorherrschenden Know-how bei der Programmierung, Inbetriebnahme und Bedienung des Steuerungssystems. Dieser Umstand muss beim Entwurf einer Testautomatisierung mittels CAST Werkzeug berücksichtigt werden.

/A4/ Die Definition von Testfällen im CAST Werkzeug muss für den Benutzer unabhängig von der herstellerspezifischen Ausprägung des SuT möglich sein (Herstellerunabhängigkeit)

Die Programmierung von Steuerungssystemen ist, wie in Kapitel 2.1.2 beschrieben, keine Tätigkeit die von Softwaretechnikern durchgeführt wird. Weiterhin aus Kapitel 2.1.2 bekannt ist die Vielzahl an grafischen Programmiersprachen. Beide Umstände sollen auch bei der Erstellung von Testfällen für die Automatisierung berücksichtigt werden.

/A5/ Die Erstellung der Testfälle erfolgt in der Sprache des Maschinenbauers (Geeignete Testfallmodellierung)

Erfahrungswerte aus Forschung und Industrie zeigen, dass der Anteil an Diagnose- und Fehlererkennungsroutinen in der Automatisierungssoftware ausgehend von ca. einem Drittel in den 1990er Jahren (Seifert 1992, S. 13) auf ca. zwei Drittel angestiegen ist (Pieloth 2013). Damit muss dieser umfangreiche und wichtige Teil der Steuerungsprogramme beim Testen berücksichtigt werden.

/A6/ Berücksichtigung von Fehler- und Diagnosefällen bei der ganzheitlichen Testautomatisierung (Schlechtfalltest)

Zu /A6/ Wichtig ist nicht nur das Verhalten der Steuerung im Schlechtfall am Zielsystem zu testen, sondern auch die richtige Verhaltensweise zur wieder Instandsetzung (Röck 2007, S. 26f).

3.4 Anforderungen für einen effizienten Testprozess

Im Grundlagenkapitel wurden die allgemeinen Phasen des Testprozesses beschrieben. Für die drei wichtigsten Phasen Testentwurf, Testdurchführung und Testauswertung werden in diesem Kapitel Anforderungen definiert, um deren Aufwand zu reduzieren. Neben der Korrektheit der Ergebnisse ist der Aufwand in im Testprozess entscheidend für dessen Anwendbarkeit. Eine aufwandsarme Testdurchführung wird bereits mit Anforderung /A1/ zur Testautomatisierung adressiert. Mit den Anforderungen /A4/ (Herstellerunabhängigkeit) und /A5/ (Geeignete Testfallmodellierung) wird u. a. der Testentwurf unterstützt. Ein weiteres Konzept zur Aufwandsreduzierung beim Testentwurf stellt die Wiederverwendung dar.

Wiederverwendung wird von der (Norm VDI/VDE 3695 Blatt 3 2010) als Methode zur Optimierung hinsichtlich Kosten, Zeit, Qualität und Wartungsaufwand der Lösung empfohlen. Auf den Testprozess von Fertigungssystemen lässt sich diese Empfehlung in zweierlei Hinsicht anwenden. Zum einen in der auf Wiederverwendung ausgelegten Erstellung von Testfällen. Zum anderen durch die Verknüpfung der Testfälle mit einem korrespondieren Bestandteilen des Fertigungssystems dessen Funktionalität mit diesem Testfall geprüft wird.

/A7/ Auf Wiederverwendung ausgelegtes Erstellen der Testfälle (Wiederverwendung)

Je nach Größe der Systeme und Komplexität von Prozessen sind Zusammenhänge in Fertigungssystemen nicht immer direkt ersichtlich. Zudem sind lange Laufzeiten, ohne wirtschaftlich schädliche Stillstandszeit, erwünscht. Somit können Irrtümer aus der Programmierung oder Konzeptphase mit einer langen Totzeit im System verweilen und führen erst spät zu Fehlern und damit zu Versagen in der Fertigung. Es wird ersichtlich, dass es im Verhalten von Fertigungssystemen zu sporadischen Effekten kommen kann. Gerade bei System-, Abnahme- und Regressionstests entsteht daher der Wunsch nach einem Dauerlauf. Im bisher manuellen Testprozess ist dies nicht abbildbar, weder die Durchführung noch die Überwachung der Testfälle.

/A8/ Die Testdurchführung muss einen Dauerlauf ermöglichen (Dauertest)

Wie in den Grundlagen dargelegt entsteht in jedem Testprozess ein Testprotokoll. Die automatisierte Testdurchführung, insbesondere im Dauerlauf, erbringt entsprechend lange Testprotokolle. Daher ist die manuelle Auswertung von z. B. einer mehrtägigen Testdurchführung nicht praktikabel.

/A9/ Die Auswertung von erzeugten Testprotokollen und dynamischen Testergebnissen muss automatisiert erfolgen (Komparator)

3.5 Zusammenfassung

In Kapitel 3 wurden systematisch Anforderungen für ein neues Konzept zur Testautomatisierung für Fertigungssysteme hergeleitet. Initial führte die Herleitung der Notwendigkeit zur automatisierten und damit werkzeuggestützten Durchführung der Tests zu den Anforderungen /A1/ bis /A3/. Auf diesen Anforderungen aufbauend wurden Anforderungen für die werkzeuggestützte Testautomatisierung speziell für Fertigungssysteme hergeleitet (Anforderungen /A4/ bis /A6/). Abschließend wurden weitere Anforderungen (/A7/ bis /A9/) für eine ganzheitliche und effiziente Testautomatisierung definiert. Der Begriff der *ganzheitlichen Testautomatisierung* wird eingeführt als eine Testautomatisierung, die werkzeuggestützt die Aspekte Management, Spezifikation, Durchführung und Auswertung von Testfällen und Testnebenzeiten ermöglicht. Abbildung 19 zeigt alle neun erfassten Anforderungen geclustert nach deren Herleitung.

Anforderungen		
Cluster	**Nr.**	**Stichwort**
Automatisierung des Tests	/A1/	Testautomatisierung mittels CAST Werkzeug
	/A2/	System-under-Test real/virtuell
	/A3/	Testtreiber
Branche des Maschinen- und Anlagenbaus	/A4/	Herstellerunabhängigkeit
	/A5/	Geeignete Testfallmodellierung
	/A6/	Schlechtfalltest
Effizienter Testprozess	/A7/	Wiederverwendung
	/A8/	Dauertest
	/A9/	Komparator

Abbildung 19 Zusammenfassung und Clusterung der Anforderungen

4 Stand der Wissenschaft und Technik und Zielsetzung der Arbeit

In diesem Kapitel werden, ausgehend von der Fragestellung und den Anforderungen, im Rahmen dieser Arbeit recherchierte, relevante wissenschaftliche Arbeiten und kommerzielle Lösungen diskutiert. Voraussetzung für die Betrachtung einer Lösung ist die Fokussierung auf Testautomatisierung im Kontext von fertigungstechnischen Steuerungssystemen.

4.1 Forschungsarbeiten zu Test und Testautomatisierung im Kontext des Maschinen- und Anlagenbaus

In diesem Kapitel werden wissenschaftliche Arbeiten zusammengefasst und bewertet, welche die Themen „Testen/Testautomatisierung im Kontext von Steuerungsprogrammen“ oder „Testen an einer virtuellen Testumgebung“, werden nur solche Arbeiten eingeschlossen, welche maschinennahe Steuerungen (NC und SPS) betrachten. Die Reihenfolge der Arbeiten ist chronologisch und alphabetisch gewählt. Unterschiede in der Ausführlichkeit der Zusammenfassungen und Bewertungen rühren von mehr oder weniger überschneidenden Aspekten der betrachteten Arbeit mit den Anforderungen her. Am Ende dieses Kapitels steht mit Abbildung 20 auf Seite 56 eine tabellarische Bewertung der Forschungsarbeiten als Schnellübersicht zur Verfügung.

(Kohring 1993)

Kurzzusammenfassung: (Kohring 1993) analysiert in seiner Dissertation „Systematisches Projektieren und Testen von Steuerungssoftware für Werkzeugmaschinen“ die SPS Softwareentwicklung, um ein systematisches Vorgehen bei deren Entwicklung zu etablieren. Der zweite Schwerpunkt ist die Erarbeitung einer Möglichkeit zur Vorabinbetriebnahme der SPS inklusive Software. Sein Ziel ist es, die Durchlaufzeiten in der Entwicklung bei kundenspezifischen Werkzeugmaschinen zu reduzieren. Besonderen Handlungsbedarf sieht Kohring in der systematischen Spezifikation der Steuerungsaufgaben und dem systematischen und

vorgezogenen Test der SPS Software. Er schlägt eine rechnergestützte und abteilungsübergreifende Erstellung von Pflichtenheften vor. Bei der Testplanung soll das Pflichtenheft als Grundlage dienen. Bei der prototypischen Realisierung nutzt Kohring eine zweite SPS zur echtzeitfähigen Simulation der Werkzeugmaschine. Ein PC dient zur Bedienung und Visualisierung des Echtzeitteils auf der SPS. Der Testablauf geschieht komplett manuell. Damit definiert Kohring einen der ersten Ansätze zur echtzeitfähigen HiL Simulation und stellt Bedingungen an eine systematische Erstellung von Testfällen.

Bewertung: Bedingt durch die frühe Entstehungszeit ist die Arbeit von Kohring kein Beitrag zum aufwandsarmen und methodischen Testen von Fertigungssystemen. Auch der Entwurf eines den Test unterstützenden Werkzeugs fehlt. Kohring hatte zu seiner Zeit zunächst die Entwicklung eines HiL Systems zum Ziel, um den koordinierten Test an einem virtuellen Modell zu ermöglichen.

(Bender 1999)

Kurzzusammenfassung: Die Quelle (Bender 1999) ist der Projektabschlussbericht des Forschungsprojekts SEMI mit Projektende im dritten Quartal 1998. Ziel des Projekts war es, die Entwicklung von intelligenten Maschinen für die Produktion zu unterstützten, insbesondere beim Test der Steuerungsprogramme. Das Ergebnis des Projekts ist eine Kombination aus Methoden und Werkzeug zur virtuellen Inbetriebnahme von SPS-gesteuerten Maschinen. Die Projektbearbeiter erstellen einen Prototyp einer RT-HiL Simulation mit erweiterten Testmitteln. Als Methoden zum Werkzeug werden definiert: Ein komponentenbasiertes Vorgehen bei der Erstellung der Maschinensimulation, ein methodisches Vorgehen zur Struktur- und Verhaltensbeschreibung der Modellkomponenten und die Wiederverwendung durch die bibliotheksbasierte Erstellung von Modellkomponenten. Auf den Seiten 95 – 111 wird explizit auf eine zur HiL Simulation passende Testumgebung eingegangen. Die Testumgebung muss laut Bender zum einen die Kommunikation zwischen Steuerung und Simulation beobachten, aufzeichnen und auswerten, und zum anderen die Zustände in der Simulation beeinflussen können. Im Projekt wurden drei Lösungskonzepte für die Testumgebung entworfen, die sich darin unterscheiden, auf welchen Hardwareplattformen sie realisiert

werden. Ziel der Testumgebung ist es, die reine Simulation des Maschinenmodells für die virtuelle Inbetriebnahme durch geeignete Testmittel zu erweitern: Visualisierung, Steuerung, Triggern und Aufzeichnung. Dahinter stecken die Funktionen Aufzeichnungen von E/A-Verläufen der SPS und Modellinformationen der Simulation, Forcieren von E/A-Signalen der SPS und Modellsignalen der Simulation, ereignisgesteuertes Triggern der Aufzeichnungen und der Forcierungen, sowie die grafische Darstellung von Signalen und grafische Anpassung von Triggereinstellungen.

Bewertung: Die grundsätzlichen Ideen aus dem Projekt SEMI sind heute Stand der Technik in der VIBN: Aufwandsarme Simulationsmodellerstellung, erweiterte Visualisierung, Aufzeichnung und Eingriffsmöglichkeiten während der Simulation, wenngleich nicht alle HiL Systeme heute echtzeitfähig und bussystemübergreifend funktionieren. Fehlend in der Lösung sind die Ankopplungen von NC Systemen und der HMI eines Steuerungssystems. Weiterhin wird die Automatisierung der Testdurchführung und Testauswertung nicht ermöglicht.

(Kreusch 2002)

Kurzzusammenfassung: (Kreusch 2002) erarbeitet in seiner Dissertation „Verifikation numerischer Steuerungen an virtuellen Werkzeugmaschinen“ eine Möglichkeit zur virtuellen Absicherung von numerischen Steuerungen an virtuellen Werkzeugmaschinen. Dazu entwickelt Kreusch zunächst ein Konzept für den RT-HiL Betrieb. Das Konzept umfasst die Möglichkeit der virtuellen Modellerstellung in den Disziplinen Kinematik, Maschinendynamik und logisches Verhalten (Aktoren und Sensoren). Die virtuelle Werkzeugmaschine erweitert er um das Konzept eines Vorgehensmodells zur Verifikation sowie die Idee eines Testrechners zur Unterstützung von Simulationsläufen. Die Funktionalitäten des Testrechners definiert er als zentrale Datenhaltung (z. B. für NC-Programme), zur Konfiguration und Steuerung von Komponenten der RT-HiL, zur Aufzeichnung und Protokollierung und zuletzt zur Auswertung von Simulationsergebnissen. Bei der Erstellung von Testfällen ist es Kreusch wichtig, dass Erfahrungswissen in Testfälle einmündet. Die Realisierung seiner Arbeit umfasst die Kopplung eines virtu-

ellen Verhaltensmodells einer 3-Achs Werkzeugmaschine mit einer 3D Visualisierung; das Verhaltensmodell wurde dabei von einer realen NC angesteuert. Bei der Erprobung der Simulationsumgebung legt Kreusch besonderen Wert auf die koordinierte Bewegung der Achsen. Bei einem Simulationslauf zeichnet Kreusch z. B. Achspositionen auf und wertet diese offline im Nachgang aus. Durch Aufzeichnung und anschließende Auswertung war eine Aussage über die Güte der Steuerung möglich. Kreusch sieht seine Konzeption einer Echtzeit-Simulationsumgebung als Vorarbeit für ein zu entwerfendes, rechnerbasiertes Qualitätsmanagement für numerische Steuerungen.

Bewertung: Das angestrebte Konzept der Arbeit ist vergleichbar mit den Arbeiten in (Bender 1999). Damals wurde eine Komplettlösung aus Methoden und Werkzeug für die testorientierte Echtzeitsimulation für SPS entwickelt. Der Fokus bei Kreusch liegt auf NC-Steuerungssystemen. Für die gesamte Arbeit von Kreusch steht die Entwicklung der echtzeitfähigen Simulation im Vordergrund. Daher beschäftigt er sich nicht tiefergehend mit der Automatisierung des Tests und verschiebt diesen Aspekt in den Ausblick seiner Arbeit. Kreusch sieht eine Rechnerunterstützung beim Steuerungstest als unabdingbar an, beschränkt diese aber auf eine teilautomatisierte Unterstützung. Als Gründe nennt er zu komplexe Testfälle, weiterhin verweist er auf Situationen die besser durch die Intuition des Testers geprüft werden könnten (S. 92). Diese Aussagen Kreuschs werden darauf zurückgeführt, dass die Simulation eines Fertigungssystems zum damaligen Zeitpunkt nicht den Umfang besaß wie heute. Zudem ist das Testwerkzeug in Kreuschs Konzept nicht derart gestaltet, dass Testfälle modelliert werden können. Der Ablauf eines Tests wird bei Kreusch über das modellierte Verhalten in der Simulation gesteuert und durch manuelle Testdurchführung ergänzt. Der Testrechner mit dem Testwerkzeug konfiguriert automatisch die Steuerung und stößt den Simulationslauf (S. 109) an. Im Testablauf kann der Testrechner Aufzeichnungsdaten aus dem Echtzeitbetrieb empfangen. Eine automatisierte Interaktion des Testrechners mit der NC und der Simulation findet nicht statt. Deshalb muss die Auswertung von Simulationsläufen am Ende separat und manuell durchgeführt werden.

(Wälde 2002)

Kurzzusammenfassung: (Wälde 2002) entwirft in seiner Dissertation „Sicherstellung der Softwarequalität von Anwendungsmodulen und Systemplattformen in offenen Steuerungssystemen“ ein durchgängiges Qualitätssicherungskonzept für Softwaremodule in offenen Steuerungssystemen. Sein Ziel ist es, dabei die Inbetriebnahmezeit zu verkürzen, als auch die Qualität der Software zu steigern. Als Schwerpunkt setzt Wälde, ein produktunabhängiges Verfahren zur herstellerübergreifenden Prüfung von Programm- und Dienstmodulen zu entwerfen. Zunächst stellt Wälde Anforderungen zu Qualitätsmerkmalen, Prüfverfahren, Prüfmethodik und Prüfling auf, welche insbesondere für offene Systeme gelten. Das Testsystem entwickelt er so, dass die Beaufschlagung des Prüflings mit Testdaten automatisiert passiert. Der Testfall kann händisch definiert werden. Als zweite Möglichkeit werden Testfälle aus vorab definierten allgemeinen Testfällen, welche den Qualitätsmerkmalen entsprechen, für das Zielmodul generiert. Wälde schafft mit seiner Arbeit ein durchgängiges und anwendbares Konzept für den Modultest von Softwaremodulen in offenen Steuerungssystemen. Realisiert hat Wälde sein Konzept anhand der OSACA (Open System Architecture for Controls with Automation Systems) Steuerungsplattform (siehe Brecher et al. 2010) auf zwei Zielsystemen.

Bewertung: Die Arbeit von Wälde ist ein Konzept für den Modultest an Steuerungssystemen und daher für den System- und den Abnahmetest von Fertigungssystemen ungeeignet.

(Bürger 2005)

Kurzzusammenfassung: Die Arbeit von (Bürger 2005) „Durchgängige analytische Qualitätssicherung für numerische Steuerungssoftware“ leistet einen Beitrag zur Qualitätssicherung von Software in industriellen Steuerungssystemen. Dazu stellt Bürger zunächst verschiedene Qualitätsmerkmale für numerische Steuerungen auf. Im Anschluss untersucht und bewertet er gängige Verfahren und Werkzeuge der Qualitätssicherung von Software. Weiterhin analysiert er die Fehlerentstehung in den einzelnen Entwicklungsphasen der NC-Software. Ziel ist die Entwicklung eines durchgängig anwendbaren testenden Prüfverfahrens,

bei dem er eine Mischung aus sinnvollem Konfigurationsmanagement der Softwareversionen, passende Auswahl an Testverfahren und Testumgebungen und eine Generierung von Testfällen kombiniert. Er zeigt je ein Konzept für die Umgebung zum Modultest und ein Konzept für die Umgebung zum Systemtest. Bei der Realisierung an einem Medizinrobotersystem setzt Bürger sechs seiner neun definierten Prüfverfahren mehr oder weniger vollständig ein.

Bewertung: Beim Systemtest beschränkt sich Bürger auf das System der numerischen Steuerung ohne dessen Peripherie, ein gesamtes Fertigungssystem, zu berücksichtigen. Auf eine HiL Simulation verzichtet Bürger explizit, den sehr hohen Modellierungsaufwand und ein unzureichender Modellumfang nennt er als Gründe. Die Portierbarkeit des Prüfansatzes auf andere Zielsysteme ist nicht gegeben.

(Wünsch 2008)

Kurzzusammenfassung: Die Dissertation von Wünsch (Wünsch 2008) „Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme“ beinhaltet auf den Seiten 134 – 141 auch Gedanken zur Automatisierung von Testabläufen an einem Hardware-in-the-Loop System. Wünsch begründet die Notwendigkeit zur Automatisierung mit steigenden Testaufwänden durch Regressionstests bei konfigurierbaren modularen Produktionssystemen. Wünsch nennt als Quellen für die Erstellung von Testszenarien die Einzelfunktionen aus den Anforderungen aus dem Lastenheft, im Gut- sowie im Schlechtfall, und Schlechtfälle aus dem Serviceumfeld. Neben den Einzelfunktionen des Produktionssystems möchte Wünsch auch die Buskommunikation prüfen, indem er hier Gut- und Schlechtfälle in der Kommunikation von Busteilnehmern skizziert. Beim Skizzieren der Architektur einer Testautomatisierungslösung zeigt er die Schwierigkeit meist proprietärer Schnittstellen in einem industriellen Steuerungssystem auf.

Bewertung: Das von Wünsch vorgestellte Konzept zeigt rudimentär einen Ansatz bezüglich Aufbau, Handhabung und Feinheiten bei der Testautomatisierung auf. Er zeigt keine Details der Lösung und bleibt auf reinkonzeptioneller Ebene. Es bleiben viele offene Fragen, z. B. wie werden die Testfälle erstellt und ausgewertet? Welche Daten und Aktionen müssen mit dem HiL System ausgetauscht

werden? Welche Aspekte des Fertigungssystems sollen getestet werden können?

(Herfs 2010)

Kurzzusammenfassung: Die Dissertation von (Herfs 2010) „Modellbasierte Software in the Loop Simulation von Werkzeugmaschinen“ geht in Teilen auf das Forschungsprojekt Ramp-Up/2 (Lohse et al. 2007) zurück. Der Beitrag von Herfs hat zum Ziel, eine höhere Qualität der Steuerungsprogramme zu Beginn der Inbetriebnahme von Fertigungssystemen sicherzustellen. Herfs erarbeitet dazu einen modellbasierten Ansatz zum Entwickeln von Steuerungsprogrammen. In den mittels UML modellierten Modellen sind Informationen (Modellkomplexe) bzgl. Struktur, Verhalten, Varianten, Abläufen und Tests enthalten. Herfs entwickelt dazu ein domänenspezifisches Sprachwerkzeug zur Modellierung. Aus den Modellen soll im Anschluss eine vollständige SiL Simulation einer Werkzeugmaschine abgeleitet werden können. Der Modellkomplex „Test“ wird von Herfs nicht in gleicher Weise als UML-Diagramm gezeigt und erläutert, wie es bei den übrigen Modellkomplexen der Fall ist. Herfs Konzept zur Erstellung von Testfällen sieht vor, die modellierten Gutfälle der mechatronischen Komponenten einer Werkzeugmaschine durch Äquivalenzklassenbildung, Grenzwertanalysen und Zufallstests zu erzeugen. Dieses Konzept wird aber nicht weiter betrachtet oder in der Realisierung umgesetzt. Zur optimalen Nutzung des SiL Systems, ohne Expertenwissen über die spezifischen Softwarewerkzeuge und Schnittstellen im SiL System besitzen zu müssen, konzipiert Herfs einen Hochlaufassistenten für den automatisierten Start des SiL Betriebs.

Bewertung: Die Arbeit von Herfs konzipiert eine Lösung zum Test an einem SiL System, welches jedoch nach Anforderungen in dieser Arbeit weniger Aussagekraft gegenüber einer HiL Simulation hat. SiL ist nicht zur Abnahme von Fertigungssystemen einsetzbar, da Laufzeit- und Kommunikationseigenschaften bzgl. der später eingesetzten Hardware nicht vorab berücksichtigt werden können. Der Ansatz einer Testfallgenerierung aus ein und demselben Modell, wie es zur Simulation genutzt wird, widerspricht der Intention des Tests. Schlussendlich erlaubt das SiL System Herfs nur die manuelle Eingabe von Testabläufen. Die

Testdurchführung geschieht nur mittels vorab definierter Abläufe und beschränkt sich in ihrem Ausmaß auf das vorab in die Simulation eingespielte Verhalten. Die Testdurchführung, wie in dieser Arbeit gefordert, muss mittels Komparator, Zugriff auf HMI, Zugriff in die Simulation reaktiv ihren Testablauf anpassen können. Eine automatisierte Testauswertung und Erstellung eines Testberichts leistet der Ansatz von Herfs nicht. Die Herstellerunabhängigkeit des Ansatzes von Herfs ist nur insoweit gegeben, als ein virtueller NC-Kern als separates Softwarepaket vom Hersteller zur Verfügung stehen muss.

(Hametner 2013)

Kurzzusammenfassung: (Hametner 2013) entwickelt in seiner Dissertation „Test Driven Software Development for Improving the Quality of Control Software for Industrial Automation Software“ einen Ansatz zum testgetriebenen Entwickeln von Steuerungsprogrammen für die industrielle Automatisierung. Hametner motiviert seine Arbeit ausgehend von der Tatsache, dass es keinen vorherrschenden Softwarequalitätsprozess beim Testen und auch keine Werkzeuge dafür gibt. Zur Erstellung eines Konzepts arbeitet Hametner die gängigen Methoden und Verfahren aus der Softwaretechnik heraus. Für den Entwurf der Software wählt der Autor die Beschreibungssprache UML. Da diese maschinenverarbeitbar ist, generiert er später aus UML-Diagrammen Testfälle. Weiterhin wird ein neues Komponentendesign für die Entwicklung von Steuerungsapplikationen vorgestellt, um diese besser testbar zu machen. Aus der Analyse von Testmethoden aus der Softwaretechnik wählt Hametner drei Verfahren („Keyword-Driven Testing“, „Unit Testing“ und „Model-Based Testing“) aus. Als Ebenen der Anwendbarkeit in der Verifikationsphase evaluiert Hametner, je nach Verfahren, alle Ebenen: Unit-, Integration- und Systemtestebene für SPS-Software.

Bewertung: Hametners Ansatz beschränkt sich auf den Test der SPS-Software. Mit seiner Lösung unterstützt er den Entwickler beim testgetriebenen Entwickeln. Der Inbetriebnehmer, der das gesamte Steuerungssystem aus SPS, NC und HMI mit der Maschine oder Anlage virtuell in Betrieb nehmen muss, kann diese Lösung nicht einsetzen.

(Ulewicz 2018)

Kurzzusammenfassung: Die Dissertation von (Ulewicz 2018) „Test coverage assessment for semi-automatic system testing and regression testing support in production automation“ beschreibt ein Verfahren zur Qualitätsverbesserung der Systemtests von SPS-Software. Dabei erarbeitet Ulewicz drei Teilaspekte: Erstens ein halbautomatisiertes Vorgehen zur Testdurchführung. Zweitens, ein Verfahren zur Abschätzung der Testabdeckung und drittens ein weiteres Verfahren zur Priorisierung von Testfällen. Die Arbeit von Ulewicz leistet ihren Beitrag in der qualitativen Auswahl der Testfälle und macht die Testabdeckung in der Software quantifizierbar. Weiterhin wird die Testart des Regressionstests methodisch und mit Werkzeugen unterstützt. Bei der Umsetzung seiner Konzepte erweitert Ulewicz den CODESYS Test Manager für seinen Prototypen.

Bewertung: Ulewicz beschränkt sich auf den Test von SPS-Software alleine. Das Testverfahren ist ein Whitebox-Verfahren welches keine Aussage über das Verhalten des gesamten Steuerungssystems zulässt. Es wird nicht gegen eine simulierte Maschine oder Anlage getestet, so dass keine Aussage über das Verhalten im Betrieb gemacht werden kann und die virtuelle Inbetriebnahme nicht direkt unterstützt wird. Bei der Realisierung beschränkt sich Ulewicz auf das CODESYS System von 3S, sodass keine Herstellerunabhängigkeit unterstellbar ist.

Zusammenfassung der wissenschaftlichen Arbeiten

Zur Bewertung der wissenschaftlichen Arbeiten wurden die Anforderungen aus Kapitel 3 genutzt. Zusätzlich wurde geprüft, ob die jeweilige Arbeit eine prototypische Realisierung des eigenen Ansatzes zeigt. Damit soll die Aussagekraft der Arbeit eingestuft werden. Abbildung 20 fasst die Bewertung der wissenschaftlichen Arbeiten gegen die Anforderungen in einer Matrix zusammen. Keine der untersuchten Arbeiten erfüllt alle gestellten Anforderungen. Zudem wird ersichtlich, dass für die Anforderungen /A3/ Herstellerunabhängigkeit, /A4/ Geeignete Modellierung und /A8/ Dauertest keine zufriedenstellende Lösung im Stand der Wissenschaft existiert.

	Kohring 1993	Bender 1999	Kreusch 2002	Wälde 2002	Bürger 2005	Wünsch 2008	Herfs 2010	Hametner 2013	Ulewicz 2018
/A1/ Testautomatisierung	○	◑	◑	●	○	◑	◑	◑	◑
/A2/ RT-HiL	●	●	●	○	◑	●	○	○	◑
/A3/ Herstellerunabhängig	○	○	○	◑	○	○	◑	○	○
/A4/ Geeignete Modellierung	○	○	○	○	○	○	◑	○	○
/A5/ Fehlerfalltest	◑	◑	◑	○	○	◑	●	○	◑
/A6/ Wiederverwendung	○	○	○	○	◑	○	○	○	●
/A7/ Testtreiber	○	○	●	○	○	○	◑	○	○
/A8/ Dauertests	○	○	○	○	○	○	○	○	○
/A9/ Komparator	○	●	○	○	○	○	◑	●	●
Prototypische Realisierung	◑	●	◑	●	●	○	●	●	●

Legende
○ … Anforderung nicht erfüllt
◑ … Anforderung zu Teilen erfüllt
● … Anforderung vollständig erfüllt
RT-HiL … Real-Time Hardware-in-the-Loop

Abbildung 20 Schnellübersicht der Bewertung mittels Harvey Balls von Forschungsarbeiten gegen die Anforderung

4.2 Kommerzielle Lösungen für Testwerkzeuge zur Testautomatisierung

In der Industrie zum Einsatz kommen häufig Eigenentwicklungen oder speziell für einen Kunden auf seine Werkzeugkette zugeschnittene Lösungen (ITQ GmbH

2016; ITQ GmbH 2016). Diese reichen dabei meist nicht über eine Skriptingumgebung zur Erstellung der Testabläufe hinaus, bspw. (Stetter et al. 2008; Frank et al. 2016; Vogel Communications Group 2016). Drei erwähnenswerte Arbeiten aus der Industrie werden im Folgenden näher erläutert.

(Menzel et al. 2004; Stöppler et al. 2005)

Aus Veröffentlichungen, die über ein Jahrzehnt zurückliegen und maßgeblich auf die Fa. Siemens AG zurückgehen, sind Konzepte zur Automatisierung der Simulation zu VIBN Zwecken bekannt (Menzel et al. 2004; Stöppler et al. 2005). Die Autoren beschreiben die Kopplung einer Hardware-in-the-Loop Simulation mit einem Testautomat. Der Testautomat beschränkt sich auf ein Konzept eines geskripteten Ablaufs, welcher in der Lage ist, die Simulation zu beeinflussen.

Bewertung: Die Veröffentlichungen von Menzel und Stöppler zeigen erstmalig eine Kopplung zwischen einem HiL System einer Werkzeugmaschine und einem Testautomaten. Die Ausführungen in beiden Veröffentlichungen beschränken sich jedoch auf jeweils einen kurzen Absatz von wenigen hundert Wörtern mit derselben schematischen Darstellung. Da bis heute keine weiteren Arbeiten der Autoren oder ein kommerzielles Werkzeug der Fa. Siemens AG erschienen sind, kann diese Arbeit aufgrund ihrer Unvollständigkeit nur schlecht bewertet und verglichen werden. Sie findet trotzdem Erwähnung, um klar zu machen, dass die Idee einer Testautomatisierung gekoppelt mit einer Simulation im Maschinen- und Anlagenbau keine Neuheit per se darstellt. Die ganzheitliche Konzipierung eines neuen Testprozesses inklusive passender Werkzeuge allerdings schon.

(3S-Smart Software Solutions GmbH 2017)

CODESYS Test Manager ist ein Werkzeug, um in der CODESYS Entwicklungsumgebung für SPS-Applikationen UnitTests mit den eigenen Programmbausteinen durchzuführen. Auch zur Nutzung nach dem Test-Driven Development ist das Werkzeug geeignet. Mittels des Plugins für die Entwicklungsumgebung von der 3S-Smart Software Solutions GmbH können Tests grafisch parametriert werden und ein Testbericht generiert werden.

Bewertung: Das Plugin für die CODESYS IDE erfüllt seinen Zweck als UnitTest-Werkzeug, welches bei der Entwicklung von SPS-Programmcode unterstützt.

Eine Übertragbarkeit auf NC-Steuerungssysteme oder für die Entwicklung bei anderen Herstellern ist nicht gegeben. Eine Nutzung mit einer Simulation ist nicht vorgesehen.

(ISG Industrielle Steuerungstechnik GmbH 2018)

Die Grundlagen des CAST Werkzeugs *ISG-dirigent* der Fa. ISG wurden in einem Forschungsprojekt des Instituts für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen und zusammen mit der Fa. ISG erarbeitet (Krebser et al. 2018; Kübler et al. 2018). Die wissenschaftliche Grundlage sowie die Konzepte und Methoden für ISG-dirigent werden in dieser Arbeit beschrieben und im Rahmen einer demonstrativen Implementierung validiert. Somit stellt diese Arbeit den Stand der Technik für ISG-dirigent dar.

Fazit zu den kommerziellen Lösungen

Bei den kommerziellen Lösungen sind keine Angebote bekannt, die ansatzweise den automatisierten Test für den System-, Abnahme- und Regressionstest von Fertigungssystemen ganzheitlich unterstützen. Anwendungsspezifische Lösungen aus mechatronischen Disziplinen existieren, genauso wie Lösungen für die reine Softwaretechnik.

4.3 Zusammenfassung, Handlungsbedarf und Ziele dieser Arbeit

In Kapitel 4 wurden die zum aktuellen Zeitpunkt bekannten Arbeiten aus Wissenschaft und Technik im Themenkomplex Testautomatisierung für Fertigungssysteme aufgezeigt und bewertet. Bei der Bewertung wurden die Anforderungen an eine gesuchte Lösung aus Kapitel 3 herangezogen. Die Ergebnisse der Untersuchung zum Stand der Wissenschaft und Technik zeigen ein klares Defizit für das ganzheitliche, methodische und aufwandsarme Vorgehen beim System- und Abnahmetest der Automatisierungssoftware für Steuerungssysteme auf. Es konnte gezeigt werden, dass kein Ansatz existiert, der die aktuellen und kommenden Herausforderungen bei der Entwicklung von automatisierten Fertigungssystemen erfüllt.

Abbildung 21 zeigt die Einordnung der in Kapitel 4.1 beschriebenen Forschungsarbeiten. Der Zielbereich für diese Dissertation ergibt sich durch die entstehende Forschungslücke im oberen rechten Eck der Schnittmenge aus *Aussagekraft* und *Aufwand* des Testkonzepts. Die Kombination aus höchstem Automatisierungsgrad und höchster Aussagekraft ist bisher ohne Beitrag.

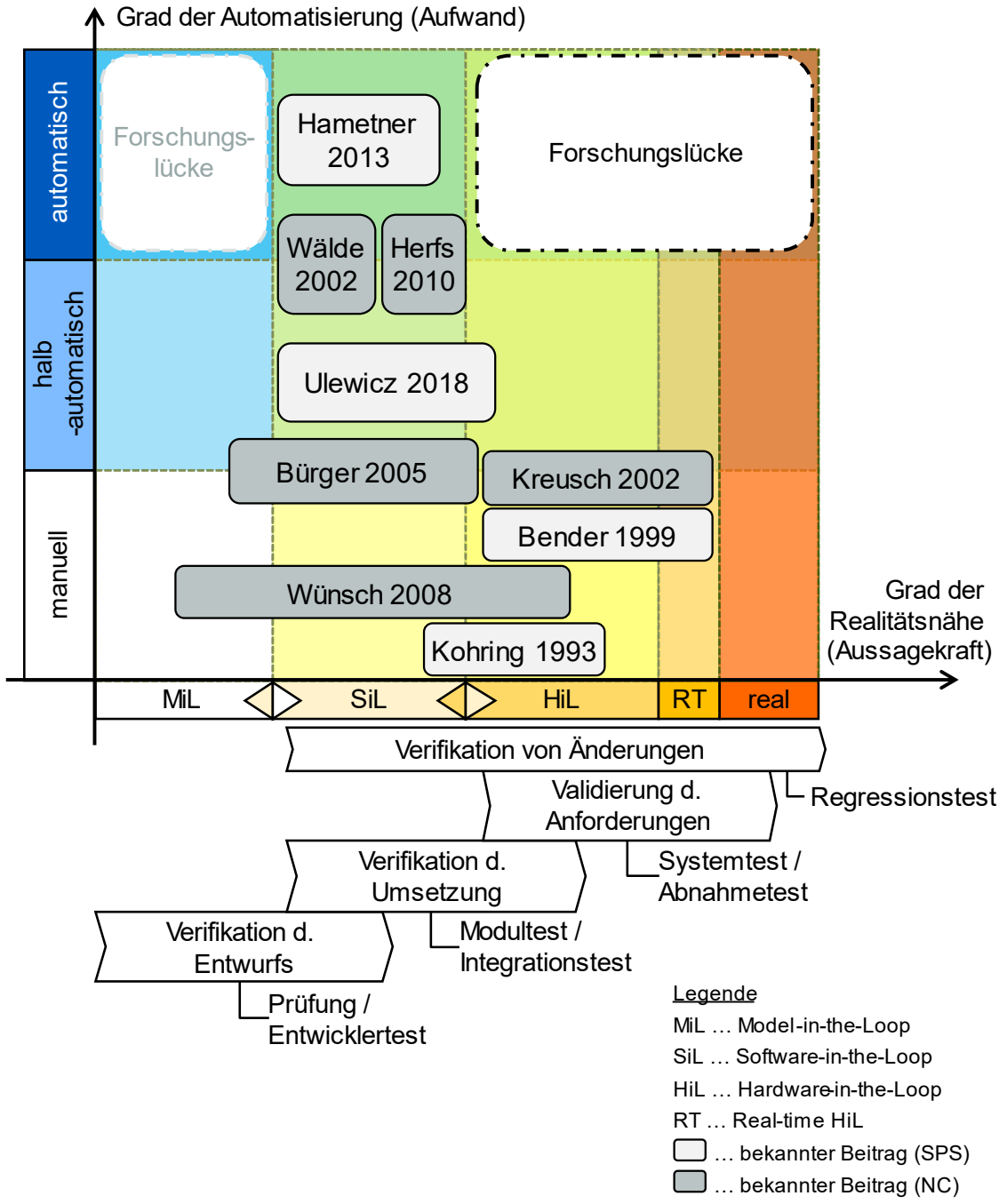


Abbildung 21 Einordnung der bekannten Forschungsarbeiten zwischen Aufwand und Aussagekraft der Testmethodik; Identifikation der Forschungslücke

Testautomatisierung konnte als adäquate Methode zur Verbesserung von System-, Abnahme- und Regressionstests identifiziert werden. Über eine reine Automatisierung der Testdurchführung hinaus ist das **Ziel dieser Arbeit**, eine ganzheitliche Testautomatisierung als Methodik zu konzeptionieren, umzusetzen und zu evaluieren. Durch die Verbesserung des Testprozesses soll in kausaler Folge auch die Qualität der Software, insbesondere die der Steuerungsprogramme, verbessert werden. Bei der Erarbeitung und Realisierung sind die neun definierten Anforderungen zu berücksichtigen.

Die zu **erwartenden Vorteile** einer ganzheitlichen Testautomatisierung sind eine Aufwands- und Kostensenkung im virtuellen Abnahmeprozess bei gleichzeitig gestiegener Qualität im Testprozess für Fertigungssysteme. Die Vorteile können damit an den allgemeinen Vorteilen einer Testautomatisierung orientiert werden (teilweise aus Gühmann et al. 2002, S. 514–515):

- Minimierung der Ressourcen zur Testdurchführung
- Reproduzierbarkeit bei den Testergebnissen
- Vereinfachung der Regressionstests durch Formalisierung der Testabläufe
- Einheitliche Dokumentation der Abnahme
- Minimierung der Fehler während der Testdurchführung
- Erhöhung der Aussagekraft von Testergebnissen
- Steigerung des Testumfangs je Softwarekomponente
- Treffen von sporadischen Fehlern durch Dauertests
- Wiederverwendung von Testkomponenten
- Unterstützung bei der Erstellung von Testabläufen
- Weiteres Frontloading und Simultaneous Engineering in der Entwicklung von Fertigungssystemen

5 Voruntersuchungen zur ganzheitlichen Testautomatisierung und Grobkonzept

Die Anforderungen an ein neues Konzept für die virtuelle Inbetriebnahme von automatisierten Fertigungssystemen mittels ganzheitlicher Testautomatisierung wurden in Kapitel 3 definiert. Ausgehend von den Untersuchungen zum Stand der Technik und Wissenschaft gegen diese Anforderungen konnte in Kapitel 4 festgestellt werden, dass keine Lösung existiert, welche die Anforderungen zufriedenstellend erfüllt. Die aufgezeigten Defizite und die erarbeitete Forschungslücke werden mit den Inhalten in Kapitel 5, 6 und 7 behoben bzw. geschlossen. Dabei erarbeitet Kapitel 5 den Grobentwurf für die neue Methodik zur ganzheitlichen Testautomatisierung. Zur Klärung, wie das Konzept aussehen muss, werden Testtechniken auf ihre Eignung hin geprüft und Kategorien und Typen von Testwerkzeugen recherchiert. Weiterhin werden bestehende Schnittstellen von Steuerungssystemen für die Anbindung betrachtet und gängige Testfälle von Herstellern von Fertigungssystemen erhoben. Ergebnisse des Kapitels sind die Konzepte für die Integration von ganzheitlicher Testautomatisierung in den Entwicklungsprozess sowie die Grobarchitektur einer Testumgebung.

5.1 Integration von ganzheitlicher Testautomatisierung in das Vorgehensmodell im Maschinen- und Anlagenbau

(Bröhl et al. 1995) beschreiben das originale V-Modell zur Entwicklung von Software. Das V-Modell findet heute in der Softwareentwicklung von mechatronischen Systemen Anwendung (bspw. Dornseiff et al. 2001; Norm VDI 2206 2004; Bender 2005; Obstbaum et al. 2017). Für die Entwicklung des gesamten mechatronischen Systems wird meist eine Erweiterung des V-Modells vorgeschlagen (Norm VDI 2206 2004; Bender 2005 Kap. 3.3; bspw. Gräßler 2017). (Puntel Schmidt 2017, S. 60f) vergleicht in seiner Arbeit das Vorgehensmodell für mechatronische Produktentwicklung nach (Norm VDI 2206 2004) mit dem Vorgehen bei der Entwicklung von Fertigungssystemen nach (Norm VDI 4499 Blatt 2 2011). Er kommt dabei zu dem Schluss, dass eine Übertragbarkeit von mechatronischer

Produktentwicklung nur beschränkt und nicht ohne Paradigmenwechsel in den betroffenen Unternehmen möglich ist, insbesondere im Hinblick auf die Nutzung von Simulation. Ein grundlegender Unterschied in den Paradigmen ist, dass die Softwareentwicklung und Programmierung als Problemstellung gesehen wird, welche nach Abschluss der Mechanik- und Elektronikentwicklung durchgeführt wird (Bender 2005, S. 8).

Auch mittels VIBN wurde kein Paradigmenwechsel in der Entwicklung von Fertigungssystemen erreicht. Für die ganzheitliche Testautomatisierung wird daher weiterhin von einem Vorgehen, angelehnt an das sequentielle Wasserfallmodell nach (Royce 1970; Benington 1983), ausgegangen.

Die ganzheitliche Testautomatisierung integriert den Testprozess früher (Frontloading) und parallel (Simultaneous Engineering) in den Entwicklungsprozess, ganz im Sinne der Digitalen Fabrik (Norm VDI 4499 Blatt 1 2008), Abbildung 22. Der Wunsch nach einem Simultaneous Engineering beschäftigt Maschinenbauunternehmen bereits seit Längerem: Die Motivation liegt in der Beherrschung der Schnittstellenproblematiken zwischen den Disziplinen und dem Ermöglichen einer präventiven Qualitätssicherung (Bauer 1995, S. 41f).

Mittels ganzheitlicher Testautomatisierung soll der Testprozess gegenüber dem bisherigen Prozess durch diese Maßnahmen verbessert werden, siehe Abbildung 22:

- Für die Entwicklung, Einführung und Durchführung des neuen Testprozesses benötigt es separates Testpersonal, unabhängig vom aktuellen Entwicklungspersonal.
- Der Subprozess Testentwurf wird von Testdurchführung und Testauswertung entkoppelt, da er strukturiert in einem Werkzeug stattfindet.
- Der entkoppelte Testentwurfsprozess geschieht parallel und zeitlich früher im Vorgehensmodell.
- Testdurchführung und Testauswertung sind schneller beendet, da der Datendurchsatz mittels Automatisierung erhöht ist und Zeiten ohne Personal über Nacht und am Wochenende genutzt werden können.

Als Resultat des integrierten, neuen Testprozesses wird für die Fehlerbehebung und Optimierung am Fertigungssystem mehr Zeit bereitgestellt, siehe Δt in Abbildung 22.

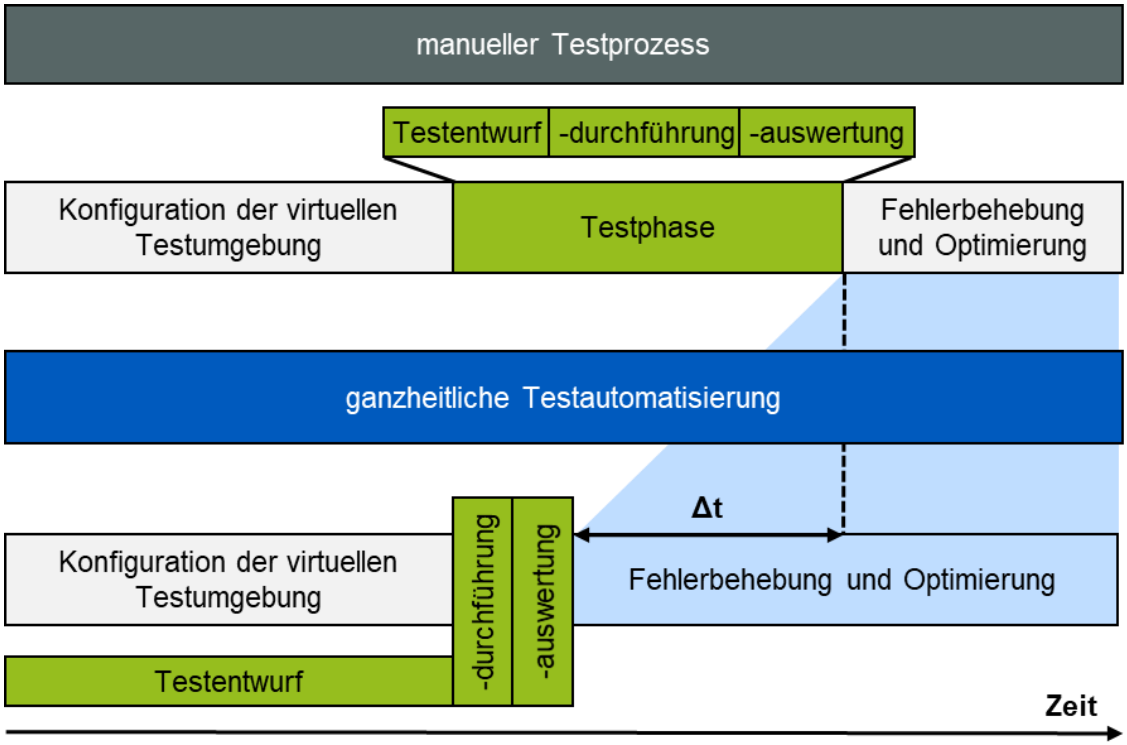


Abbildung 22 Vergleich beim manuellen Vorgehen in der Testphase mit der Methodik zur ganzheitlichen Testautomatisierung

Der Testentwurf kann durch separates Personal und eine Werkzeugunterstützung parallel begonnen sowie parallel zur Testdurchführung und -auswertung weitergeführt werden, da die Durchführung und Auswertung automatisiert sind. Im Vergleich zum aktuellen Verifikations- und Validierungsprozess ist es mittels ganzheitlicher Testautomatisierung möglich, häufiger Tests, bzw. mehr Testfälle insgesamt, durchzuführen. Dies kann zur Erhöhung der Testtiefe und Testbreite genutzt werden. Neben der neuen Integration in die Entwicklung ist auch die vollautomatische Testdurchführung und -auswertung dafür verantwortlich, da diese einen 24/7 Betrieb (Dauertest) ermöglicht. Zum Dauertest entstehen eine Großzahl an Testprotokollen und dynamischen Testergebnisdaten. Die reine automatische Testdurchführung führt folglich nicht zu einer Einsparung von Zeitaufwand, da ansonsten bei der Testauswertung, durch eine erhöhte manuelle Datenauswertung, zusätzlich Zeit und Ressourcen geplant werden müssen.

Für die Steuerungsprogramme und das gesamte Fertigungssystem entsteht im neuen Vorgehen der Entwicklung, durch ganzheitliche Testautomatisierung eine längere Zeitdauer zum Beheben von Fehlern und Einbringen von Optimierungen. Zudem stehen die strukturiert erstellten Testfälle aus dem Testentwurf in jeglichen Retrofit- oder Regressionsszenarien wieder zur Verfügung.

5.2 Auswahl von Testtechniken für die ganzheitliche Testautomatisierung

Im Rahmen der Voruntersuchungen wird in diesem Kapitel erarbeitet, welche aus der Literatur bekannten Testtechniken für die ganzheitliche Testautomatisierung von Fertigungssystemen sinnvoll anzuwenden sind. Dazu werden auch die im aktuellen, manuellen Vorgehen üblichen Techniken bewertet.

Unter Testtechniken werden Testarten, Testverfahren und Testmethoden aus der Softwaretechnik verstanden. Angewandt werden die Testtechniken auf „Testobjekte“, die Artefakte einer Entwicklung, welche jeweils einer Prüfung bzw. einem Test unterzogen werden. Die „Testart“ drückt aus, zu welcher Zeit in der Entwicklung getestet wird. Das in Kapitel 2.3 gezeigte V-Modell (Abbildung 16) macht die Zuordnung von Testart zu Entwicklungsphase deutlich. Die „Testmethode“ klassifiziert Art und Weise beim Vorgehen einer Testart, siehe Zuordnung mittels Linien in Abbildung 23. Mittels eines „Testverfahren“ wird die Erstellung von Testfällen und der benötigten Testdaten systematisiert. Für die neue Methodik der ganzheitlichen Testautomatisierung wird in Abbildung 23 mittels grüner Umrandungen dargestellt, welche Testtechniken geeignet sind. Grundlegend wurde im Kapitel 3 zu den Anforderungen bereits festgelegt, dass, erstens, die Software, vgl. „Programmcode“ in Abbildung 23, das Testobjekt ist. Zweitens, die Testarten Systemtest, Abnahmetest und Regressionstest mit der ganzheitlichen Testautomatisierung adressiert werden und diese dynamische Blackbox-Testverfahren sind.

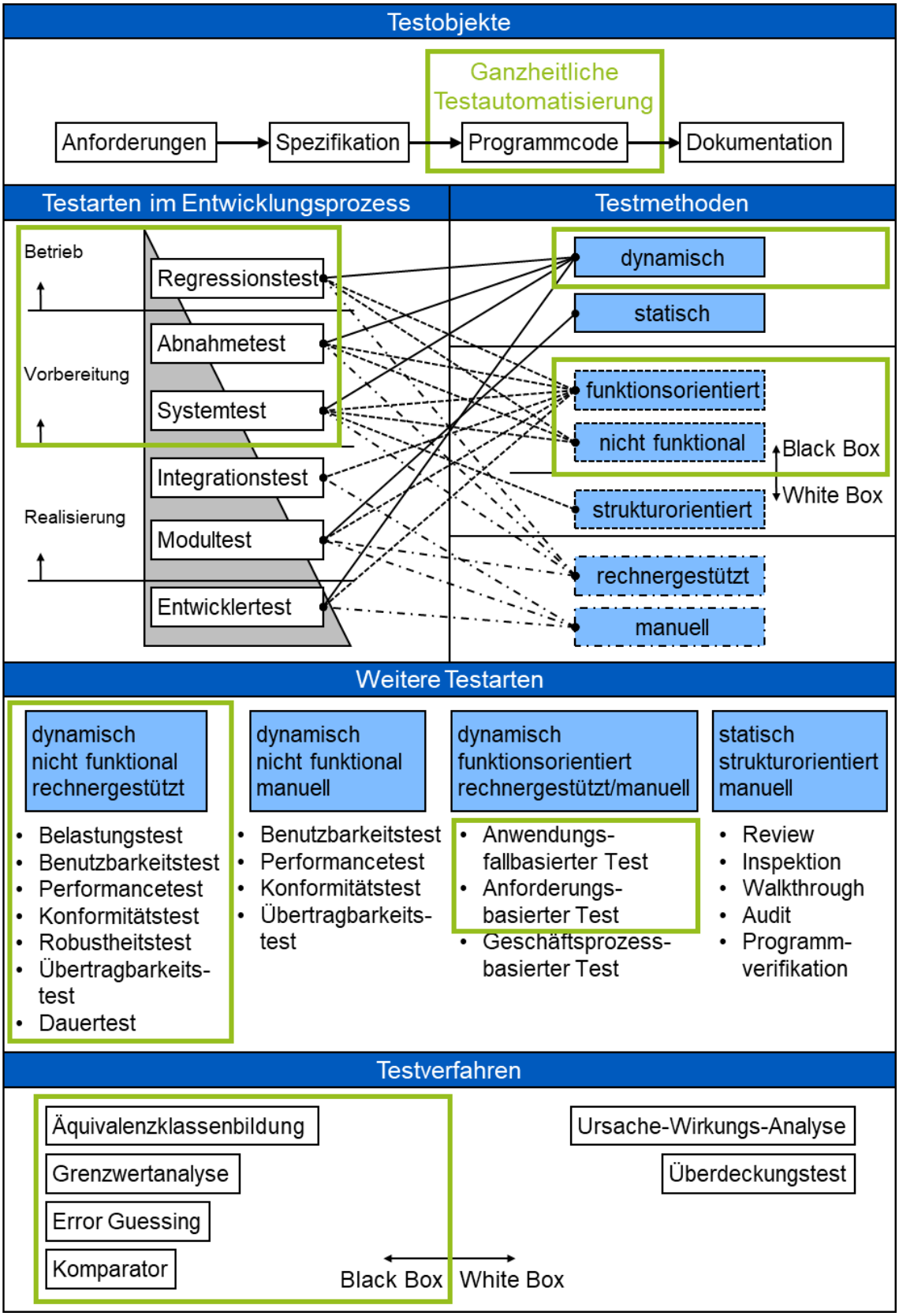


Abbildung 23 Übersicht zu Testtechniken: Testarten, Testmethoden und Testverfahren (eigene Darstellung von Informationen aus (Helfen et al. 2006; Balzert 2008; Liggesmeyer 2009; Myers et al. 2012; Witte 2016))

Grundsätzlich erscheinen Testarten, welche eine hohe Rate bei der Verarbeitung von Daten benötigen, sowohl bei deren Eingabe als auch bei deren Ausgabe, für eine rechnergestützte Testautomatisierung als geeignet: „Regressionstest“, „Belastungstest“, „Robustheitstest“ und „Dauertests“. „Anwendungsfall- und anforderungsbasierte Tests“ werden als funktionsorientierte Tests für System-, Abnahme- und Regressionstest eingesetzt. Für anwendungsfallbasierte Tests müssen z. B. in Use Cases beschriebene Sequenzen durch mindestens einen Testfall abgedeckt werden (Hoffmann 2013, S. 189). Statische Testarten oder „geschäftsprozessbezogene“ Tests sind nicht der Einsatzzweck von ganzheitlicher Testautomatisierung.

Bereiche der Eingabewerte von Variablen, die beim Test dasselbe Systemverhalten zur Folge haben, werden in „Äquivalenzklassen“ zusammengefasst. Jeder Äquivalenzklasse werden repräsentative Werte zugeordnet. Unter der Annahme, dass sich das Programm für alle Kombinationen dieser Werte gleich verhält, müssen lediglich die Kombinationen von gewählten Repräsentanten der Klassen getestet werden. (Liggesmeyer 2009) Dieses Testverfahren ist damit für die ganzheitliche Testautomatisierung geeignet, da es die Anzahl an Tests methodisch reduziert. Die Auswahl von gewählten Repräsentanten kann z. B. mittels einer Grenzwertanalyse erfolgen. Die „Grenzwertanalyse“ ist ein Beispiel für eine fehlerorientierte Testtechnik. Sie basiert auf der Annahme, dass Fehler an Bereichsgrenzen wahrscheinlich sind“ (Liggesmeyer 2009, S. 511).

Aktuell werden im Testprozess von Fertigungssystemen kein einheitlicher und qualitätsbewusster Testentwurf und Testrealisierung durchgeführt. Testentwurf, Testrealisierung und Testdurchführung finden teilweise in einem Schritt statt und werden von der gleichen Rolle durchgeführt: Der Tester führt das Steuerungsprogramm aus und gibt dabei – geleitet vom Ablauf – spontan Daten vor. Die Ergebnisse der Ausführung betrachtet er kurz und entdeckt dort eventuell Fehler. Die Datenauswahl basiert dabei auf dem persönlichen Erfahrungswissen. Zudem existierten nach einem solchen Test selten archivierbare Testergebnisse oder ein Testbericht. Die Literatur bezeichnet einen solchen Test als *Wegwerftest* (Ludewig et al. 2010, S. 490). Die Qualität eines solchen Tests hinsichtlich seiner Aussagekraft ist schlecht und nicht nachvollziehbar. Ein wichtiges Verfahren im

aktuellen, manuellen Testprozess stellt das *Error Guessing* (Thaller 2000, S. 88) dar: Die persönliche Erfahrung eines Testers mit dem Gesamtsystem oder den Grenzen der Programmiersprache wird genutzt, um Stellen im Programmcode zu identifizieren, welche mutmaßlich Fehler enthalten könnten. Mittels Error Guessing erstellte Testfälle sind auch mittels Testautomatisierung weiterhin sinnvoll einsetzbar und profitieren bspw. in Regressionstests von der aufwandsarmen Testdurchführung und -auswertung.

Mittels ganzheitlicher Testautomatisierung werden die Tests werkzeugunterstützt erstellt und ausgeführt. Dieses Vorgehen erzwingt die Digitalisierung der Spezifikation von Testfällen und Testdaten und ein Auseinandersetzen mit Qualitätsmerkmalen und systematischem Testentwurf. Die Einführung von Qualitätsmerkmalen für die Erstellung von Software kann dazu genutzt werden, gezielt nach Merkmalen Testfälle zu erstellen. Die Auswahl von Qualitätsmerkmalen und die Quantifizierung deren Erfüllung definieren damit direkt, mit welcher Testart das Erreichen der Ziele in der Qualität geprüft werden kann. (Norm ISO/IEC/IEEE 29119-1 2013, S. 17) stellt diesen Sachverhalt grafisch dar. Mit diesem Vorgehen kann ein qualitätsgetriebenes Erstellen von Testfällen ermöglicht werden. Darüber hinaus ermöglicht die automatisierte Testdurchführung und -auswertung ein effizienteres und fehlerfreies Testen (Myers et al. 2012, S. 219).

5.3 Auswahl von Typen von Testwerkzeugen

Zur Unterstützung der benannten Testarten, Testmethoden und Testverfahren aus dem vorherigen Kapitel existieren verschiedene Testwerkzeuge. Im Folgenden werden Kategorien von Testwerkzeugen und einzelne Typen der Kategorien erläutert. Anschließend wird eine passende Auswahl für das CAST Werkzeug getroffen, um die Anforderung /A1/ (CAST Werkzeug) zu erfüllen.

Grundsätzlich können Testwerkzeuge in drei Kategorien eingeteilt werden (Korotkiy 2010, S. 37):

- Testmanagementwerkzeuge
- Testspezifikationswerkzeuge
- Testdurchführungswerkzeuge

Testmanagementwerkzeuge (TMW) dienen der Verwaltung von spezifizierten Testfällen und erfolgten Testdurchführungen. Mit dem TMW können auch Ressourcenplanungen für die Testdurchführung erledigt werden. Weitere Metriken, die erfasst werden können, sind Anzahl an Testdurchläufen und Status der Testergebnisse. Einige Werkzeuge erlauben auch die Verknüpfung mit einem Anforderungsmanagement, dem Fehlermanagement oder der Versionsverwaltung der Softwareentwicklung. (Korotkiy 2010, S. 37–38; Spillner et al. 2012, S. 211)

Ein **Testspezifikationswerkzeug (TSW)** unterstützt bei der Erstellung von Testfällen (Testentwurf) und Testdaten (Testrealisierung). Es werden Testgeneratoren und Testeditoren unterschieden (Korotkiy 2010, S. 38). Während die Testeditoren als TSW die manuelle Spezifikation unterstützen, werden mit Testgeneratoren automatisch Testfälle und Testdaten aus verschiedenen Quellen generiert. (Spillner et al. 2012, S. 213–214) nennen als Quellen Datenbanken, Programmcode, Schnittstellen und die Spezifikation.

Testdurchführungswerkzeuge (TDW) übernehmen die Arbeit eines Testers. Das bedeutet, sie führen die Schritte eines Testfalls aus, zeichnen Daten während der Testdurchführung auf und werten Ergebnisse ggfs. automatisch aus. Je nach Ausbaustufe des TDW können auch Testberichte automatisch generiert werden.

Den drei Kategorien TMW, TSW und TDW können Typen von Testwerkzeugen zugeordnet werden. Konkretere Typen finden sich in der Literatur (Thaller 2002, S. 265–268; Utting et al. 2007, S. 22–26; Ludewig et al. 2010, S. 498; Spillner et al. 2012, S. 209–222). Abbildung 24 zeigt eine Übersicht der im Rahmen dieser Arbeit betrachteten Testwerkzeugtypen und eine Einordnung nach Testwerkzeugkategorie und Testart. In der ganz rechten Spalte ist die Eignung des Werkzeugtyps für die neue Testmethodik abgetragen. Die Eignung wurde anhand der Anforderungen aus Kapitel 3 festgelegt. Die Auswertung nach der Tabelle in Abbildung 24 ergibt eine Eignung von Testskripte/-treiber, Komparator, datengetriebene Werkzeuge, schlüsselwortgetriebene Werkzeuge und Spezialanwendung

wie z. B. Dauertest. Wohingegen Werkzeuge für Capture & Replay, Testabdeckung, statische Analyse und Debugger ungeeignet sind. Testfallgenerator, Testdatengenerator erfüllen die aufgeführten Anforderungen teilweise.

Testwerkzeugtypen für die Testabdeckung, statische Analyse und Debugging unterstützen Testmethoden und Testverfahren des Whitebox Testings. Diese Methoden und Verfahren kommen bei den anvisierten Testarten System-, Abnahme- und Regressionstest nicht zum Einsatz.

Testwerkzeuge für Capture & Replay zeichnen Benutzungsreihenfolgen, meist an grafischen Benutzerschnittstellen, auf und geben diese auf Kommando automatisch wieder. Für schnelle und sichtbare Erfolge in der Testautomatisierung von Systemen mit Benutzeroberfläche sind diese Werkzeuge geeignet. Bei der neuen Methodik zur Testautomatisierung werden Wiederverwendung (/A7/), Herstellerunabhängigkeit (/A4/) und Zugriff auf (nicht grafische) Programmierschnittstellen mittels Testtreiber (/A3/) für die Testfallspezifikation gefordert. Daher ist das Prinzip des Capture & Replay Werkzeugs ungeeignet.

Werkzeugtyp	Kategorie			Testart				Eignung
	TMW	TSW	TDW	MT	IT	ST	AT	
Capture & Replay		x	x			x	x	○ wg. /A7/
Testskripte/-treiber		x	x	x	x	x	x	● /A1, A3, A7/
Komparator	x		x	x	x	x	x	● /A9/
Testabdeckung			x	x	x			○ wg. /A1/
Testfallgenerator		x		x	x	x	x	◑ /A4, A5/
Testdatengenerator		x		x	x	x		◑ /A4, A5/
Statische Analyse			x	x	x			○ wg. /A1/
Debugger			x	x	x			○ wg. /A2/
Datengetriebene Werkzeuge		x	x	x	x	x	x	● /A5, A8/
Schlüsselwortgetriebene Werkzeuge		x	x	x	x	x	x	● /A4, A5, A7/
Spezialanwendung: Last-, Performance- und Dauertest		x	x	x	x	x	x	● /A8/

Legende
● … geeignet
◑ … teilweise geeignet
○ … nicht geeignet
x … sinnvoll einsetzbar
TMW … Testmanagementwerkzeug
TSW … Testspezifikationswerkzeug
TAW … Testdurchführungswerkzeug
MT … Modultest
IT … Integrationstest
ST … Systemtest
AT … Abnahmetest

Abbildung 24 Eignung von Werkzeugtypen für das CAST Werkzeug der ganzheitlichen Testautomatisierung basierend auf den Anforderungen

Zur Automatisierung des Testprozesses mittels eines digitalen Werkzeugs /A1/ können Testskripte sinnvoll eingesetzt werden. Zusammen mit der Anforderung /A3/ nach Interaktion zwischen CAST Werkzeug und SuT sind auf Testtreibern aufbauende Testskripte die einzig passende Lösung. Testtreiber sind auf die Programmierschnittstellen von SuT angepasste Softwaremodule, um eine Interaktionsmöglichkeit zwischen CAST Werkzeug und SuT zu schaffen. Der Umfang der

am Testtreiber zur Verfügung stehenden Funktionen hängt maßgeblich von der spezifizierten Programmierschnittstelle des SuT ab. Programmierte Testskripte, welche ein oder mehrere Testtreiber einbinden sind je nach Programmierweise wiederverwendbar, für das CAST Werkzeug wird dies gefordert /A7/.

Der Komparator ermöglicht den schnellen und reproduzierbaren Vergleich großer Datenmengen nach der Testdurchführung oder von geringeren Datenmengen bereits während der Testdurchführung. Dies wurde in Anforderung /A9/ gefordert. Im Testmanagement kann der Komparator verschiedene Testdurchführungen, z. B. auf Fehleranzahl oder die Versionsnummer, vergleichen.

Testwerkzeugtypen, welche für Spezialanwendungen ausgelegt sind und z. B. das Verarbeiten großer Datenmengen oder das Generieren von randomisierten Testdaten erlauben, werden für das CAST Werkzeug benötigt, um Dauertests zu ermöglichen, Anforderung /A8/.

Die Anforderungen /A4/ Herstellerunabhängigkeit und /A5/ grafische Testfallmodellierung ergeben eine teilweise Eignung mit den Testwerkzeugtypen Testfall- und Testdatengenerator. Die Programmierung von Testskripten unter Einbindung von Testtreibern erscheint als Tätigkeit, welche Programmiererfahrung und softwaretechnisches Verständnis bzgl. verschiedener Technologien benötigt, nicht geeignet für die Zielgruppe der Produktionstechnik. Demgegenüber steht der Anspruch, den Maschinenbauer, respektive SPS-Programmierer, Testfälle erstellen zu lassen, um das Erfahrungswissen und Know-how der beiden Gruppen bzgl. der Fertigungssysteme zu nutzen. Mittels Techniken der Testfall- und Testdatengenerierung kann die Überführung von grafischer und herstellerunabhängiger Beschreibung der Testfälle in ausführbare Testskripte, automatisiert im Sinne einer Generierung, erfolgen.

Das Vorgehen für die Entwicklung des CAST Werkzeugs ist die Realisierung und Verknüpfung mehrerer Typen von Testwerkzeugen: Testskripte/-treiber, Komparator, daten- und schlüsselwortgetrieben, Testfall- und Testdatengenerator, Spezialanwendung, z. B. Dauertest. Die Kombination aus verschiedenen Werkzeugtypen wird als *Werkzeugsuite* bezeichnet, die auf Wiederverwendung ausgelegte Werkzeuge mit Bibliotheken als *Testframework* (Spillner et al. 2012, S. 209). Das

CAST Werkzeug wird im weiteren Verlauf als Werkzeugsuite entwickelt, welche den Gedanken eines Testframeworks umsetzt.

Zusammenfassend unterstützen die Festlegungen aus dem Grobentwurf für das /A1/ CAST Werkzeug auch die Anforderungen /A3/ Testtreiber, /A4/ Herstellerunabhängigkeit, /A5/ Grafische Modellierung, /A7/ Wiederverwendung, /A8/ Dauertest und /A9/ Komparator.

5.4 Betrachtung von Diversität von Steuerungsschnittstellen

Abbildung 25 zeigt das Ergebnis der Recherche von Schnittstellentechnologien gängiger Steuerungssysteme (Beckhoff Automation GmbH & Co. KG; Bosch Rexroth AG 2011; Dr. Johannes Heidenhain GmbH 2014; Siemens AG 2014; Fanuc Corporation 2016). Die Ergebnisse können wie folgt zusammengefasst werden: Über verschiedene Steuerungshersteller hinweg zeigen sich diverse, proprietäre Schnittstellen zur Kommunikation mit dem Steuerungssystem. Eine Gemeinsamkeit der Schnittstellen besteht jedoch darin, dass sie auf die weitverbreitete Kommunikation über Ethernet mit TCP (Transfer Communication Protocol) und IP (Internet Protocol) setzen. Die Schnittstellen der Siemens und Bosch Rexroth Steuerungen bieten beide den Standard OPC UA (Open Platform Communications Unified Architecture) zur Kommunikation. Durch unterschiedliche Adressräume und Methoden unterscheiden sich die beiden Steuerungssysteme allerdings im Detail. Als Kommunikationsplattform für das CAST Werkzeug wird daher ein PC mit einer ethernetfähigen Netzwerkkarte festgelegt.

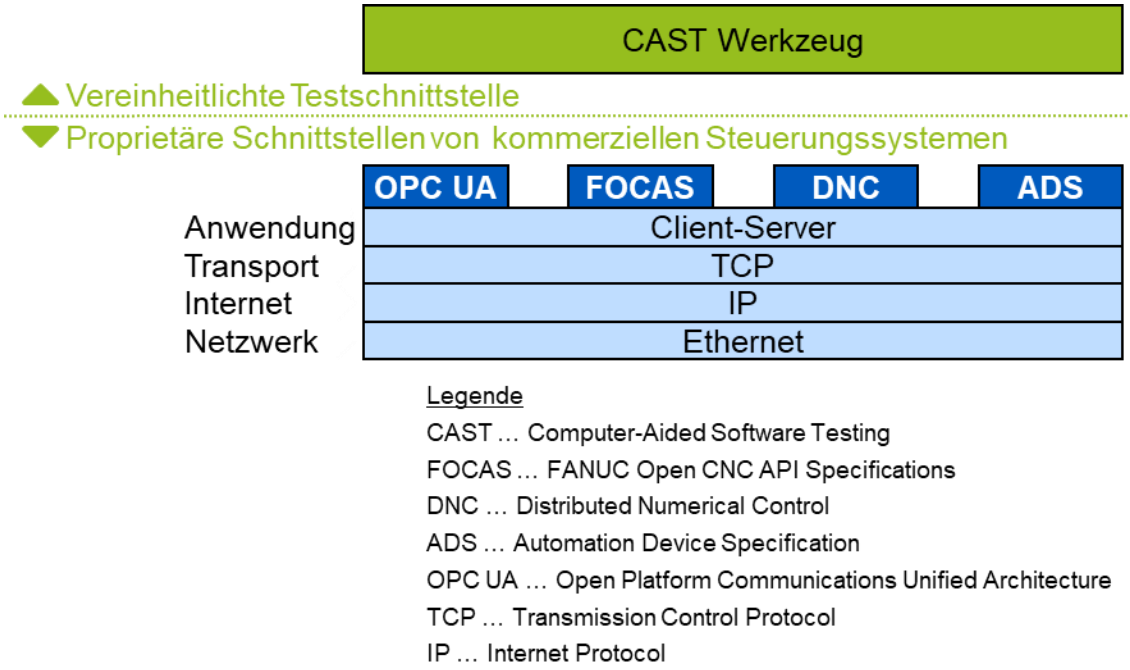


Abbildung 25 Diversität und fehlende Middleware zwischen den Schnittstellen von gängigen Steuerungssystemen eines Fertigungssystems und dem CAST Werkzeug

Mit OPC UA zeichnet sich ein standardisierter Kommunikationskanal zwischen verschiedenen Steuerungssystemherstellern ab. Zudem existieren sogenannte *Companion Specifications,* welche die Informationsdarstellung und Weitergabe mittels OPC UA über verschiedene Hersteller einer gleichen Produktgruppe (z. B. Robotik, Wägetechnik, Bilderkennung, Tabakindustrie (OPC Foundation 2022)) hinweg standardisieren. Mit diesem Schritt wäre nicht nur der Netzwerk- und Transportweg definiert und vereinheitlicht, sondern auch der Umgang mit den Informationen. Da diese Vision noch nicht umgesetzt ist, muss bis jetzt der diversen Landschaft an proprietären Schnittstellen Rechnung getragen werden. Über die aktuell betrachteten Hersteller von Steuerungssystemen hinaus muss ein einfaches Einbinden weiterer Hersteller bei zukünftigen Erweiterungen gegeben sein. Diese Erweiterbarkeit ist beim Entwurf des CAST Werkzeugs zu berücksichtigen, siehe Kapitel 0.

5.5 Entwurf der Architektur eines Werkzeugs für die ganzheitliche Testautomatisierung

Die nachfolgende Tabelle in Abbildung 26 zeigt eine Auswahl an gesammelten Testfällen von Herstellern von Fertigungssystemen, siehe auch (Kübler et al. 2016). Die gesammelten Testfälle lagen in den meisten Fällen nicht systematisch erfasst vor und wurden erst im Rahmen dieser Erfassung niedergeschrieben. Für die Anwendbarkeit in einem Testwerkzeug und damit zur automatisierten Testdurchführung und -auswertung ist dies notwendig. Ziel der Sammlung der Testfälle ist es, herauszuarbeiten, welche Schnittstellen einer RT-HiL als Testumgebung beim Testen genutzt werden. Für die Erfassung der Testfälle wurden die folgenden Daten erhoben:

- **Beschreibung** – Kurzer erläuternder Text zum Testfall.
- **Vorbedingung** – Anweisungshafte Beschreibung, welche Vorkehrungen für die Ausführung des Testfalls zu treffen sind.
- **Abbruchbedingung** – Beschreibung einer Bedingung, bei welcher die Testdurchführung frühzeitig abgebrochen werden kann.
- **Bedienschritte** – Anweisungshafte Beschreibung, aus welchen Einzelschritten der eigentliche Testfall besteht.
- **Nachbedingung** – Anweisungshafte Beschreibung, welche nachbereitenden Schritte nach der Ausführung eines Testfalls durchzuführen sind.
- **Erwartetes Resultat** – Beschreibung einer Bedingung, bei welcher die Testdurchführung als erfolgreich beendet wird.
- **Speicherstelle** – An welchen Stellen des Steuerungssystems stehen die Daten, welche bei der Testdurchführung benötigt werden.
- **Auswertezeitversatz (s)** – Gibt an, welche unter „Erwartetes Resultat“ angegebene Bedingung sofort zu prüfen ist oder mit einem zeitlichen Versatz.

Die im Testprozess (Kapitel 2.4.2) definierte Testdurchführung wird weiter chronologisch unterteilt in *Testvorbereitung*, *Testdurchführung* und *Testabschluss*. Die Testvorbereitung dient dazu, den Zustand des SuT auf die Testdurchführung

vorzubereiten, bzw. den Status aus den Vorbedingungen abzuprüfen. Während der Testdurchführung werden die Bedienschritte ausgeführt und darauf geachtet, ob eine Abbruchbedingung auftritt. Im Testabschluss wird das SuT in seinen Zustand vor der Testdurchführung gebracht, indem die Schritte Nachbedingungen ausgeführt und geprüft werden. Zudem führt der Testabschluss einen Vergleich mit den erwarteten Resultaten durch und erstellt den Testbericht.

Die Tabelle in Abbildung 26 zeigt u. a. eine Spalte für die notwendigen Bedienschritte zur Ausführung des Testfalls. Bei der Durchsicht dieser Spalte ist zu erkennen, dass zur Ausführung fast immer eine Interaktion mit dem HMI stattfindet, sowie Werte aus der NC, der SPS und der Busverbindung benötigt werden. Signale der Busverbindung können in der Simulation ausgelesen werden und auch beeinflusst werden, siehe z. B. Temperatur in Testfall 7. Dies bedeutet für die Werkzeugunterstützung bei der ganzheitlichen Testautomatisierung, dass das zu entwerfende Werkzeug in der Lage sein muss, mit allen Komponenten eines Steuerungssystems sowie der laufenden Simulation zu kommunizieren. Über die HMI sind die Interaktionsschritte des Bedieners abzubilden. Für die NC und SPS Steuerungen sind deren Fehlermeldungen und Werte, wie z. B. aus dem NC-Kanal, wichtig.

Nr.	Beschreibung	Vorbedingung	Bedienschritte	Erwartetes Resultat
1	NC-Programm und WZW überprüfen	1. Testprogramm laden 2. Testprogramm anwählen	1. NC-Reset auf dem HMI drücken 2. Auto-Modus-Taste auf HMI drücken 3. NC-Start-Knopf am HMI drücken 4. Prüfen, richtiger Werkzeugbezeichner aktiv 5. Prüfen, richtige Schneide aktiv 6. Prüfen, Rechenparameter richtig gesetzt	a) Werkzeugbezeichner ist richtig b) Schneide ist richtig c) R-Parameter hat richtigen Wert
2	NC-Stop ohne Freischneiden bei G0	1. Testprogramm laden 2. Testprogramm anwählen 3. NC-Start-Knopf am HMI drücken 4. Warten (20s) bis Spindel Drehzahl erreicht hat	1. NC-Stop am HMI auslösen 2. Kurzes warten (1s) 3. Überprüfen ob Spindel gestoppt hat	Spindel stoppt sofort
3	Maschine in Grundstellung nach Reset	1. Auf Steuerungshochlauf warten 2. Warten auf Meldung "Vorschub nicht eingeschaltet"	1. Stop A/ Stop D Meldung löschen 2. Fehler kreuzweise Datenvergleich löschen 3. Fehler löschen 4. HMI in das Maschinengrundbild schalten 5. Automatik aktivieren 6. Reset und Fehler löschen	a) Maschine befindet sich in Grundstellung b) Alle Fehler gelöscht
4	Abbruchsituation im Werkzeugwechsel-zyklus	1. Werkzeugdaten synchronisieren 2. Testprogramm laden 3. Testprogramm anwählen 4. NC-Start-Knopf am HMI drücken	Bei Fehlermeldung: 1. Reset 2. Vorschub Start 3. Grundstellung	Passende Fehlermeldung zum blockierten Eingang
5	Werkezeug einlagern in Hintergrund-magazin	1. Werkzeugdaten synchronisieren 2. Grundzustand herstellen	Wenn Beladetüre entriegelt: 1. Türe öffnen 2. WZ einlagern 3. Türe schließen	Werkzeug korrekt einlagern
6	Grundstellungs-fahrt	1. Testprogramm laden 2. Testprogramm anwählen 3. NC-Start-Knopf am HMI drücken	1. x-Sekunden warten 2. Reset drücken 3. Grundstellung fahren	Maschine fährt in Grundstellung
7	Temperatur-kompensation	1. Testprogramm laden 2. Testprogramm anwählen 3. NC-Start-Knopf am HMI drücken 4. bis bestimmten Programmpunkt warten	1. Spindeltemperatur erhöhen	Andere Position in kompensierter Achse

Abbildung 26 Beispielhafte Testfälle von Werkzeugmaschinenherstellern, eigene Erhebung

Aus den Erkenntnissen der Kapitel 5.2 bis 5.5 kann ein Grobentwurf der Architektur für das CAST Werkzeug abgeleitet werden. Die Abbildung 27 stellt die Architektur dar und zeigt dabei interne Aspekte des CAST Werkzeugs, siehe linke Seite, sowie die externe Kopplung zum RT-HiL System, siehe rechte Seite. Der Zusammenschluss der digitalen Werkzeuge CAST und RT-HiL wird als Virtuelle Steuerungstestbench (ViST) bezeichnet. Die Verknüpfung eines externen Systems mit der Simulationsumgebung, HiL oder SiL findet sich auch in den Arbeiten von (Kohring 1993; Bender 1999; Kreusch 2002; Herfs 2010), meist um Signale und Daten, wie NC-Programme, während der Testdurchführung aufzuzeichnen, zu manipulieren und zu übertragen. Die Methodik der ganzheitlichen Testautomatisierung überträgt zum einen diese bekannte Architektur auf moderne RT-HiL Systeme und berücksichtigt gleichzeitig die vollautomatische Durchführung und Auswertung der Testfälle.

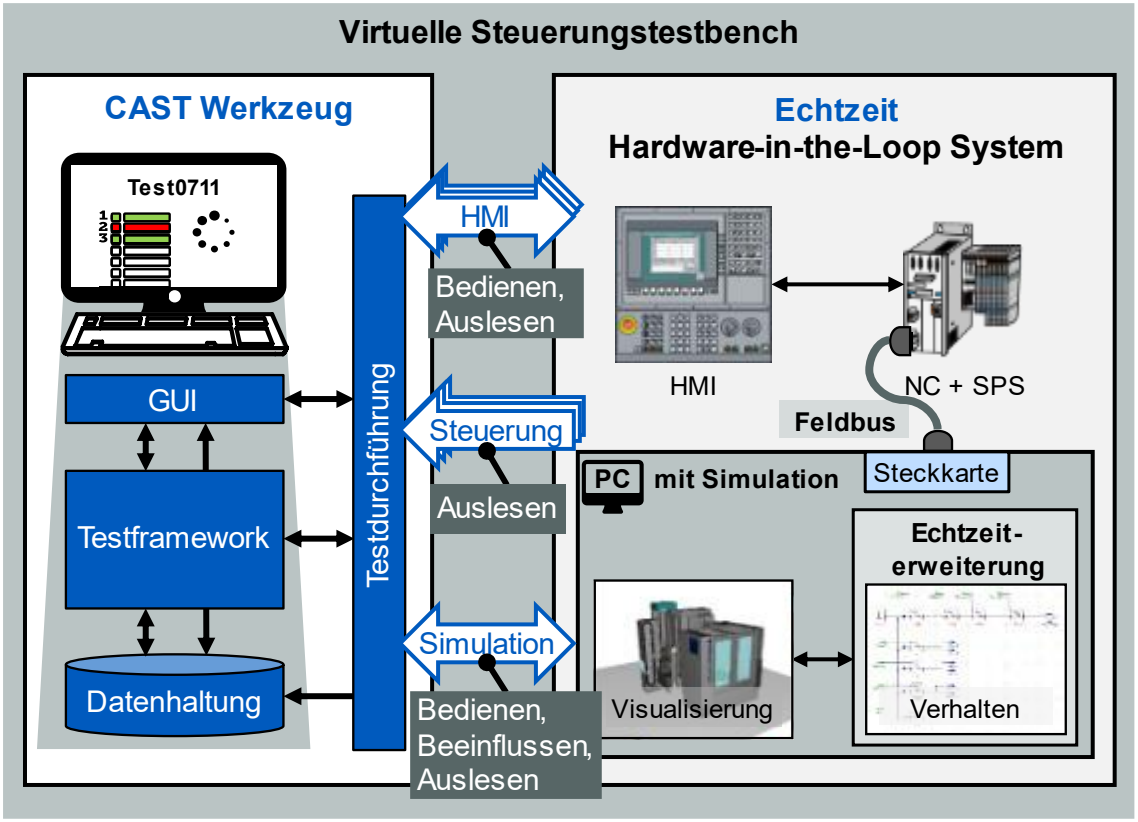


Abbildung 27 Architektur der virtuellen Steuerungstestbench aus CAST Werkzeug und RT-HiL System

Der Grobentwurf sieht vor, dass drei Schnittstellen, siehe weiße Pfeile in der Abbildung, realisiert werden. Die Schnittstellen zum HMI und zur Simulation müssen

bidirektional ausgelegt werden, die Schnittstelle zur Steuerung nur unidirektional. Der Ablauf in der Steuerung soll zur Laufzeit des Tests nicht beeinflusst werden, da sonst die Gültigkeit der Testergebnisse nicht mehr mit dem späteren realen Betrieb übereinstimmen könnten. Beeinflussungen der Steuerung, z. B. bzgl. des aktuell auszuführenden Programms, werden wie im späteren realen Betrieb über die HMI vorgenommen. Die angedeuteten Mehrfachpfeile bei HMI und Steuerung sind der Anforderung nach Herstellerunabhängigkeit geschuldet. Für die Ankopplung der Simulation wurden, nach Anforderung /A2/, nur Simulationswerkzeuge zur Echtzeitsimulation von Steuerungssystemen untersucht. Zum Zeitpunkt der Entstehung dieser Arbeit existierte nur ein kommerzielles Simulationswerkzeug für die RT-HiL, siehe Kapitel 2.2.1. Ein eigenes Simulationswerkzeug zu konzipieren liegt außerhalb des Fokus dieser Arbeit.

Das CAST Werkzeug wird entworfen als eine Testsuite, welche über eine „grafische Benutzeroberfläche (GUI)“ zu Zwecken des Testmanagements und zum Testentwurf verfügt. Weiterhin besitzt das CAST Werkzeug für die Ausführung der modellierten Testfälle ein Testframework. Das „Testframework“ wandelt dabei grafisch modellierte Testfälle in ausführbare Skripte um und übersetzt zwischen einer generischen Beschreibung der Aktionen in einem Testfall und den verschiedenen Steuerungssystemen. Die „Datenhaltung“ dient dem Testmanagement, z. B. durch Betrachten aller abgeschlossenen Testdurchführungen. Beim Erstellen von Testfällen hält die Datenhaltung eine Bibliothek an wiederverwendbaren Testfällen vor. Während der Testdurchführung werden Informationen zum laufenden Testfall synchron an die GUI übertragen und Protokoll geführt. Sowohl das Testprotokoll als auch der Testbericht werden in der Datenhaltung archiviert.

5.6 Zusammenfassung

In diesem Kapitel wurde das Grobkonzept der Methodik zu ganzheitlicher Testautomatisierung für automatisierte Fertigungssysteme aufgestellt. Dazu wurden zunächst die Einordnung und Integration in das aktuelle Vorgehen bei der Entwicklung betrachtet. Daraus ergab sich, dass die Erstellung von Testfällen bereits parallel zur Konstruktion des Fertigungssystems und zur Erstellung der virtuellen

Testumgebung durchgeführt wird. Damit ergibt sich mittels ganzheitlicher Testautomatisierung ein weiteres Potenzial zum Frontloading und Simultaneous Engineering in der Entwicklungsphase. Vor der Testdurchführung muss entweder eine virtuelle Testumgebung als RT-HiL System oder das reale Fertigungssystem zur Verfügung stehen. Für den Regressionstest am realen Fertigungssystem ergibt sich Einsparungspotenzial, da aus der Entwicklungsphase bestehende Testfälle mit der Testautomatisierung erneut aufwandsarm durchgeführt werden können.

Der neue Testprozess mittels ganzheitlicher Testautomatisierung erweitert den klassischen Testprozess um Techniken, welche von einer Automatisierung profitieren, bspw. Performancetests, Robustheitstests und weitere datenintensive Testarten. Auch Testverfahren zur Spezifikation von Testfällen können mittels ganzheitlicher Testautomatisierung sinnvoller umgesetzt werden, dazu zählen Grenzwertanalysen und Error Guessing. Aus Testtechniken und Testwerkzeugtypen der Softwaretechnik wurde eine geeignete Auswahl für ein Testwerkzeug getroffen. Die Auswahl ergab ein CAST Werkzeug als Werkzeugsuite, darin werden mehrere Testwerkzeugtypen vereint: Testskripte/-treiber, Komparator, Testfall- und Testdatengenerator, Spezialanwendung: z. B. Dauertest.

Zur Ermittlung der Schnittstellen des CAST Werkzeugs wurden zum einen vorhandene Steuerungssysteme und Simulationswerkzeuge untersucht, zum anderen wurden Testfälle bei Herstellern von Fertigungssystemen erfasst. Die Schnittstellen der Steuerungssysteme ergaben ein diversifiziertes Bild mit der Gemeinsamkeit des Transportweges der Daten mittels TCP/IP über Ethernet. Die erfassten Testfälle der Hersteller machen separate Schnittstellen zur HMI, SPS, NC und Simulation des RT-HiL Systems notwendig.

Abschließend wurde basierend auf den obigen Informationen ein Grobentwurf der Architektur des CAST Werkzeugs, inklusive internem Aufbau und externem Anschluss an das RT-HiL System, vorgestellt und erläutert.

Die folgenden Anforderungen aus Kapitel 3 wurden im Grobentwurf erfüllt:

/A1/ Der Testprozess wird durch Automatisierung mittels eines digitalen Werkzeugs unterstützt (Testautomatisierung mittels CAST Werkzeug)

/A2/ Neben einer realen Testumgebung muss eine virtuelle Testumgebung mittels RT-HiL unterstützt werden (System-under-Test real/virtuell)

/A3/ Das CAST Werkzeug muss in der Lage sein, mit den Komponenten des SuT zum Zweck der Testautomatisierung interagieren können (Testtreiber)

6 Methodik zur ganzheitlichen Testautomatisierung

Das Kapitel 6 stellt den zentralen Teil dieser Arbeit dar, denn hierin wird die Methodik zur ganzheitlichen Testautomatisierung von automatisierten Fertigungssystemen entworfen. Abbildung 28 zeigt eine Einteilung der einzelnen Methoden nach der Digitalen Fabrik: digitale Werkzeuge, Methoden und Modelle.

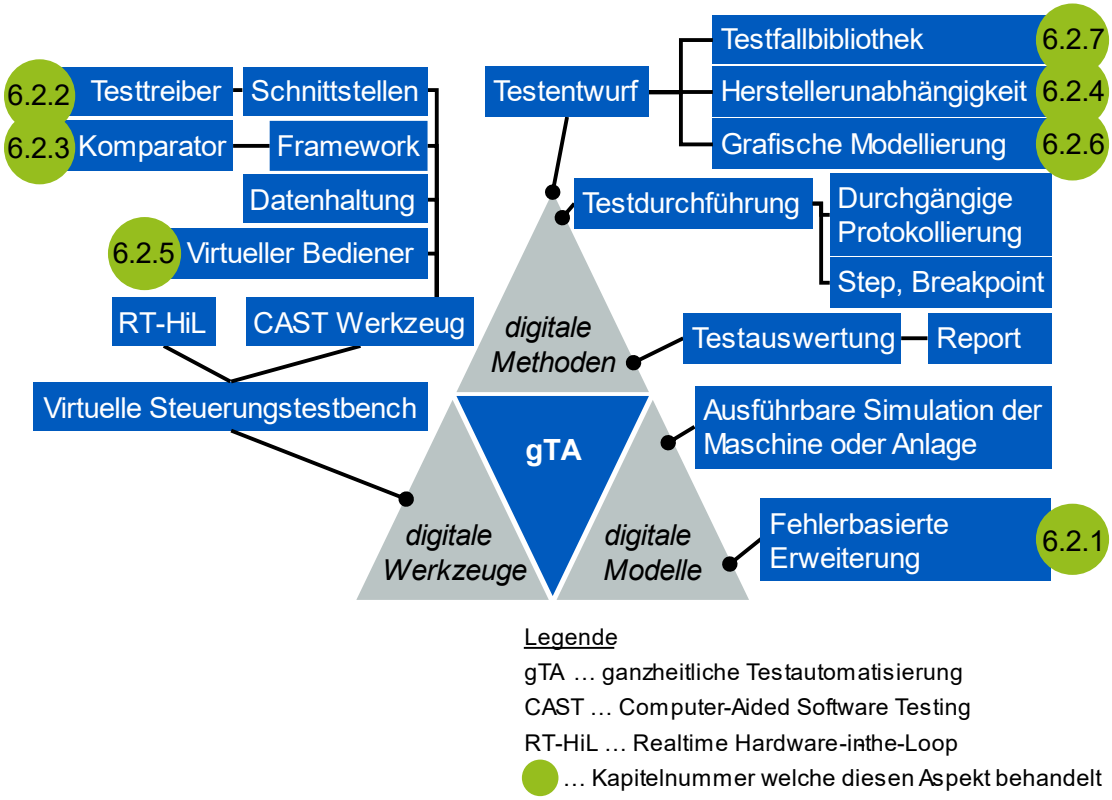


Abbildung 28 Umfang der neuen Testmethodik, eingeteilt nach digitalen Methoden, Modellen und Werkzeugen

Zunächst wird in diesem Kapitel der Grobentwurf der Architektur aus Kapitel 5 gegen eine bestehende allgemeingültige Testsystemarchitektur aus der Literatur verifiziert. Nach der Verifikation wird das Vorgehen beim Feinentwurf nach der Bottom-up-Methode erläutert. Darauf folgt der detaillierte Entwurf für die notwendigen digitalen Modellen, Werkzeuge und Methoden. Abbildung 28 zeigt anhand grüner Kreise, welche Aspekte der Lösung in welchem Unterkapitel besprochen werden. Die Integration des Gesamtsystems zu einer ausführbaren Lösung und dessen Validierung findet anschließend in Kapitel 7 statt.

6.1 Verifikation des Grobkonzepts und Vorgehen bei der Erarbeitung der Methodik

(Korotkiy 2010) beschreibt in seiner Dissertation auf den Seiten 61 – 66 eine universelle Testsystemarchitektur für den Systemtest mechatronischer Produkte. Die Arbeit von Korotkiy erscheint geeignet, um den Grobentwurf der Architektur des CAST Werkzeugs zu verifizieren, da Fertigungssysteme als mechatronische Produkte betrachtet werden können. Dazu werden die bereits identifizierten Komponenten des CAST Werkzeugs in die universelle Testsystemarchitektur eingezeichnet und geprüft, ob alle Aspekte der universellen Testsystemarchitektur erfüllt sind. Abbildung 29 zeigt das positive Ergebnis der Prüfung.

Anschließend wird ein Bottom-up Vorgehen, umgekehrt zur Position des Benutzers, eingesetzt, um den Feinentwurf der Architektur umzusetzen. Auf der linken Seite von Abbildung 29 ist dieser Zusammenhang visualisiert. Die Steuerungsprogramme auf der untersten Ebene der Architektur sind die Testobjekte für das CAST Werkzeug. Die Programme bleiben daher unverändert und müssen nicht weiter betrachtet werden. Die eingesetzten RT-HiL Systeme zur echtzeitfähigen Simulation von Fertigungssystemen bestimmen mit ihren Schnittstellen maßgeblich, wie die Testautomatisierung realisiert werden muss. Zunächst werden die Schnittstellen analysiert und mittels Testtreibern als ausführbare Einheiten in das Testframework integriert. Das Regelwerk zur herstellerunabhängigen Vereinheitlichung der Testtreiber wird in das Testframework integriert. In das Testframework muss zudem der Komparator integriert werden, welcher die Werte und Signale der Testtreiber auswertet und gegen die vom Benutzer gesetzten Sollwerte vergleicht. Aus dem Testmanagement erhält das Testframework in Prosa definierte bzw. grafisch beschriebene Testfälle, welche in ausführbare Testfälle in Form von Testskripten überführt werden müssen. Die GUI muss dem Benutzer die am ausgewählten RT-HiL System zur Verfügung stehenden Aktionen bereitstellen und die grafische Testfallspezifikation ermöglichen. Zusätzlich ermöglicht die GUI es, Aufgaben aus dem Testmanagement zu erledigen. Dafür muss eine Datenhaltung entworfen werden, welche Daten aus GUI, Testframework und Testtreibern entgegennehmen kann, diese speichert und teilweise an die GUI

bereitstellt. Wichtig bei der Entgegennahme von Daten ist die Verknüpfung mit zugehörigen Daten aus weiteren Quellen: Bspw. muss das Testprotokoll einer Testdurchführung zusammen mit den ausgeführten Testfällen, dem erstellten Testbericht und der Konfiguration des RT-HiL Systems abgelegt werden. Nur so kann im Testmanagement nachvollzogen werden, welche Tests, in welcher Version, mit welchen Ergebnissen durchgeführt wurden. Auch Regressionstests und der Vergleich mit vorangegangenen Testdurchführungen sind so möglich.

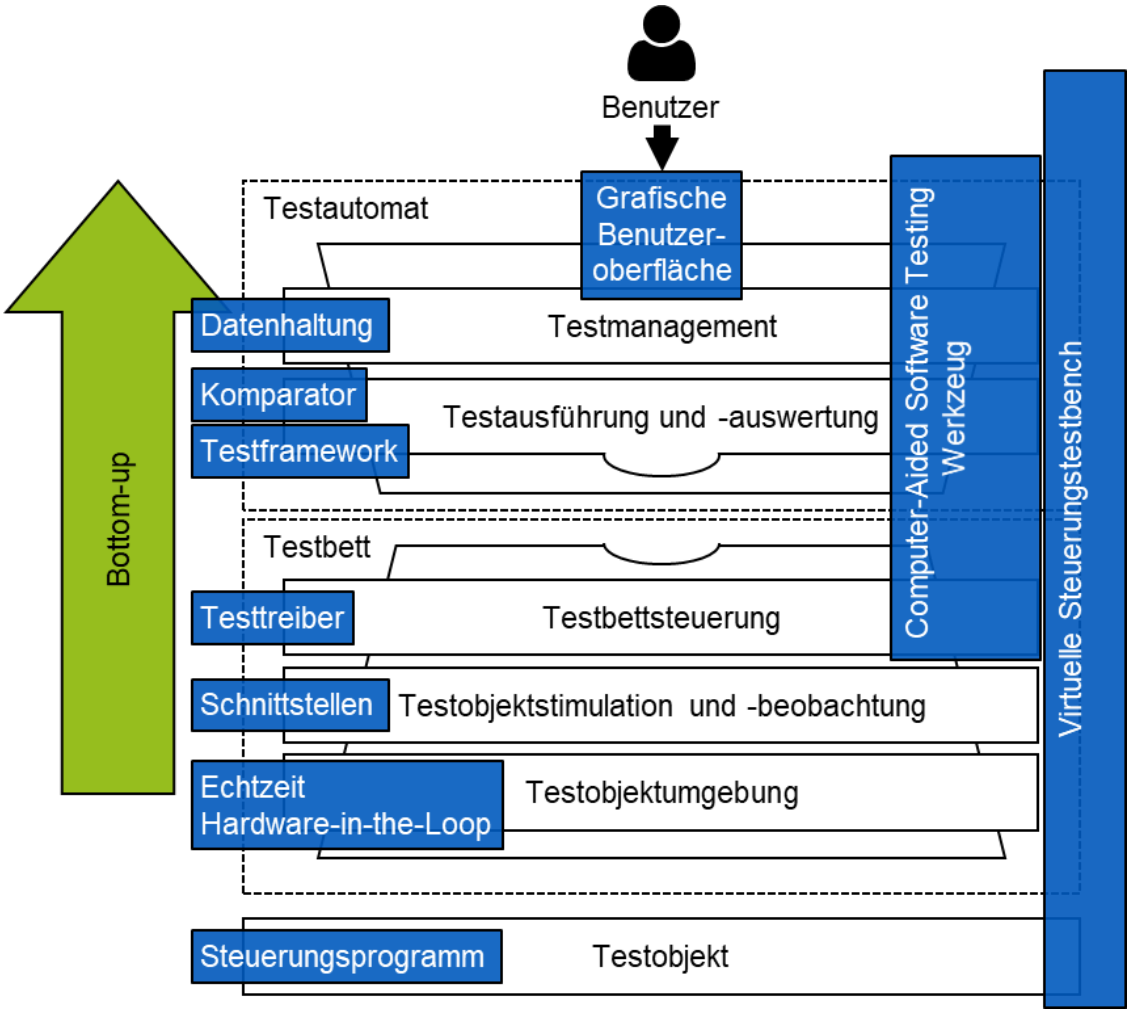


Abbildung 29 In Anlehnung an die universelle Testsystemarchitektur für mechatronische Produkte nach (Korotkiy 2010) wurde das Konzept für das Werkzeug für die ganzheitliche Testautomatisierung als blaue Flächen eingezeichnet

Bei der Gruppierung von ViST und CAST in Abbildung 29 wird ein Unterschied zur universellen Testsystemarchitektur sichtbar. Während (Korotkiy 2010) zwischen Testautomat und Testbett unterscheidet, ist der Testtreiber noch Teil des

CAST Werkzeugs. Diese Tatsache ist dem geschuldet, dass zum einen am Steuerungssystem im RT-HiL System keine Änderungen vorgenommen werden sollen, um einen gleichen Ablauf in der virtuellen Testumgebung wie später im Produktivbetrieb zu gewährleisten. Zum anderen ermöglicht die Integration der Testtreiber in das CAST Werkzeug (in den Testautomat nach Korotkiy) eine saubere software- und kommunikationstechnische Trennung zwischen RT-HiL und CAST Werkzeug, sodass ein einfacher Austausch der Konfiguration des RT-HiL Systems gewährleistet ist. Somit ist ein CAST Werkzeug, mit den entsprechend umgesetzten Schnittstellen, in der Lage, an verschiedene RT-HiL Systeme gekoppelt zu werden.

6.2 Methodik für die ganzheitliche Testautomatisierung

In diesem Kapitel werden gemäß den Anforderungen die einzelnen Lösungskomponenten der ganzheitlichen Testautomatisierung erarbeitet.

6.2.1 Fehlerbasierte Erweiterung des Simulationsmodells

Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A6/ Berücksichtigung von Fehler- und Diagnosefällen bei der ganzheitlichen Testautomatisierung (Schlechtfalltest)

Wie in Kapitel 3.3 gezeigt, sind ca. zwei Drittel der Module in den Steuerungsprogrammen vorhanden, um Fehler- und Diagnosefälle im Fertigungssystem zu erkennen und durch fehlersichere Programmierung Schaden an Maschine, Werkstück oder Personal zu verhindern. Ziel der fehlerbasierten Modellerweiterung für die ganzheitliche Testautomatisierung ist es, mittels eines erweiterten Simulationsmodells Schlechtfalltests in Form von Störsituationen und Fehlerfällen durchführen zu können. Die fehlerbasierte Modellerweiterung leistet damit einen Beitrag zur Erhöhung der Testabdeckung, indem die erweiterten Simulationsmodellbereiche über den laufenden Testfall ausgelöst werden können.

Als **Fehlerfall** werden Fehler bezeichnet, welche im Betrieb, also nach der IBN, innerhalb des Fertigungssystems auftreten.

Die **Störsituation** bezeichnet einen Fehler, welcher ebenfalls im Betrieb auftritt, aber von außerhalb des Systems einwirkt. Dazu zählen besonders eingebrachte Materialen, wie Werkstücke, und die Bedienhandlungen des Maschinenführers.

Testfälle, welche eine Störsituation oder einen Fehlerfall testen, werden hier als **Schlechtfalltest** bezeichnet.

Um eine Modellerweiterung zu ermöglichen, werden zunächst typische Fehlerfälle von Komponenten eines Fertigungssystems recherchiert. Aus den Fehlerfällen werden dann Signalverhalten extrahiert, welche anschließend im Simulationsmodell modelliert werden. Zuletzt werden für das Auslösen der Fehlerfälle relevante Szenarien definiert. Die Auslöseszenarien werden in das Simulationsmodell integriert und nicht im Testfall definiert. Damit wird gewährleistet, dass das Auslösen der Schlechtfalltests im Simulationstakt geschehen kann.

Fehler des Steuerungssystems selbst welche in Hardware (z. B. Produktionsfehler von Bauteilen, EMV-Störungen) oder Software (z. B. Programmierfehler im Betriebssystem) auftreten können werden im Rahmen dieser Arbeit nicht betrachtet. Röck (Röck 2007, S. 22-24) liefert in seiner Arbeit eine brauchbare Übersicht, welche Laufzeitfehler in Hardware und Software von Steuerungssystemen auftreten können. In den Arbeiten von (Wünsch 2008; Hackenberg et al. 2015) sind Versagen, welche an Komponenten eines Fertigungssystems auftreten können, benannt. Abbildung 30 zeigt deren Auflistung und klassifiziert das jeweilige Versagen nach Fehlertypen an einem Fertigungssystem. Die Fehlertypen haben, in Klammern ergänzt, die Einteilung nach (Laible 2005), von Seite 24.

Fehlertypen an Fertigungssystemen	Versagen nach (Wünsch 2008)	Versagen nach (Hackenberg et al. 2015)
Elektrikfehler (vor/nach IBN) (systematisch/zufällig)	• Kabelbruch • Sensordefekt • Aktordefekt	• Kabelbruch
Mechanikfehler (vor/nach IBN) (systematisch/zufällig)	• Schwergängige Mechanik • Sensordefekt • Aktordefekt • Sensor fehlerhaft (Span auf Sensor)	• Schwingungen • Achsbruch • Defektes Werkzeug • Nässe
Steuerungsfehler (vor IBN) (systematisch)	• -	• Falsche Implementierung
Werkstückfehler (nach IBN) (außerhalb)	• -	• Falsches Rohteil • Falsches Material
Bedienfehler (nach IBN) (außerhalb)	• -	• Falsche Eingabe • Geöffnete Schutztür
Automatisierungsfehler (nach IBN) (außerhalb/systematisch)	• Mechanisches Problem	• Falscher Startbefehl • Lageabweichung
Versorgungsfehler (nach IBN) (außerhalb/systematisch)	• Druckabfall • Medium fehlt	• Energieausfall

Abbildung 30 Auflistung und Gegenüberstellung von Versagen innerhalb eines Fertigungssystems aus Literaturquellen

Zu erkennen ist, dass die Fehler entweder auf eine mechatronische Komponente des Fertigungssystems zurückzuführen sind oder über eine der Schnittstellen zur Umgebung des Fertigungssystems (Bediener, Werkstück) eingebracht werden. Der Einsatzzweck von ganzheitlicher Testautomatisierung zum Finden von Steuerungsfehlern bedingt, dass alle Fehler betrachtet werden: Elektrikfehler, Mechanikfehler, Werkstückfehler, Bedienfehler, Automatisierungsfehler, Versorgungsfehler. Alle Fehler bis auf die Bedienfehler werden in der Simulation abgebildet. Dazu werden zunächst Signalverläufe identifiziert, welche mit den Fehlern assoziiert werden können.

Signaltypen zur Modellerweiterung

Das Steuerungssystem interagiert im Wesentlichen mit der Sensorik und Aktorik des Fertigungssystems. An einem RT-HiL System übernimmt die Simulation die Generierung der Signale von Sensoren und Aktoren. Somit muss das Versagen von Komponenten mittels der Signale der Sensoren und Aktoren an die Steuerung übermittelt werden. Ausgehend von den gesammelten Versagensarten wurde eine umfangreiche und detaillierte Klassifizierung von Versagen in den mechatronischen Komponenten eines Fertigungssystems vorgenommen. Dazu wurden sowohl Quellen aus der Literatur (Kollmann et al. 1954; Weck et al. 2006; Isermann 2008; Weidauer 2013; Wittel et al. 2013; Norm DIN EN ISO 13849-2 2013; Früh et al. 2015; Jan et al. 2017), als auch Montage- und Bedienhandbücher genutzt (Friedrich Braun GmbH; Janisch; Nau 2004; JUMO GmbH & Co. KG 2009; Getriebebau NORD GmbH & Co. KG 2016; Schneider Electric Industries SAS 2016). Bei der Recherche zu den Versagensarten konnten acht eigenständige und zwei erweiterte Signaltypen identifiziert werden, welche die Veränderung in der Simulation nach einem komponentenbasierten Versagen darstellen. Die insgesamt zehn Signaltypen sind in Abbildung 31 aufgeführt und erläutert.

#	Signaltyp	Erläuterung
1	Drift	Eine linear ansteigende Abweichung vom Sollwert
2	Nullpunktfehler	Konstante Abweichung vom Sollwert
2.1	Teilweiser Einbruch	Ist im Vergleich zum Nullpunktfehler erst nach einer Totzeit aktiv
3	Vorzeichenfehler	Umgekehrtes Vorzeichen gegenüber dem Sollwert
4	Freezing	Istwert bleibt ab einem Zeitpunkt unverändert stehen
4.1	Stuck	Äquivalent zu Freezing für ein binäres Signal
5	Hard-over	Plötzlicher Sprung über den Maximalwert
6	Kompletter Einbruch	Plötzliches Absacken des Istwerts auf den Minimalwert bzw. auf „0“
7	Stochastische Überlagerung	Überlagerung des Istwerts mit z. B. Rauschen
8	Deterministische Überlagerung	Überlagerung des Istwerts mit z. B. Sinus, Rechteck, Sägezahn, gedämpfte Schwingung
9	Zeitverzögerung	Zeitliche verzögertes Nachfolgen zum Sollwert

Abbildung 31 Auflistung der identifizierten Signaltypen für Fehlermodellierung mit Erläuterung

Zusätzlich zeigen Abbildung 58 und Abbildung 59 im Anhang dieser Arbeit die Verläufe der zehn Signaltypen. Dazu sind die Sollverläufe (durchgezogen) und Istverläufe (gestrichelt) gegenübergestellt.

Abbildung 32 zeigt beispielhaft eine realisierte Simulationsmodellerweiterung um einige Fehlersignale. Links oben ist eine Schwenkbrücke zum Ein- und Auslasten von Paletten mit aufgespannten Werkstücken gezeigt. Die Paletten werden mit der Z-Achse in den Bearbeitungsraum der Werkzeugmaschine zugestellt. Die Schwenkbrücke kann angehoben und gesenkt sowie gedreht werden. Links unten findet sich ein Teil des Simulationsmodells der Schwenkbrücke: die ergänzten Signalfehlerblöcke sind durch halbtransparente Quader markiert. Auf der rechten Seite der Abbildung ist eine zusätzlich erstellte virtuelle Bedientafel zu sehen, mit der die Signalblöcke und damit einer der fehlerhaften Signalverläufe auf den Aktor zum Drehen oder zum Heben/Senken geschalten werden kann.

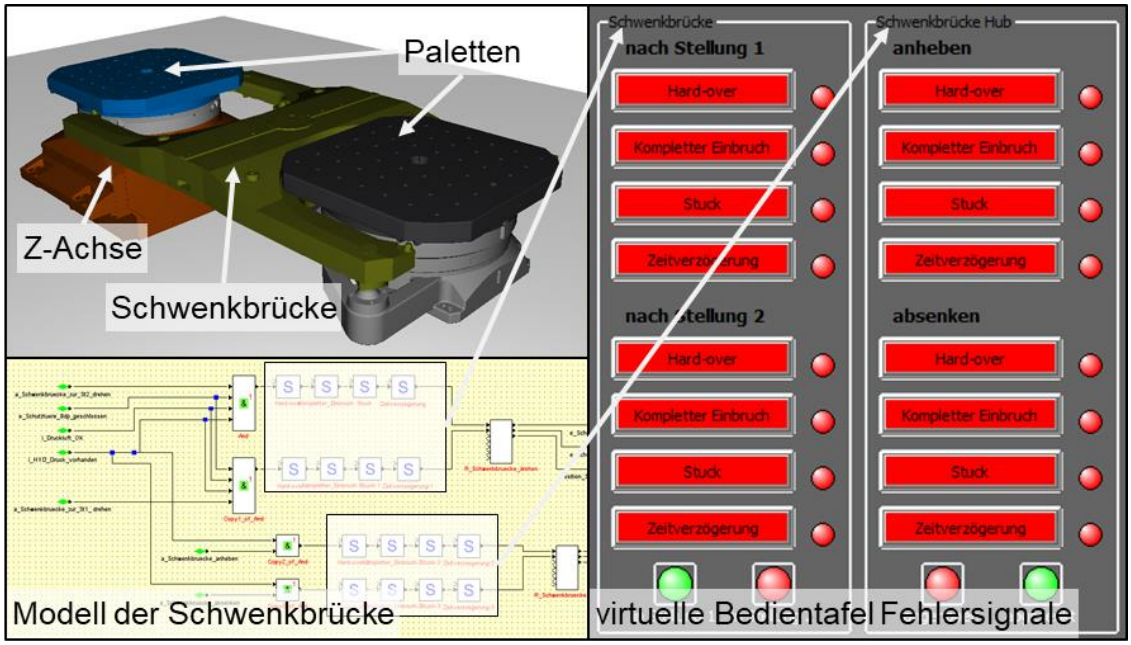


Abbildung 32 Modellerweiterung einer Schwenkbrücke (oben links) eines CNC-Bearbeitungszentrums um fehlerhafte Aktorsignale (unten links) und eine manuelle, virtuelle Bedientafel zum Auslösen (rechts)

Die erstellten Signalfehlerblöcke werden in einer Bibliothek abgespeichert und können im Folgenden in jedem Simulationsmodell eingefügt werden, das über bestimmte Sensoren und Aktoren verfügt.

6.2.2 Entwurf der Testtreiber für das RT-HiL System

Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A3/ Das CAST Werkzeug muss in der Lage sein, mit den Komponenten des SuT zum Zweck der Testautomatisierung interagieren können (Testtreiber)

In den Kapiteln 5.4 und 5.5 wurden die für das CAST Werkzeug relevanten Schnittstellen am RT-HiL System identifiziert: HMI, Steuerung und Simulation. Im CAST Werkzeug sollen verschiedene HMI, Steuerungen und Simulationen angebunden werden können. Die Schnittstellen zu den einzelnen Systemen zu realisieren und dem Benutzer beim Testentwurf zur Verfügung zu stellen, ist die Aufgabe bei der Erstellung der Testtreiber.

Zur Erstellung der Testtreiber müssen zunächst die vorhandenen Technologien der einzelnen Komponenten untersucht werden. Bei der Bereitstellung der Testtreiber soll das Know-how zur Nutzung der Schnittstellen gekapselt werden.

Wie in den Voruntersuchungen gezeigt, basiert die Umsetzung jeder Schnittstelle auf dem Client-Server Prinzip: HMI, Steuerung oder Simulation stellen einen Server bereit, der mittels einer Clientapplikation verbunden werden kann. Nach dem erfolgreichen Verbinden stehen unterschiedliche Methoden zur Anfrage von Daten, dem Schreiben von Werten und weiteren Methoden zur Verfügung. Der Funktionsumfang zur Verbindung mit dem Server wird als nicht-funktionale Instruktionen bezeichnet:

- **SetConnectionInfo**() – Konfiguration der Verbindungsdaten zur Komponente
- **Connect**() – Herstellen und Halten einer aktiven Verbindung zur Komponente
- **Disconnect**() – Schließen und Löschen der Verbindung zur Komponente
- **GetStatusOfInterface**() – Abfrage des Zustands der Verbindung zur Komponente

Sollte die nicht-funktionale Instruktion GetStatusOfInterface) nicht von der Schnittstelle proprietär angeboten werden, ist es sinnvoll, diese selbst zu erstellen. Somit kann die Robustheit während der Testdurchführung im Testframework erhöht werden. Vor einer Daten- oder Methodenanfrage an den Server kann die Instruktion als Vorbedingung im Testtreiber geschalten werden, sodass Zugriffe aufgrund einer inaktiven Verbindung nicht fehlerhaft enden.

Neben den nichtfunktionalen Instruktionen bestehen funktionale Instruktionen je Schnittstelle. Diese Instruktionen zeichnen sich dadurch aus, dass sie während eines Testablaufs genutzt werden. Während eines Testablaufs müssen Datensätze gelesen, Datensätze geschrieben und proprietäre Funktionen auf der Komponente ausgeführt werden. Verallgemeinert lassen sich somit die folgenden funktionalen Instruktionen beschreiben:

- **ReadDataset**() – Lesen eines beliebig großen Datensatzes aus der verbundenen Komponente.
- **WriteDataset**() – Schreiben eines beliebig großen Datensatzes in der verbundenen Komponente (gilt nicht für die Steuerungsschnittstelle).
- **TriggerMethod**() – Auslösen einer Aktion in der verbundenen Komponente.

Aus Benutzersicht ist das Kapseln von nicht relevanten Zwischenschritten im CAST Werkzeug sinnvoll. Dabei kann es sich um Wandlung von Datentypen oder Einheiten handeln sowie um das spezifische Know-how einer proprietären Schnittstelle. Weiterhin sind Hilfsfunktionen notwendig, z. B. zur Erstellung bzw. Abarbeitung eines beliebig großen Datensatzes. Das Datenhandling von beliebig großen Datenmengen wird nicht von jeder Schnittstelle unterstützt und muss anderweitig im CAST Werkzeug bereitgestellt werden, siehe Abschnitt 6.2.6.

Die insgesamt sieben identifizierten Instruktionen werden für das Testframework als zwingend notwendig definiert. Erweiterungen um proprietäre Instruktionen einer Schnittstelle sind nicht ausgeschlossen.

6.2.3 Entwicklung eines Komparators für den automatisierten Testablauf

Der für das CAST Werkzeug ausgewählte Testwerkzeugtyp *Komparator* wird in diesem Abschnitt entworfen. Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A9/ Die Auswertung von erzeugten Testprotokollen und dynamischen Testergebnissen muss automatisiert erfolgen (Komparator).

Wie in der Anforderung verlangt, soll der Komparator zu allen drei Phasen der Testdurchführung eingesetzt werden können: Während der Testvorbereitung, der Testdurchführung und dem Testabschluss. Während der Testvorbereitung und Testdurchführung können die Ergebnisse des Komparators verwendet werden, um den Verlauf der Testdurchführung dynamisch anzupassen. Somit sind verzweigte oder wiederkehrende Abläufe realisierbar. Im Testabschluss unterstützt

der Komparator das Erstellen des Testberichts, indem er gesammelte Daten aus dem Test vergleichbar macht.

In Abbildung 33 sind die Beziehung zwischen Testfall und Komparator, als Teil des Testframeworks, sowie die verschiedenen Formen von Komparatoren dargestellt. Prinzipiell vergleicht der Komparator einen erfassten Wert anhand eines gewählten logischen Komparators mit einem Referenzwert. Die verschiedenen Formen des Komparators zeichnen sich durch Variation der Referenzwerte (Quelle, Zeitpunkt, Anzahl) sowie der Variation der logischen Operatoren aus. Zudem müssen Einzelwerte, Werte im Bulk oder kontinuierliche Werte verglichen werden können.

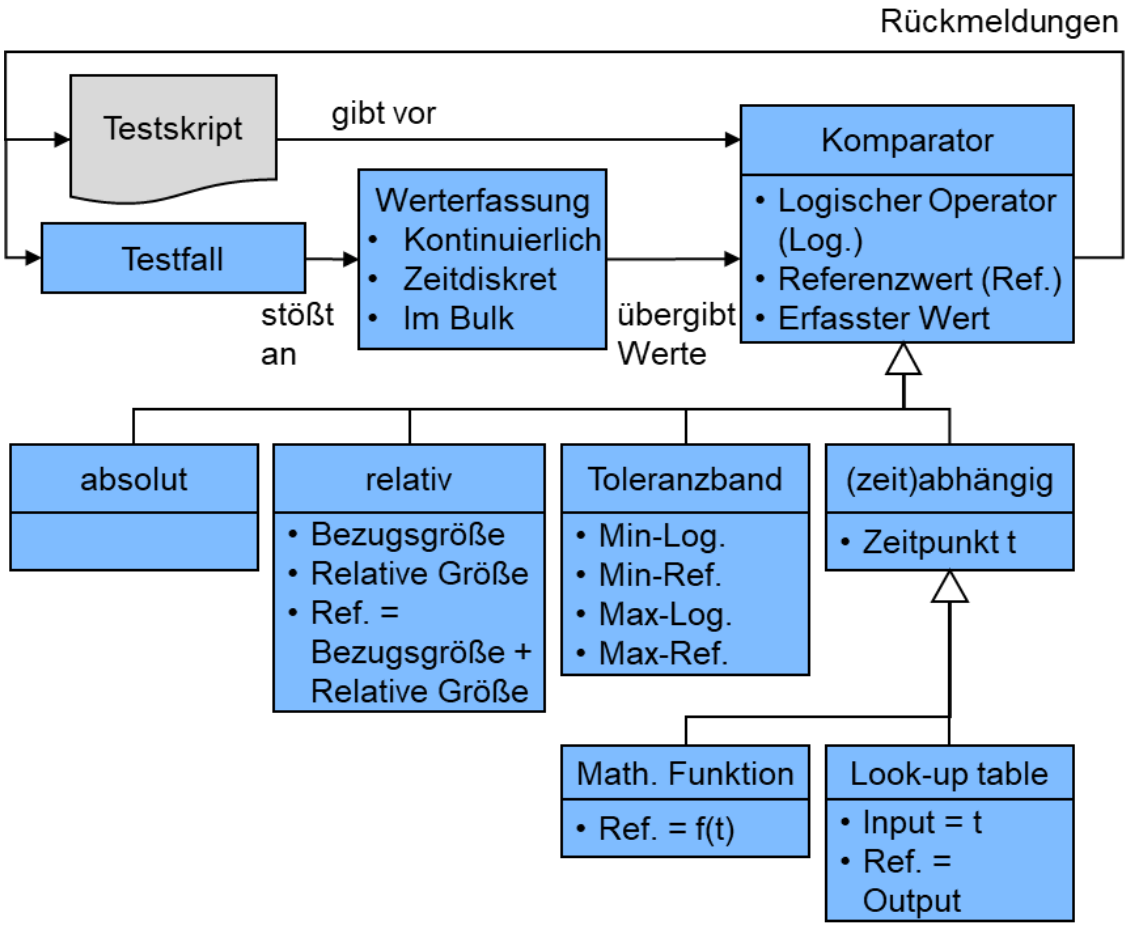


Abbildung 33 Einbettung und Funktionsprinzipien des Komparators

Die Abbildung zeigt die verschiedenen Formen des Komparators als Klassen der Superklasse „Komparator“, modelliert in UML. Der modellierte zeitabhängige „Referenzwert“ kann auch von einer anderen Größe abhängen. Bspw. kann die

„look-up table“ Chargennummern eines Auftrags enthalten, der beim Test Berücksichtigung findet. Das vom Benutzer erstellte Dokument „Testskript“ gibt für den einzelnen „Testfall“ den „logischen Operator“ und den „Referenzwert“ vor. Zudem wählt es auch die zu „erfassenden Werte“ aus. Der laufende Testfall stößt die „Werterfassung“ an und beauftragt das Testframework mit der Auswertung über den Komparator. Die Werterfassung erfolgt dabei:

- Kontinuierlich oder
- zeitdiskret und
- als Einzelwert oder
- im Bulk von mehreren Einzelwerten oder Wertebereichen.

Bei der Erstellung der Testfälle und der Auswahl des Komparators muss sich der Tester bewusst machen, wie das Verhalten der Steuerung spezifiziert worden ist, bzw. wie der Signalwert überwacht wird in der Steuerung, über die Zeit (Laufzeitüberwachung), über einen Trend (z. B. Schwellwerte), einen Einzelwert (Grenzwert), einen Wertebereich (Toleranzband) oder die Prüfung auf mehrere Wert gleichzeitig (z. B. Paarfehler).

6.2.4 Erreichen einer Herstellerunabhängigkeit für Steuerungssysteme

Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A4/ Die Definition von Testfällen im CAST Werkzeug muss für den Benutzer unabhängig von der herstellerspezifischen Ausprägung des SuT möglich sein (Herstellerunabhängigkeit)

In diesem Unterkapitel wird spezifischer auf die Schnittstellen zwischen Steuerung und HMI und dem CAST Werkzeug eingegangen. Ziel ist es, die Erstellung der Testfälle durch den Benutzer generisch und damit unabhängig von proprietären Eigenschaften des Steuerungssystems zu gestalten. Dies ermöglicht einen schnelleren Einstieg, da das Know-how über die Schnittstellen des Steuerungssystems entkoppelt wird vom Wissen über den Testentwurf. Weiterhin entsteht eine bessere Übertragbarkeit auf andere SuT und die Wiederverwendbarkeit von bereits erstellten Testfällen.

Mittels *Keyword-Driven Testing* (KDT) (schlüsselwortgetriebenes Testen) wird in der Softwaretechnik die Entwicklung von Testautomatisierungslösungen unterstützt. Die Anfänge des KDT gehen bis in die 1990er Jahre zurück (Buwalda 1999). Die Methode kann für alle Testarten angewandt werden (Norm ISO/IEC/IEEE 29119-5 2016, S. 4). Dazu müssen passende Schlüsselworte auf festgelegten Abstraktionsebenen definiert werden. Auf höheren Abstraktionsebenen können dann mehrere Funktionen des SuT hinter einem Schlüsselwort integriert sein. Im Folgenden wird nach der Richtlinie ISO/IEC/IEEE 29119-5 ein schlüsselwortgetriebenes Testen für die Steuerungssysteme im RT-HiL Aufbau entworfen. Der erste Schritt ist die Definition von passenden Abstraktionsebenen für die Schlüsselworte. Danach werden die notwendigen Funktionen auf den Abstraktionsebenen identifiziert und durch Vereinheitlichung festgelegt. Abschließend müssen auch die Rückgabewerte aus den Funktionen vereinheitlicht werden, da verschiedene Hersteller bei ähnlichen Funktionen andere Umfänge (z. B. 13 Bytes statt 2 Bytes) und Datentypen (z. B. Struct von Shorts statt Uword) benutzen.

Die Richtlinie definiert keine festen Ebenen, spricht von üblicherweise drei Ebenen (Norm ISO/IEC/IEEE 29119-5 2016, S. 7–10):

1. Endbenutzerebene,
2. Domänenebene,
3. Testschnittstellenebene.

Die drei Ebenen werden als passend erachtet für den Anwendungsfall der Steuerungssysteme. Die **Endbenutzerebene** wird mit Schlüsselworten besetzt, die der Benutzer von der Bedienfolgen der HMI kennt. Dazu werden die Knöpfe einer Steuertafel und die Funktionen der grafischen Benutzeroberfläche als Schlüsselworte formuliert. Auf der **Domänenebene** werden die Funktionen der Steuerung zusammengefasst. Dabei handelt es sich um das unidirektionale Abfragen von Werten aus der Steuerung. Bei der Wahl der Schlüsselworte für die Domänenebene wurde die VDW OPC UA Companion Specification zur Klassifizierung von NC-Steuerungssystemen als Anhaltspunkt herangezogen (Norm OPC 40502

2017). Diese definiert bereits eine große Anzahl an Schlüsselworten zur Interaktion mit NC-gesteuerten Maschinen. Zusätzlich wurden gängige NC-Steuerungssysteme untersucht. Die **Testschnittstellenebene** wird festgelegt als die proprietäre, technische Schnittstelle zu HMI und den Steuerungen. Die Schlüsselworte umfassen die in 6.2.2 festgelegten funktionalen und nicht-funktionalen Instruktionen. Die definierten Abstraktionsebenen und definierten Schlüsselwörter für den KDT-Ansatz sind Abbildung 34, Abbildung 35 und Abbildung 36 zu entnehmen.

Neben dem

Abbildung 34 zeigt einen Ausschnitt der für die Endnutzerebene definierten Schlüsselworte. Über das HMI interagiert der Endnutzer im manuellen Testfall mit dem Steuerungssystem. Daher erfolgte die Auswahl der Schlüsselworte durch Vereinheitlichung der HMI Funktionen gängiger Steuerungssysteme.

	Endnutzerebene
#	**Steuerung / HMI**
1	LoadProgram
2	UnloadProgram
3	CycleStart
4	CycleStop
5	Reset
6	Cancel
7	ChangeOpMode
8	JogPlus
9	JogMinus
	…

Abbildung 34 Definierte Schlüsselworte auf der Endnutzerebene

Bei der Domänenebene der HMI wird in Abbildung 35 unterschieden zwischen der Lösung *proprietäre Schnittstelle*, siehe Zeilen 1 bis 5 und der Lösung *grafische Schnittstelle*, siehe Zeilen 6 bis 21. Die Zeilen 5 und 21 in Abbildung 35 weisen darauf hin, dass weitere Schlüsselworte in der Domäne HMI gebraucht werden, allerdings wird aus Platzgründen auf deren Nennung verzichtet.

	Domänenebene	
#	**Steuerung**	**HMI**
1	CncSpindleActSpeed	GuiGetElementByName
2	CncProgramInfoActLineNumber	GuiGetAttributsOfElement
3	CncStateOpMode	GuiGetChildOfElement
4	CncLastAlarmNumber	GuiGetParentOfElement
5	CncLastAlarmTime	…
6	CncMachineAxisX	GuiMoveMouseTo
7	CncClock	GuiClickRightMouse
8	CncStateActTool	GuiReleaseRightMouse
9	CncProgramPointerProgName	GuiClickLeftMouse
10	CncProgramInfoDirectory	GuiReleseLeftMouse
11	CncResetActive	GuiEnterText
12	CncReadyActive	GuiPressKeyboardButtons
13	CncAxisPosition	GuiReleaseKeyboardButtons
14	Plc…	GuiMouseClick
15		GuiMouseDoubleClick
17		GuiMouseRightClick
18		GuiKeyboardCopy
19		GuiKeyboardPaste
20		GuiCaptureScreenshot
		…

Abbildung 35 Definierte Schlüsselworte auf der Domänenebene

Die Schlüsselworte auf der Testschnittstellenebene in Abbildung 36 sind zwischen Steuerung und HMI gleich, da beide Schnittstellen nach dem Server-Client Prinzip aufgebaut sind.

	Testschnittstellenebene	
#	**Steuerung**	**HMI**
1	SetConnectionInfo	SetConnectionInfo
2	Connect	Connect
3	Disconnect	Disconnect
4	GetStatusOfInterface	GetStatusOfInterface
5	ReadDataset	ReadDataset
6	WriteDataset	WriteDataset
7	TriggerMethod	TriggerMethod

Abbildung 36 Definierte Schlüsselworte auf der Testschnittstellenebene

Abbildung 37 zeigt schematisch den Ablauf bei Nutzung eines Schlüsselworts im Testablauf. Im Beispiel wird der Status des NC-Kanals 1 der Steuerung abgefragt. Als Zielsystem ist eine Fanuc NC ausgewählt. Das Modul „Testgenerierung“ erkennt ein Schlüsselwort und lässt dieses vom Modul „Keyword Mapping“ auflösen in das semantische Äquivalent bei Fanuc FOCAS: „cnc_statinfo“. Die Testgenerierung erstellt einen ausführbaren *Testschritt*[10], welcher sowohl die Verbindung zur FOCAS-Schnittstelle als auch die Abfrage des Status durchführt.

[10] Erläuterung siehe Seite 102

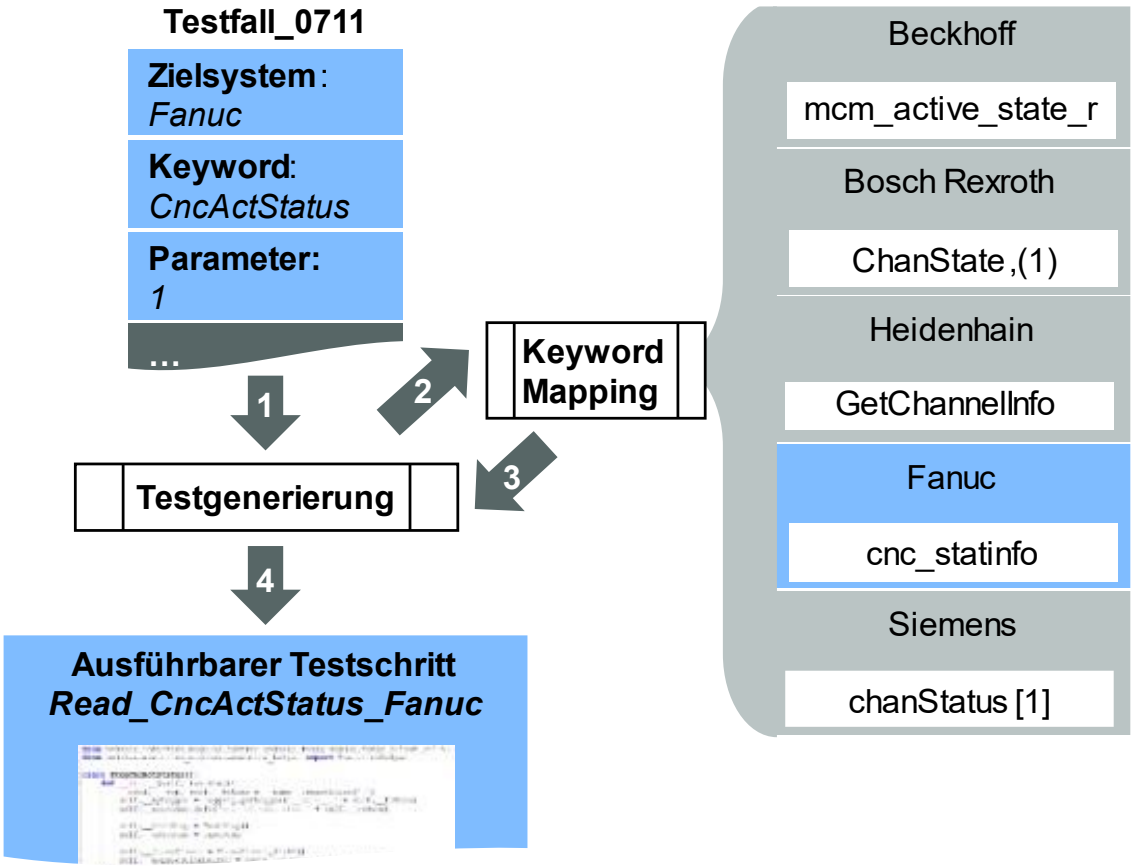


Abbildung 37 Funktionsweise der herstellerunabhängigen Definition und Ausführung von Testfällen an der Steuerungsschnittstelle

6.2.5 Virtueller Bediener zur Automatisierung der Mensch-Maschine-Schnittstelle

Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A2/ Neben einer realen Testumgebung muss eine virtuelle Testumgebung mittels RT-HiL unterstützt werden (System-under-Test real/virtuell)

Bei der Testdurchführung während der VIBN nimmt der Tester Bedienhandlungen an der HMI vor, siehe Beispieltestfälle in Abbildung 26. Diese Bedienhandlungen müssen auch zwischen dem CAST Werkzeug und dem RT-HiL System möglich sein, um eine vollautomatische Testdurchführung zu gewährleisten.

Um die HMI Schnittstelle des CAST Werkzeugs zu entwerfen, werden zunächst HMI Lösungen von bekannten Steuerungssystemen untersucht, um anschließend eine passende Lösung zu entwickeln.

In den gängigen Steuerungssystemen am Markt werden zwei Techniken verwendet, um die HMI zu betreiben. Erstens, die Nutzung einer Thin Client-Architektur, (erläutert z.B. in Balzert et al. 2011, S. 194) und, zweitens, die Nutzung einer PC Applikation als HMI. In industriellen Fertigungssystemen findet meist die erste Variante Anwendung, da sie deutlich kostengünstiger ist. Der Thin Client übernimmt dabei die Anzeige, ähnlich wie ein Monitor. Zudem erfasst er Maus- und Tastatureingaben des Bedieners, die er an den eigentlichen HMI Prozess weitergibt. Die Verarbeitung und Auswertung der Eingabebefehle finden auf einem entfernten System statt. Als Kommunikation zwischen Thin Client und HMI Prozess wird z. B. das Remote Framebuffer Protocol genutzt oder proprietär erweitert. Mittels einer Client-Server Struktur wird dieses Prinzip beim sogenannten Virtual Network Computing (VNC) genutzt. VNC ist ein plattformunabhängiges Prinzip und es existieren einige freiverfügbare Softwareapplikationen. Die HMI von Siemens kann z. B. mit der freien VNC Applikation *UltraVNC*[11] angesprochen werden. Fanuc bspw. hat eine proprietäre Umsetzung des VNC Prinzips und daher auch eine proprietäre Applikation dafür: *CNC Screen Display Function*. Für diese Systeme benötigt es entweder die Automatisierung des VNC Clients oder die Umsetzung eines eigenen Clients, sofern das Übertragungsprotokoll frei zugänglich ist. Andere HMI Lösungen, welche die HMI-Hardware und -Software auf einer gemeinsamen PC-Plattform aufgesetzt haben, können mit der nativen Automatisierung (z. B. Windows Automation unter Microsoft Windows) automatisiert werden. Diese Lösung funktioniert z. B. für Beckhoff HMIs.

Zur Veranschaulichung sind in Abbildung 38 verschiedene Elemente einer NC-Bedienoberfläche und die Variation, in welcher die Elemente auftreten können, dargestellt. Die Menüschaltfläche „Diagnose“ in der ersten Zeile unterscheidet sich im gedrücktem und nichtgedrücktem Zustand nur leicht durch ihre Farbe und den Rand der Schaltfläche. Bei nativen Applikationen kann hingegen der Kontext des Elements eindeutig über dessen Namen und den Namen des Attributs, z. B. „pressed – true/false“, ausgelesen werden. Um die grafikbasierte

[11] https://www.uvnc.com

Oberflächenautomatisierung trotzdem anwendbar und robuster zu machen, wird ein modellbasierter Zugriff auf die HMI erarbeitet.

	aktiv	inaktiv	aktiviert	ausgewählt
Menü		--		--
Schaltfläche		--		--
Listenelement		--		

Abbildung 38 Übersicht zu drei verschiedenen grafischen Elementen (Zeilen) und deren möglichen Zuständen (Spalten)

Die Lösung zum modellbasierten Zugriff besteht aus zwei Komponenten: Zunächst wird ein hierarchisches Modell der zu automatisierenden Oberflächen erstellt. In diesem Modell sind die Elemente, deren Verbindungen und möglichen Zustände der HMI Applikation abgebildet. Die zweite Komponente sind die, wie in Abbildung 38 dargestellt, Bilder der einzelnen Elemente der Oberfläche in ihren unterschiedlichen Zuständen. Der Aufbau des Modells der HMI Applikation entspricht einer Baumstruktur, welche sich immer weiter in die Tiefe verzweigt. Das Modell wird an jedem Knoten mit dem zum Zustand des Elements passenden Screenshot erweitert. Jede Verbindungskante zwischen zwei Elementen entspricht dem Auslösen einer Aktion auf der HMI Oberfläche. Zuletzt werden die Schlüsselworte der Domänenebene *HMI* mit einem Pfad durch das Modell verknüpft, beispielhaft gezeigt am Schlüsselwort *GuiSelectNcProgram*, die Auswahl eines NC-Programms für den Testablauf:

1. Wechsel in das Menü des Dateimanagers mittels der Schaltflächen „Menü“ und „Dateimanager“
2. Navigation zum Zielverzeichnis mit dem NC-Programm, Öffnen mit Doppelklick

3. Navigation auf den Namen des gewünschten NC-Programms mittels der Tastatur-Pfeiltasten
4. Anwahl des NC-Programms für den aktiven NC-Kanal mittels der Schaltfläche „Anwahl“

Abbildung 39 zeigt die für die Bedienfolge „GuiSelectNcProgramm(‚Kreisfahrt‘)“ notwendigen Schritte, die dabei genutzten Elemente und Aktionen sowie die Zustandsänderung der Elemente.

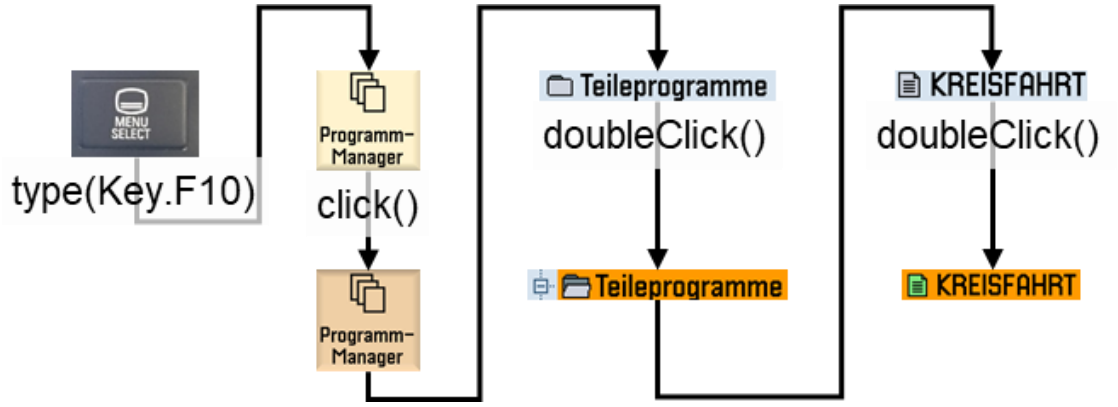


Abbildung 39 Beispielhafte Bedienfolge GuiSelectNcProgram mit allen Schaltflächen und Knöpfen zur Auswahl des NC-Programms „Kreisfahrt“

Der virtuelle Bediener zur Automatisierung der Bedienbefehle während der Testdurchführung am HiL System besteht somit aus einem Client (VNC oder nativer Zugriff) und dem Modell der HMI Applikation. Durch das modellbasierte Vorgehen für den virtuellen Bediener wird einerseits die Automatisierung von Bedienhandlungen bei der Testdurchführung ermöglicht. Andererseits wird auch ein methodisches Vorgehen für die Vorbereitung zur Automatisierung gegeben: Erstens, wird ein korrektes und wiederholgenaues Anlegen der Bedienhandlungen durch einen Tester ermöglicht. Zweitens, ist die Pflege der Modelle, z. B. nach einem Softwareupdate seitens des Steuerungsherstellers, durch Austausch der Screenshots oder Bezeichner im Modell möglich.

Der Aufwand für die Erstellung des Modells kann stark reduziert werden, da meist nur einige Bedienfolgen der gesamten HMI Applikation für den Test gebraucht werden. Auf Basis des Modells kann dann ein Skript generiert werden, um die Durchführung der Bedienhandlungen zu automatisieren. Durch die Erweiterung

des Modells der HMI mit den Aktionen („click()“, „doubleClick()“, „type()“, etc.) zu jedem Elementtyp („Button“, „ListElement“, etc.) wird eine automatisierte Erstellung von Bedienhandlungen ermöglicht: Ein Pfad durch das Modell besteht aus den Knoten (Elemente auf der HMI und dessen Zustand) und Kanten (Aktionen zur Manipulation) und dient als Vorlage einer modellbasierten Generierung.

6.2.6 Entwicklung einer Sprache für die Erstellung von Testfällen

Dieser Abschnitt berücksichtigt die folgenden Anforderungen aus Kapitel 3:

/A5/ Die Erstellung der Testfälle erfolgt in der Sprache des Maschinenbauers (Geeignete Testfallmodellierung)

/A8/ Die Testdurchführung muss einen Dauerlauf ermöglichen (Dauertest)

Neben der automatischen Testdurchführung und -auswertung entscheidet die korrekte Erstellung der Testfälle maßgeblich über den Erfolg eines Tests. Neben Korrektheit entscheidet die Art und Weise der Erstellung maßgeblich über die Akzeptanz beim Benutzer. Für beide Anforderungen müssen eine Sprache und eine Methode zum Testentwurf und zur Testrealisierung entworfen werden. In diesem Kapitel wird die Sprache entworfen, im folgenden Kapitel 6.2.7 die Methode.

Vor dem Entwurf der Sprache wird deren Verwendung spezifiziert. Der Begriff des Testfalls wurde schon mehrfach benutzt, ist aber nicht ausreichend und muss differenziert werden.

Als „Testschritt“ wird ein Blackbox-Element eines Testfalls bezeichnet. Ein „Testfall“ verknüpft mehrere Testschritte. Innerhalb eines Testschritts können sich wiederum rekursiv weitere Testschritte befinden. Testschritte werden im CAST Werkzeug mittels einer Bibliothek bereitgestellt, siehe nächstes Kapitel 6.2.7.

Werden mehrere Testfälle in einer Testdurchführung eingeplant und ausgeführt, so wird dies als „Testszenario“ bezeichnet. Angelehnt an das Szenario Testen aus der Softwaretechnik (Kaner 2003; Norm ISO/IEC/IEEE 29119-1 2013, S. 5) verknüpft ein Testszenario mehrere Testfälle in solcher Weise, dass damit eine aufwändige Abfolge an einem Fertigungssystem getestet werden kann. Vorlage

der Szenarien sind bspw. Benutzervorgehen (Use Case) und Anforderungen des Kunden.

Zur Durchführung eines Testszenarios bedarf es einer Beschreibung des Testsystems (RT-HiL), dafür wird eine „Testkonfiguration“ angelegt. Die Testkonfiguration wird um weitere Informationen ergänzt, z. B. Pfade für Datenimport und -export, sowie Metadaten wie Datum, Uhrzeit und Name des Testers. Während (Kreusch 2002, S. 87) die Testkonfiguration als die gesamte Beschreibung eines Tests versteht, wird in dieser Arbeit eine Unterscheidung zwischen Testszenario, Testfall und Testkonfiguration getroffen. Dadurch bleiben Testfälle und Testszenarien unabhängig vom Testsystem. In Abbildung 40 wird die Testdurchführung im Testframework mit den beteiligten Daten und Dokumenten dargestellt. Vor der Testdurchführung wird neben Testfällen, Testszenarien, Testkonfiguration eine Vorlage für den „Testbericht“ benötigt. Während der Testdurchführung entsteht eine Vielzahl an dynamische Testdaten, welche über das „Testprotokoll“ gesammelt werden. Der Testbericht entsteht durch Aufbereitung des Testprotokolls, unter Berücksichtigung der Vorlage, und bietet nach der Testdurchführung eine Übersicht zu den Testergebnissen. Mit Abschluss der Testdurchführung wird die gesamte Testdurchführung in Form der erstellten Dokumente und Daten in ein „Testarchiv“ überführt. Dadurch werden Testergebnisse nachvollziehbar, inklusive der Dokumente aus der Testdurchführung, abgespeichert.

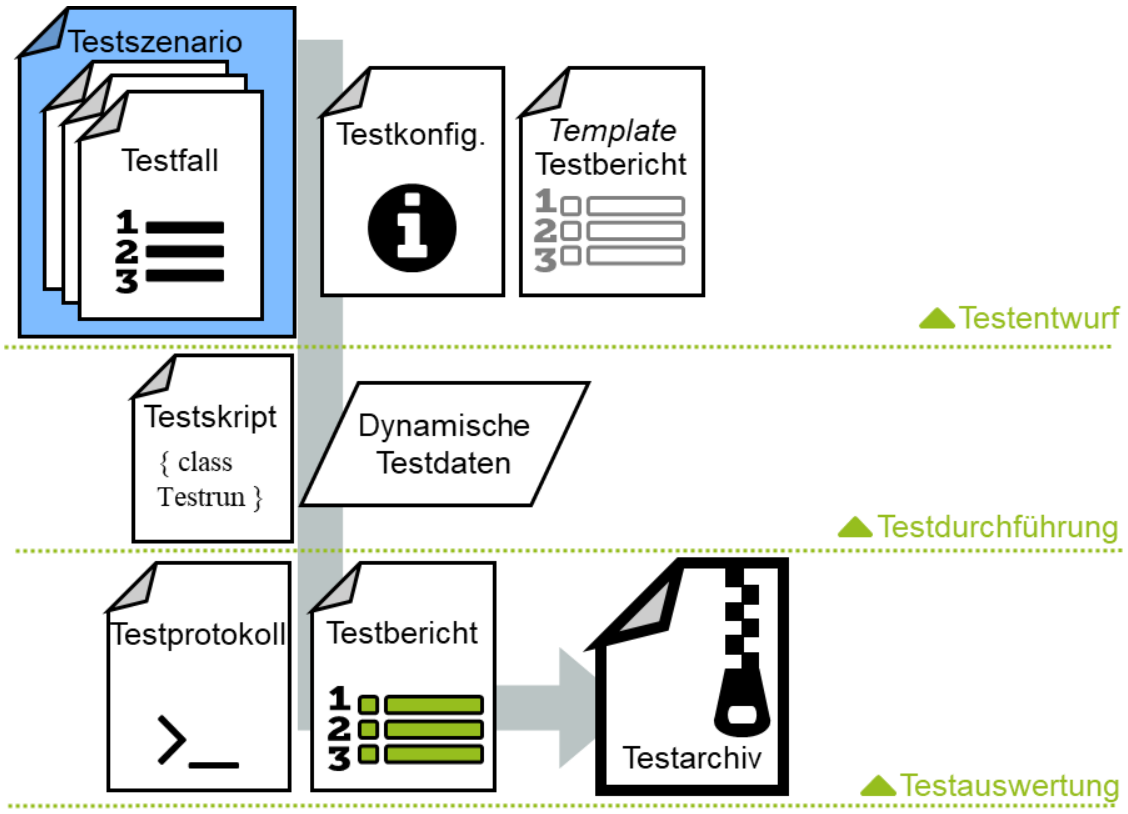


Abbildung 40 Phasen und Artefakte der ganzheitlichen Testautomatisierung

Für den Entwurf der Sprache werden verschiedene Aspekte berücksichtigt: Es werden die späteren Benutzer, die Sachverhalte zur Abbildung und bekannte Beschreibungsformen aus der Automatisierungstechnik betrachtet.

Beschreibungs- und Programmiersprachen im Maschinen- und Anlagenbau sind in ihrer Mehrzahl grafische Sprachen, siehe (Norm VDI/VDE 3681 2005). In der VDI/VDE 3681, die Beschreibungssprachen der Automatisierungstechnik untersucht, darunter auch die Programmiersprachen für SPS (Norm DIN EN 61131-3 2013), werden 11 von 14 der untersuchten Sprachen der grafischen Darstellung zugeordnet und nur 6 der textuellen oder mathematisch-symbolischen Darstellung. Die grafische Darstellung überwiegt auch bei den Modellierungssprachen zum Erstellen von Simulationsmodellen. Abbildung 41 zeigt Beispiele beider Disziplinen, links die Funktionsbausteinsprache und der Kontaktplan zur SPS-Programmierung, rechts Ausschnitte aus Simulationsmodellen zweier Simulationswerkzeuge.

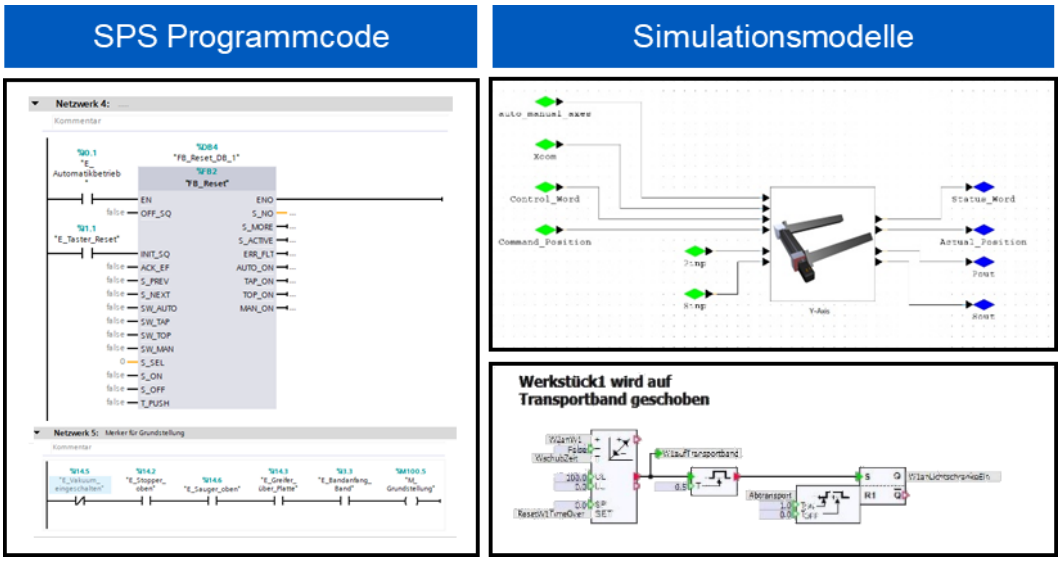


Abbildung 41 Auswahl an grafischen Programmier- und Modellierungssprachen im Umfeld der Fertigungstechnik

Um dem geschulten Personal aus der Erstellung von SPS Programmen oder Simulationsmodellen den Einstieg in die Modellierung von Testfällen zu erleichtern und damit die Akzeptanz solcher Werkzeuge zu steigern, wird für den Testentwurf primär eine grafische Modellierung konzipiert. Ganz ohne Anteil an textuellen Programmiersprachen wird sich aber ein Testfall nicht erstellen lassen, bspw. um die Schnittstellen des RT-HiL Systems anbinden zu können. Da eine optimale Benutzerfreundlichkeit im besten Fall eine Wahlmöglichkeit für den Benutzer lässt, ist die grafische Modellierung der Testabläufe nicht als exklusiv anzusehen. Die Priorität sieht den Entwurf zunächst als grafische Beschreibungsform vor. Bei der Nutzung einer textuellen Beschreibung parallel zum grafischen Entwurf ist auf eine Kompatibilität der beiden Beschreibungsformen zu achten, sodass Sachverhalte zwischen den Beschreibungsformen 1:1 übertragen werden können.

In der Beschreibung eines Testfalls müssen die folgenden Sachverhalte abgebildet werden:

1. Beschreiben von Abläufen
2. Mathematische und logische Operationen
3. Datenverarbeitung

Ein Testfall beschreibt, genau wie ein Steuerungsprogramm, ein Verhalten. Gleichzeitig werden zeitliche Aspekte dargestellt und auch eine Struktur beschrieben. Im Gegensatz zum Ablauf einer zyklischen SPS oder eines zyklischen Simulationsmodells ist der Durchlauf einer Testdurchführung ein sequentieller Ablauf. Der Test benötigt dabei Abläufe wie aus der Hochsprachenprogrammierung: Dazu zählen Schleifen, Verzweigungen und Sprünge. In Abbildung 42 sind die möglichen Abläufe in Ablaufgraphen dargestellt. Diese müssen mit der Beschreibungssprache für Testfälle abbildbar sein und vom Testframework zur Laufzeit umgesetzt werden. Die drei o. g. Sachverhalte stehen einer grafischen Darstellung von Testfällen nicht konträr gegenüber, somit bleibt die grafische Darstellung als primäre Beschreibungsform erhalten.

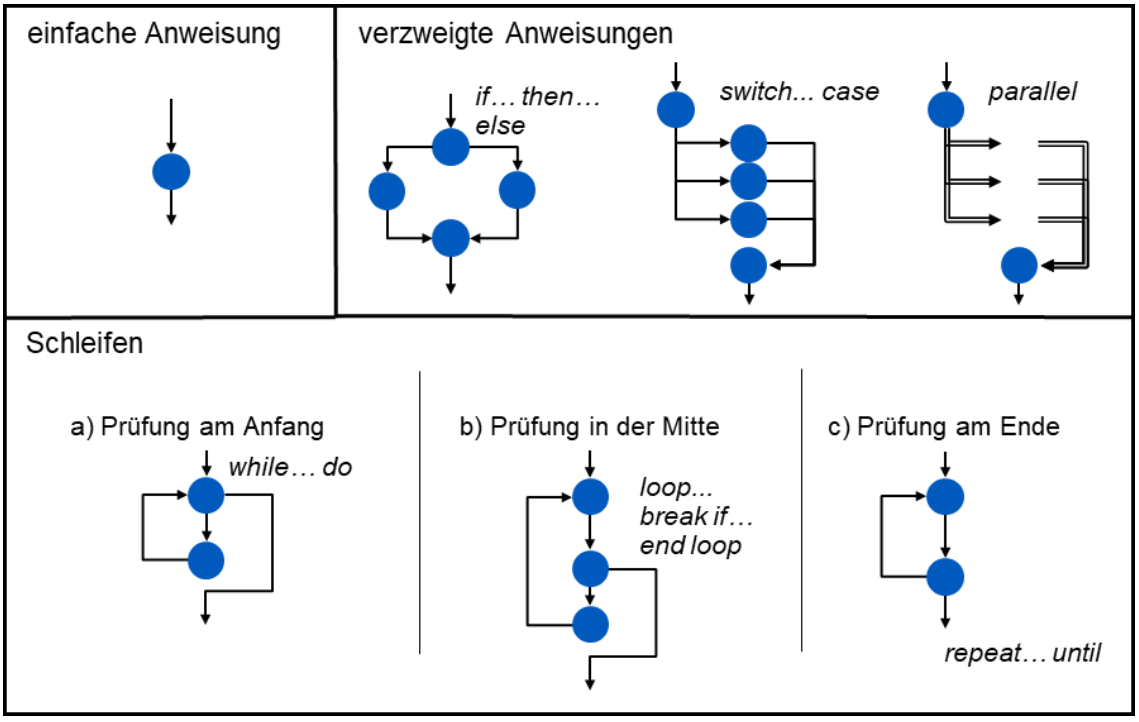


Abbildung 42 Darstellung verschiedener Abläufe in der Testdurchführung (erweitert nach (Frühauf et al. 2007, S. 58))

Einfluss der Testdurchführung auf die Sprache

Neben der automatischen Testdurchführung ist auch eine manuelle Steuerung im Ablauf notwendig. Beim Erstellen der Testfälle kann so deren Prüfung vor der Testdurchführung Schritt für Schritt erfolgen. Auch komplexe oder schnell wech-

selnde Zustände einer Testdurchführung können so geprüft werden. Dazu gehören das Pausieren der Testdurchführung, das Setzen von Breakpoints sowie das schrittweise Ausführen der Testdurchführung. Für die manuelle Teststeuerung müssen strukturell abgrenzbare Einheiten definiert werden, dies sind die Testschritt. Zeitlich und logisch vor bzw. nach den Testschritten kann die Testdurchführung gezielt pausiert werden. Die Testschritte bestimmen damit als kleinste Einheit die Schrittweite in einem Testfall. Anschaulich gesprochen sind die Testschritte die blauen Knoten in den Abläufen in Abbildung 42.

Durch die rechnerunterstütze Ausführung der Tests im CAST Werkzeug können Testdurchführungen nicht nur aufwandsarm ausgeführt, sondern auch parallelisiert oder in Schleifen wiederholt werden. Dies ermöglicht die Durchführung von Dauertests in Testläufen über Nacht oder während Wochenenden.

(Norm VDI 4499 Blatt 1 2008) beschreibt in Kapitel 4.1, wie Prozesse definiert für die Digitale Fabrik abgebildet werden können. Als Möglichkeiten werden SADT und UML genannt. Beides sind grafische Beschreibungsformen für Prozesse. Structured Analysis and Design Technique (SADT) geht zurück auf (Ross 1977). Die Idee von Ross war, die grafische Beschreibung von Gesamtsystemen inklusive deren Dekomposition zu zeigen. Ross beschreibt zwei interessante Ansätze, welche zur Beschreibung von einzelnen Testschritten sowie von komplexeren Testabläufen genutzt werden sollen. Das SADT Basiselement besteht aus Function, Input, Output, Control und Mechanism und der Structured Decomposition. Dieses Konzept wird aufgegriffen, um eine Beschreibungsform für den Ablauf eines Testfalls zu erhalten.

In Abbildung 43 wird ein Testschritt „n“ als Blackbox inklusive seiner Peripherie dargestellt. Die unmittelbare Peripherie des Testschritts wird mittels Pfeile und den angrenzten Testschritten „n-1“ und „n+1“ verdeutlicht.

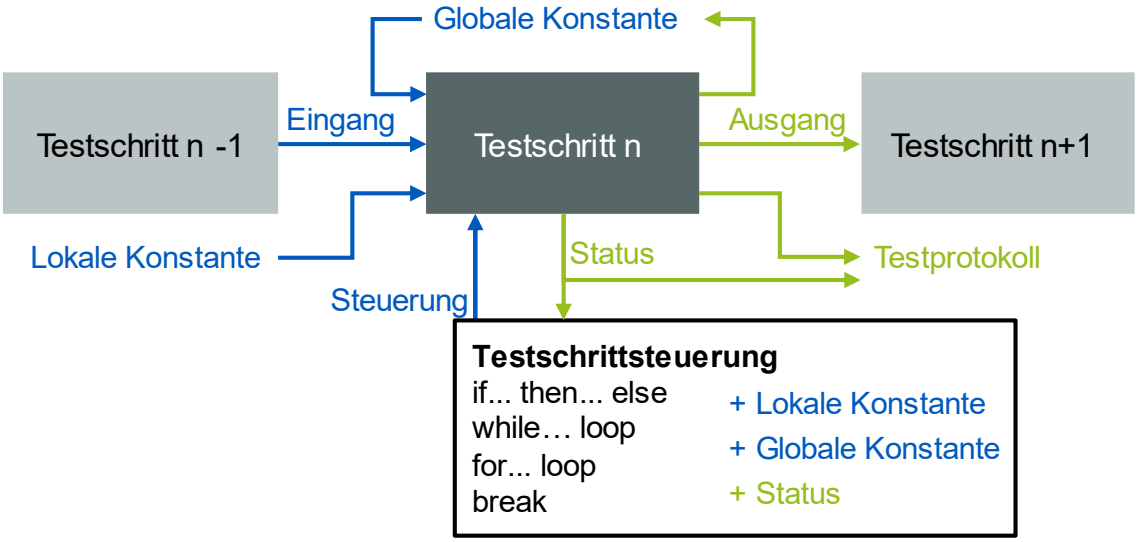


Abbildung 43 Modellierung eines Testschritts als Blackbox mit seiner Umgebung

In Blau werden eingehende Pfeile zum Testschritt n dargestellt. Die grünen Pfeile weisen auf ausgehende Informationen hin.

Eingang

Als Eingangswerte eines Testschritts werden die Werte bezeichnet, welche aus einem oder mehreren vorherigen Testschritten als Ausgangswerte entstanden sind und im aktuellen Testschritt zur Abarbeitung benötigt werden.

Bsp.: Der Vorgängertestschritt zählt die Anzahl an Aufrufen einer SPS-Funktion. Der Nachfolgetestschritt vergleicht die Anzahl mit einem Sollwert und leitet das Ergebnis („true/false“) weiter.

Lokale Konstante

Als Eingang für einen Testschritt können einzelne Datenwerte dienen. Ähnlich eines Aufrufparameters in einer Funktion sind diese konstant und nur im Kontext dieses Aufrufs sichtbar.

Bsp.: Der Testschritt besitzt einen Parameter, der bei der Ausführung bestimmt, wie viele Werkstücke in die Simulation geladen werden.

Testschrittsteuerung

Der Ablauf eines Testfalls und die darin enthaltenen Testschritte werden über die Testschrittsteuerung (TSS) bestimmt. Die TSS nimmt lokale, globale Konstanten

und den Status eines Vorgängertestschritts, um zu bestimmen, ob und wie oft ein Testschritt ausgeführt werden soll. Je nach Status eines Vorgängertestschritts ist es notwendig, die Ausführung eines späteren Testschritts zu beeinflussen.

Bsp.: Die lokale Konstante aus dem Beispiel bestimmt die Anzahl der Werkstücke. Die TSS nutzt die Konstante als Parameter in einer *for*-Schleife, um x-mal den Testschritt zum Einlasten eines Werkstücks in die Simulation aufzurufen. Bei der Betrachtung des Status kann z. B. das Warten auf eine Wertänderung das weitere Durchschreiten des Testfalls verzögern. Der Status bleibt bis dahin auf dem Wert „failed“.

Globale Konstante

Einige Datenwerte werden testfall- und testschrittübergreifend benötigt. Diese werden als globale Konstante angelegt. Jeder Testschritt hat die Möglichkeit, lesend und schreibend auf die globalen Konstanten zuzugreifen.

Bsp.: Die Verbindungsdaten zwischen CAST Werkzeug und RT-HiL in Form von IP-Adressen und Ports oder Verzeichnisse für den Datenzugriff lokal und remote.

Status

Die Abarbeitung eines Testschritts ist mit einer Abbruchbedingung verknüpft. Die Abbruchbedingung dient zum einen dazu, ein definiertes Ende für den Ablauf zu erhalten. Zum anderen wird mittels der Abbruchbedingung ermittelt, ob die Ausführung dieses Testschritts erfolgreich oder fehlerhaft war.

Bsp.: Zwei Zahlenwerte werden verglichen, um ein erfolgreiches Testende zu ermitteln. Sind die Werte identisch, war der Testschritt und damit der Testfall erfolgreich und wird als „passed*“*, andernfalls als „failed*“* gekennzeichnet.

Ausgang

Der Ausgang eines Testschritts enthält ein Ergebnis in Form eines Datenwerts in gängigen Datenformaten wie Ganzzahl, Gleitkommazahl, Wahrheitswert oder Zeichenfolge. Der Datenwert am Ausgang kann als Eingang für einen weiteren Testschritt dienen, in eine globale Konstante geschrieben werden oder in der TSS weiterverarbeitet werden.

Bsp.: Der Testschritt vergleicht den Namen zweier Zeichenfolgen, die ein NC-Programmname repräsentieren, und gibt das Ergebnis an den nächsten Testschritt weiter, um die Anwahl des NC-Programms durchzuführen.

Testprotokoll

Jede in einem Testfall und Testschritt durchgeführte Aktion wird in ein Testprotokoll geschrieben. Dabei unterscheidet das Loglevel, welche Daten relevant sind, bspw. können nur die zum Test gehörigen Aktionen protokolliert werden oder auch interne Aktionen des Testframeworks. Für die Aufzeichnung einer Aktion müssen weitere Metainformationen abgespeichert werden: Zeitstempel der Aktion an der Quelle, Sender der Nachrichten und Datenwerte.

Bsp.: Der Testschritt zum Pausieren der Testdurchführung wurde mit der Zeitdauer von vier Sekunden aufgerufen, siehe Abbildung 44.

```
2018-11-16 11:23:53.548 __main__.control_Validate_Act_Spindle_Speed [INFO]: ------------------------------------------
2018-11-16 11:23:53.548 __main__.control_Validate_Act_Spindle_Speed [STATUS]: # Start of test step step_PauseExecution
2018-11-16 11:23:53.548 __main__.control_Validate_Act_Spindle_Speed [INFO]: # Remarks:
2018-11-16 11:23:53.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: # Parameter:
2018-11-16 11:23:53.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: #       time_to_pause: 4
2018-11-16 11:23:53.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: # Repeat enabled: False
2018-11-16 11:23:53.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: ------------------------------------------
2018-11-16 11:23:53.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: Run conditions met
2018-11-16 11:23:53.549 __main__.step_PauseExecution [INFO]: Init of test step step_PauseExecution
2018-11-16 11:23:53.549 __main__.step_PauseExecution [INFO]: Waiting for 4 seconds
2018-11-16 11:23:53.549 __main__.step_PauseExecution [STATUS]: Waiting until 2018-11-16 11:23:57
2018-11-16 11:23:57.549 __main__.step_PauseExecution [STATUS]: Waiting finished
2018-11-16 11:23:57.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: ------------------------------------------
2018-11-16 11:23:57.549 __main__.control_Validate_Act_Spindle_Speed [STATUS]: * End of test step step_PauseExecution
2018-11-16 11:23:57.549 __main__.control_Validate_Act_Spindle_Speed [STATUS]: * Result: SUCCESSFUL
2018-11-16 11:23:57.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: * Duration: 0:00:04.001
2018-11-16 11:23:57.549 __main__.control_Validate_Act_Spindle_Speed [INFO]: ------------------------------------------
```

Abbildung 44 Zeilen des Testprotokolls für den Testschritt "PauseExecution"

Nachdem die Sprache zur Erstellung von Testfällen entworfen wurde, wird im verbleibenden Teil dieses Unterkapitels auf die Dokumente und deren Abhängigkeit für die Testdurchführung eingegangen.

6.2.7 Wiederverwendung von Testfällen durch Modularisierung

Dieses Unterkapitel erfüllt die folgende Anforderung aus Kapitel 3:

/A7/ Auf Wiederverwendung ausgelegtes Erstellen der Testfälle (Wiederverwendung)

Durch ganzheitliche Testautomatisierung entfällt der manuelle Aufwand der Testdurchführung und -auswertung. Um dem Begriff der Ganzheitlichkeit gerecht zu

werden, setzt sich die Methodik in dieser Arbeit auch mit dem Testentwurf und der Testrealisierung auseinander. Ziel ist es, den Aufwand und die Fehleranfälligkeit bei der Erstellung eines Tests zu reduzieren.

Ein erfolgreiches Konzept, aus der Softwareentwicklung den Aufwand in der Entwicklung zu verringern, ist die Wiederverwendung. Die Einführung und das Praktizieren von Wiederverwendung ist mit Mehraufwand verbunden, der sich erst nach mehrmaligem Einsatz lohnt (Puntigam 2007, S. 32ff). Zur erfolgreichen Anwendung von Wiederverwendung in Entwicklungsprozessen empfiehlt die Richtlinie (Norm VDI/VDE 3695 Blatt 3 2010, S. 7–15) unter anderem: die Schaffung einer Ablagemöglichkeit und das Erstellen von Referenzmodellen zur Wiederverwendung.

Für eine optimale Methode der Wiederverwendung muss sowohl das CAST Werkzeug ausgelegt sein, als auch das Vorgehen des Benutzers methodisch sein. Die Schaffung der Ablagemöglichkeit für das CAST Werkzeug ist in Form der Datenhaltung vorgesehen, Abbildung 27 auf Seite 77. Die Ablagemöglichkeit soll eine Bibliothek an wiederverwendbaren Elementen enthalten, aus welchen Testfälle durch Parametrierung erstellt werden können. In der Bibliothek können Testschritte, Testfälle und ganze Testszenarien abgelegt werden.

Für den zweiten Aspekt der Wiederverwendung, das Referenzmodell, wird in dieser Arbeit ein Modell basierend auf der Hierarchie der Virtuellen Steuerungstestbench vorgeschlagen. Dadurch entstehen mehrere Ebenen, auf welchen sich der Benutzer beim Testentwurf an wiederverwendbaren Elementen bedienen kann. Abbildung 45 zeigt den Aufbau der Bibliothek. Es werden drei vertikale Ebenen definiert:

I. die Basisebene,
II. die Testprozessebene und
III. die Testfallebene.

Zu jeder Ebene ist in der Abbildung beschriftet, welche Rolle im Unternehmen die Erstellung und Wartung der Testschritte/Testfälle übernehmen sollte. Auf Ebene I, der **Basisebene**, werden die Testtreiberfunktionen und damit die Schnittstellen zwischen CAST Werkzeug und RT-HiL System verwaltet. Ebenso

liegen auf dieser Ebene die Testframeworkfunktionen. Diese sind der Standardumfang des CAST Werkzeugs, der unabhängig von Benutzer und Anwendungsfall benötigt wird, um Testfälle zu erstellen. Bspw. das Einlesen von Daten aus einer Quelle, das Konvertieren von Datentypen und mathematische und logische Operatoren. Für die Erstellung der Funktionen benötigt es Technologie- und Softwareexperten.

Auf Ebene II, der **Testprozessebene**, werden in der Bibliothek Funktionen aggregiert, welche die Testdurchführung steuern (Teststeuerungsfunktionen) und die Testauswertung durchführen (Testauswertungsfunktionen). Die Testdurchführung muss bspw. immer wieder Zwischenergebnisse evaluieren und an die Testschrittsteuerung weitergeben. Die Testauswertung erstellt aufgrund von Zwischenergebnissen und Vorgaben für den Testbericht den finalen Testbericht. Dafür „zieht“ sie alle relevanten Informationen aus dem Testprotokoll auf die oberste Ebene des Testberichts, z. B. die konkreten Fehlermeldungen und zugehörigen Werte, inklusive ihrer Entstehungskette. Für die Funktionen dieser Ebene benötigt es Testexperten.

Ebene III wird als **Testfallebene** bezeichnet. Die Ebene wird weiterhin horizontal geteilt, um mehrere Anwendungsfelder zu ermöglichen. In Abbildung 45 sind beispielhaft drei Anwendungsfelder gezeigt. Ganz links in grau sind die Testarten über den Entwicklungsprozess dargestellt. In der Mitte in hellblau spezielle Bibliotheken für den Test von Fertigungssystemen, ganz rechts in hellgrün eine Spezialisierungsbibliothek für den Test einer komponentenorientierten Entwicklung eines Fertigungssystems: Herrscht z. B. ein baukastenbasiertes Entwickeln im Unternehmen vor, können die Testfälle nach der gleichen Unterteilung in der Bibliothek sortiert werden. Zur Veranschaulichung wurde eine Zerlegung des Werkzeugwechsler (WZW)-Systems einer Werkzeugmaschine genutzt. Im Einsatz dieser Spezialisierungsbibliothek kann bei Änderungen an der Software für den WZW-Arm direkt auf alle Testfälle aus der Bibliothek regressiv getestet werden. Für die Testfallebene benötigt es Systemexperten, die mit den Fertigungssystemen vertraut sind und Testexperten, um eine passende Einteilung zu finden.

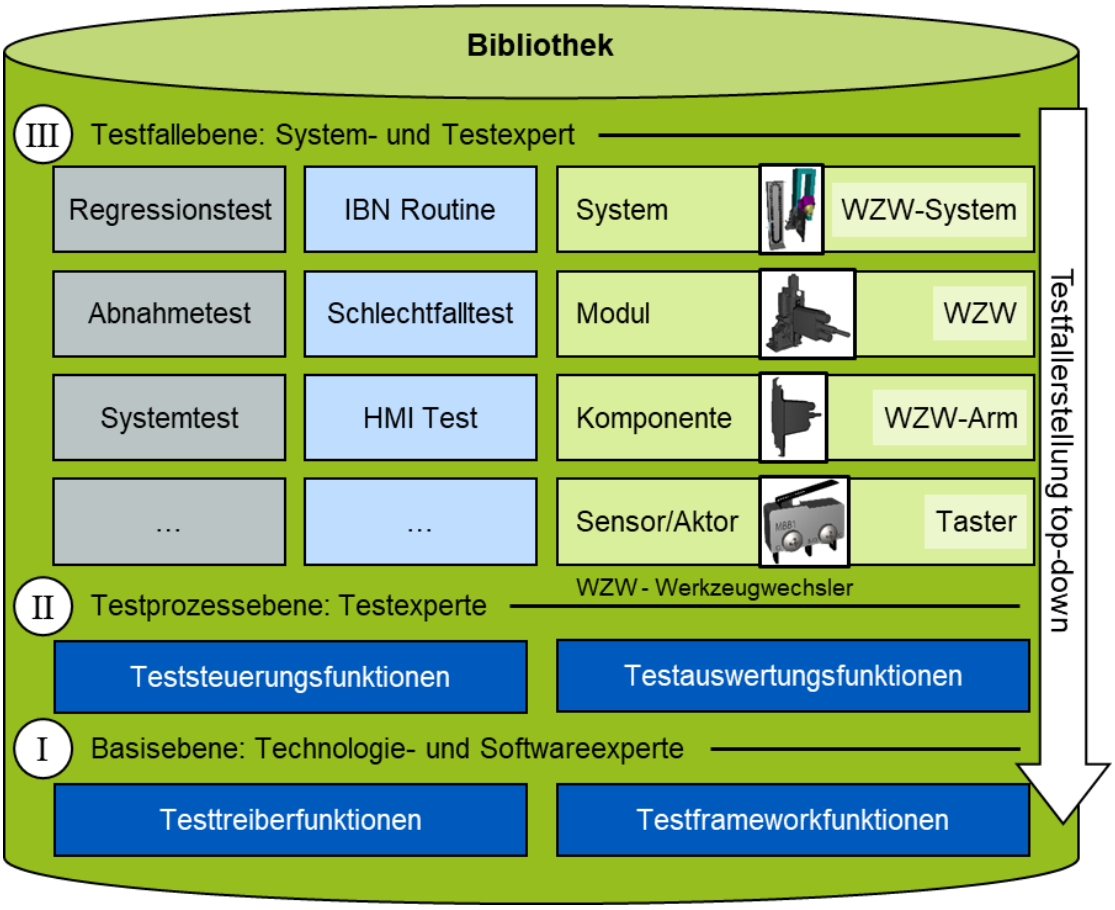


Abbildung 45 Prinzip zum Aufbau der Bibliothek: I Basisebene II Testprozessebene III Testfallebene mit horizontalen Anwendungsfeldern, grau: Testarten aus der Entwicklung; hellblau: spezifische Testarten; hellgrün: nach Baukasten

Die Unterteilung der Bibliothek zur Wiederverwendung bezweckt die Clusterung von zugehörigen Testfällen und ermöglicht damit ein schnelles und übersichtliches Arbeiten. Auf der Ebene III kann die Übersichtlichkeit weiter gesteigert werden, wenn eine horizontale Einteilung nach eigenen Anwendungsfeldern erstellt wird, denn mit fortlaufender Erweiterung der Bibliothek wird die Ebene III am meisten Testschritte beinhalten. Die Einteilung der Bibliothek in die drei vertikalen Ebenen ist gleichzeitig eine Einteilung nach den an der Testautomatisierung beteiligten Rollen. Auf Ebene I befinden sich die technischen Experten in Bezug auf RT-HiL Systeme und Schnittstellen. Auf Ebene II befinden sich die Testautomatisierungsexperten und auf Ebene III die Test- und Softwareexperten für den Maschinen- und Anlagenbau. Häufig werden diese drei Rollen zu Beginn der Einführung von ganzheitlicher Testautomatisierung mit einer einzigen Person repräsentiert werden müssen.

Die Methode der Wiederverwendung weist eine Bindung zu weiteren Methoden von GEATVI aus: Testtreiber, Komparator, Herstellerunabhängigkeit und virtueller Bediener. Die Bibliothek sollte die Lösungen der genannten Methoden auf der passenden Ebene zur Verfügung stellen. Damit ist das CAST Werkzeug bereits mit einer hohen Anzahl an Bibliothekselementen ausgestattet, bevor der Nutzer weitere unternehmensspezifische Anwendungsfelder hinzufügt.

Durch die Kopplung der Datenhaltung mit der GUI wird dem Benutzer der Zugang zu den Elementen der Bibliothek ermöglicht.

Nutzung und Erweiterung der Bibliothek

Neben Ablagemöglichkeit und Referenzmodell ist der dritte Aspekt einer erfolgreichen Methode der Wiederverwendung, das Vorgehen des Benutzers, zu definieren. Zum Vorgehen gehören die beiden Aspekte der Bibliotheksnutzung und -erweiterung. Für die Erweiterung ist es wichtig zu wissen, an welcher Stelle der Benutzer einen Testfall modularisiert und wie das Modul zu parametrisieren ist. Die Modularisierung der Testfälle muss als kontinuierlicher Prozess verstanden werden. Während ein neuer Testfall erstellt wird, kann immer wieder gefragt werden: Kann dieser gerade erstellte Teil in allgemeingültiger Form auch in anderen Testfällen eingesetzt werden? Dabei kann es sich um Funktionen aus der Basisebene bis hin zum Systemtest auf der Testfallebene handeln. Bspw. muss in einem Testfall ein Stringvergleich auf Zeichenbasis durchgeführt werden. Dieser Testschritt könnte als Modul aus dem Testfall gelöst werden und in der Bibliothek angeboten werden. Dabei würden die beiden Strings zum Vergleichen als Eingangsvariablen parametrisiert werden. Eine sinnvolle und zielgerichtete Modularisierung und Bereitstellung zur Wiederverwendung entsteht dann, wenn wiederkehrende Elemente aus einem oder mehreren Testfällen erkannt werden und diese der Bibliothek zugeführt werden. Künstliches Erzeugen von Elementen für die Bibliothek zeigt sich in der Praxis als nicht zielführend, da der Aufwand der Erstellung höher ist als der Nutzen. Zudem ist eine überladene Bibliothek nicht hilfreich im Sinne einer einfachen Benutzbarkeit, siehe übernächster Abschnitt im Text.

Beim Anlegen der Bibliothek ist darauf zu achten, dass die Elemente sprechende Namen bekommen. Diese können, je nach Unternehmen oder Abteilung, mit eigenen Abkürzungen oder Bezeichnern versehen sein. Die Nutzung von grafischen Symbolen bei der Anzeige von Bibliothekselementen, wie in Abbildung 45 rechts, erleichtert das schnelle Erkennen der Elemente. Neben Namen und Symbolbild ist auch eine Kurzbeschreibung des Elements sowie dessen Parametern vorzusehen.

Die Nutzung der Bibliothek erfolgt in zwei Schritten. Zum schnellen Arbeiten bei beim Testentwurf muss der Benutzer eingearbeitet sein, d. h. die Elemente der Bibliothek und deren Funktionsweise kennen. Danach kann er neue Testfälle nach dem „top-down“-Vorgehen (Norm VDI 3633 Blatt 1 2014, S. 23f) erstellen: Schrittweise versucht er Teile seines Testfalls aus vorhandenen Elementen der Bibliothek zu erstellen und beginnt dabei in der Ebene der umfangreichen Elemente (Ebene III in der Bibliothek). Der Vorteil dieses Vorgehens liegt darin, dass bei der Verwendung von umfangreichen Elementen direkt ein größerer Teil des Testfalls erstellt wird und das Vorgehen schneller zum gesamten Testfall/Testszenario führt als bei der Nutzung von Basisfunktionen. Weiterhin nutzt jedes Unternehmen eigene Vorbereitungs- und Nachbereitungsschritte sowie Log- und Prüfschritte für Testfälle. Diese sollten in der Ebene II -Bibliothek bereits integriert sein und müssen nicht zum wiederholten Male in jedem neuen Testfall erstellt werden. Sobald die Bibliothek auf der Testfallebene viele Elemente und eine ausgereifte Struktur besitzt, sind auch weniger spezialisierte Benutzer in der Lage, Testfälle durch reine Parametrierung und Verknüpfung der Elemente zu erstellen.

Weitere Quellen für den Testentwurf

Die Erstellung der Testfälle ist immer ein zweckgebundenes Vorgehen. Je nach Testart wird eine andere Testtechnik genutzt und nach deren Vorgehen die Testfälle erstellt. Jede Testtechnik verwendet daher auch andere Quellen zum Testentwurf, die Wichtigsten sind: Die Anforderungen aus Lasten- und Pflichtenheft, die Spezifikation einer Funktionalität oder Komponente, die Funktionsbeschreibung einer Hardware- oder Softwarekomponente. Zudem kann das Erfahrungswissen der Mitarbeiter entlang der Entwicklungsprozesse des Fertigungssystems

in Testfälle überführt werden. Insbesondere die Inbetriebnehmer und erfahrene Entwickler kennen Schwachstellen im eigenen System oder Prozess.

Neben dem Entwicklungsprozess liefert auch der Betrieb eines Fertigungssystems eine Quelle für den Testentwurf: Durch die Rückführung von Service- und Kundendienstfällen werden die im Feld auftretenden Störsituationen und Fehlerfälle so spezifiziert, dass sie in der Bibliothek zur Wiederverwendung zur Verfügung gestellt werden können. Jedes neues Fertigungssystem kann im Folgenden vorab auf Störsituationen und Fehlerfälle getestet werden.

Basierend auf dem fehlerbasiert erweiterten Simulationsmodell aus Kapitel 6.2.1 können weitere Testfälle für Schlechtfälle entworfen werden. Ziel ist es dabei, ein Szenario zu erstellen, welches eine mit einem Fehler beaufschlagte Komponente gebraucht. Wird z. B. die mechanische Verklemmung eines Aktors in das Simulationsmodell integriert, wird passend ein Testfall für die Bibliothek erstellt, der ein Programm ausführt, in welchem der Aktor betätigt wird. Neben dieser gezielten und offensichtlichen Integration eines Versagens in den Testfall macht auch eine randomisierte Integration Sinn. Randomisiert werden kann dabei sowohl der Zeitpunkt als auch die Häufigkeit des Auslösens des Komponentenversagens. Ziel dieser Testfälle ist es, nicht beachtete Interdependenzen innerhalb des Gesamtsystems zu identifizieren. Das randomisierte Setzen von Komponentenversagen macht zusätzlich Sinn unter Anwendung eines Dauerlaufs. Im Kontext der fehlerbasierten Modellerweiterung innerhalb der Simulation können weiterhin Testfälle zur Prüfung von Fehlerbehebungsstrategien durchgeführt werden. Für bekannte Fehlerszenarien liegen oft Strategien vor, wie das Fertigungssystem wieder in Betrieb gesetzt werden kann. Bspw. müssen in materialintensiven Systemen gewisse Pufferstellen geleert werden, um einen Neustart zu ermöglichen.

6.3 Zusammenfassung

Das Kapitel 6 bildet den Hauptbeitrag zum Entwurf der ganzheitlichen Testautomatisierung für automatisierte Fertigungssysteme in Form einer Lösung aus Methoden, Modellen und Werkzeugen.

Im Laufe des Kapitels wurden alle Anforderungen aus Kapitel 3, welche bisher nicht im Grobentwurf von Kapitel 5 behandelt wurden, betrachtet. Für alle Anforderungen wurde eine Lösung entworfen und ausspezifiziert. Zusammengefasst wurden dafür die Simulationsmodelle eines RT-HiL Systems fehlerbasiert erweitert, das CAST Werkzeug aus Kapitel 5 mit Funktionen und Methoden erweitert, eine Sprache zur Beschreibung von Testfällen entwickelt und das Vorgehen beim Testentwurf als methodisches Vorgehen beschrieben. Bei der Erarbeitung der Lösungen wurden mehrmals Techniken aus dem Bereich der Softwaretechnik und des Softwaretests entliehen und übertragen. Bspw. wurde die schlüsselwortgetriebene Testtechnik genutzt, um eine herstellerunabhängige Testbeschreibung zu entwerfen. Zur vollständigen Automatisierung der Testdurchführung wurde der virtuelle Bediener entworfen, um für den Test notwendige Bedienhandlungen abbilden zu können. Für den virtuellen Bediener entscheidend waren die Schnittstellen zum GUI sowie die Erstellung von Bedienabläufen, basierend auf den Bedienketten der Steuerungen. Auf die methodische Erstellung von Testfällen wurde in den letzten Abschnitten dieses Kapitels eingegangen. Als Grundlage wurde eine Sprache spezifiziert, die es mittels grafischer Elemente erlaubt, den Ablauf eines Tests abzubilden. Zuletzt wurde ein Vorgehen basierend auf Modularisierung und Wiederverwendung entworfen, um mittels einer stetig erweiterbaren Bibliothek anwendungsorientiert und schneller zu Testfällen und Testszenarien zu kommen.

Zusammenfassend zeigt Abbildung 46 den Gesamtaufbau und -umfang der Methodik zur ganzheitlichen Testautomatisierung. Das CAST Werkzeug in der Mitte der Abbildung bildet die Basis bei der Realisierung. Um das CAST Werkzeug herum werden die in den Kapiteln 5, 6 entworfenen Lösungen gezeigt. Im nachfolgenden Kapitel 7 wird auf die Implementierung eingegangen sowie eine Validierung vorgenommen.

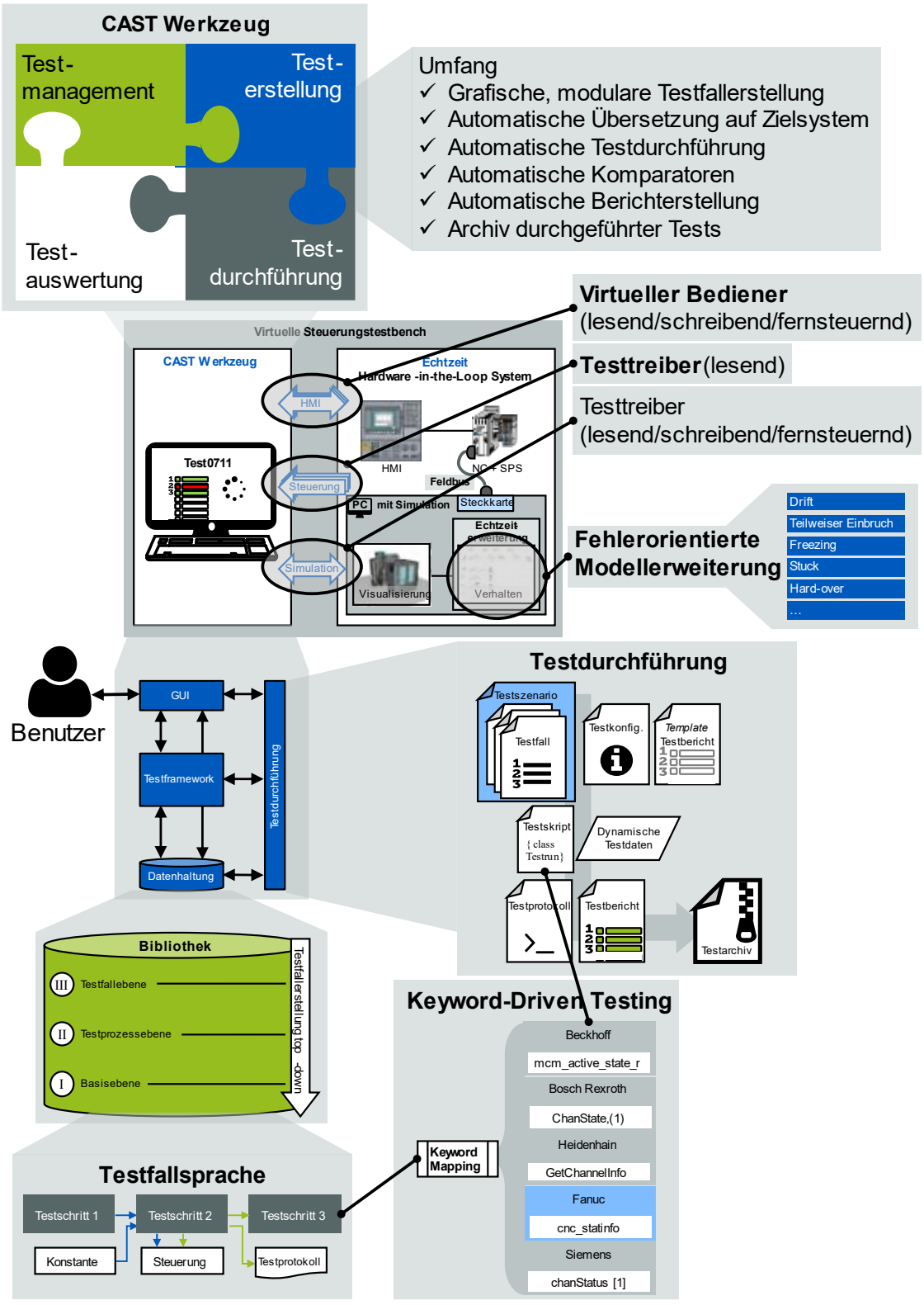


Abbildung 46 Gesamtarchitektur und Lösungen für die Methodik der ganzheitlichen Testautomatisierung

Mit GUI, Bibliothek und dem beschriebenen „top-down“-Vorgehen beim Testentwurf zielt die ganzheitliche Testautomatisierung darauf ab, Expertenwissen von Inbetriebnehmern und erfahrenen Testern für Neueinsteiger nutzbar zu machen. Meist tun sich Neueinsteiger aufgrund ungenauer Anforderungen oder Spezifikationen für die Softwareprogramme oder deren hoher Komplexität schwer beim Testentwurf (Tatsumi 1987). Durch die Nutzung einer frameworkbasierten Testautomatisierung mit einer Bibliothek an Testfällen und -szenarien sind neue Tester in der Lage, sich an Vorlagen von erfahrenen Testern zu orientieren und von deren Wissen zu profitieren.

7 Implementierung und praktische Validierung

Dieses Kapitel beschreibt die Implementierung und Validierung der ganzheitlichen Testautomatisierung. Im ersten Teil der Validierung wird das Kernstück der Methodik, das CAST Werkzeug, auf zwei verschiedene Weisen implementiert. Anschließend werden diese Realisierungen miteinander verglichen. Dadurch wird die Allgemeingültigkeit der entwickelten Testmethodik demonstriert. Der zweite Teil der Validierung vergleicht die ganzheitliche Testautomatisierung mit dem aktuell vorherrschenden manuellen Testprozess bei Werkzeugmaschinen- und Steuerungsherstellern.

Die Realisierung der beiden CAST Werkzeuge sieht einmal die vollständige Implementierung vor. Zum anderen, die Realisierung mit einem bestehenden CAST Werkzeugs, welches für den Einsatz an automatisierten Fertigungssystemen angepasst wird. In Abbildung 47 werden die vier realisierten Derivate der *virtuellen Steuerungstestbench*, Kopplung von CAST Werkzeug und RT-HIL System, gezeigt. Zwei der Derivate wurden auf Basis des im Rahmen dieser Arbeit implementierten CAST Werkzeugs, genannt *ViTau*, aufgebaut. Zwei weitere wurden auf Basis eines bestehenden CAST Werkzeugs realisiert, genannt *expecco* der Fa. *eXept*[12]. In den vier Derivaten wurden, neben den beiden CAST Werkzeugen, drei verschiedene Steuerungssysteme mit drei unterschiedlichen Schnittstellen eingebunden und drei verschiedene virtuelle Bediener realisiert.

Variante 1 wurde für einen Werkzeugmaschinenhersteller aufgebaut, um Testfälle aus der üblichen VIBN in den neuen Test der ganzheitlichen Testautomatisierung zu überführen. Die Steuerung ist eine Siemens Sinumerik 840D sl, die Kommunikationsstrecke ist eine Profibus-Verbindung. Als virtuelles Bearbeitungszentrum wurde eine HELLER CP8000 eingesetzt.

Variante 2 wurde für den direkten Vergleich zwischen ViTau und expecco aufgebaut. Das Setup des RT-HiL System entspricht deshalb genau der Variante 1.

[12] https://www.exept.de/produkte/expecco.html, zuletzt aufgerufen am 01.06.20

Variante 3 wurde für einen Steuerungshersteller aufgebaut, um die Messzyklen der Steuerung in einer Systemtestumgebung testen zu können. Als Steuerung kommt eine Beckhoff TwinCAT CNC zum Einsatz, die Kommunikationsstrecke ist eine EtherCAT Verbindung. Die virtuelle Werkzeugmaschine ist eine exeron HSC 600.

Variante 4 wurde für die Validierung der Anforderung der Herstellerunabhängigkeit aufgebaut. Die Steuerung ist eine Fanuc Model 31i-A, die Kommunikationsstrecke ist eine Profibus-Verbindung. Die virtuelle Werkzeugmaschine ist eine GROB G350. Variante 1 und 4 wurden mit ViTau automatisiert und beherrschen eine herstellerunabhängige Durchführung einiger Testfälle.

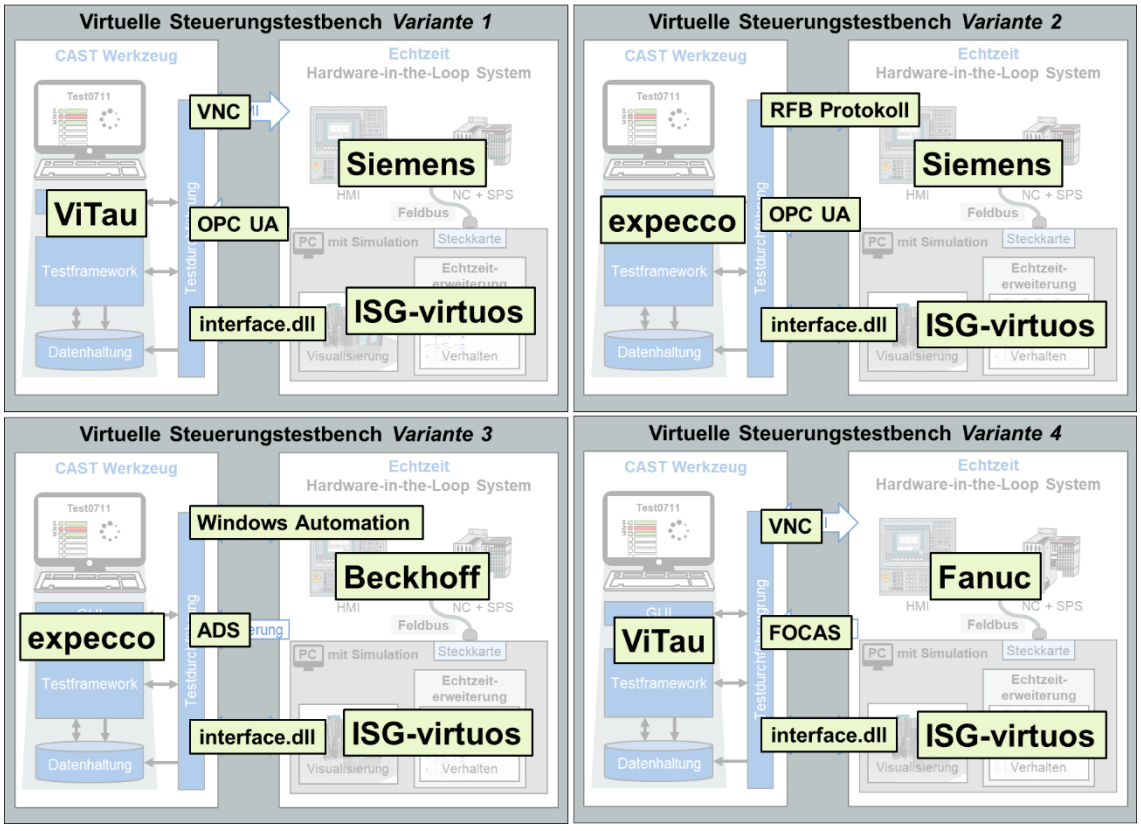


Abbildung 47 Visualisierung der im Rahmen dieser Arbeit erstellten Demonstratoren als Derivate der Virtuellen Steuerungstestbench

7.1 Entwicklung eines eigenen Testwerkzeugs

Dieses Kapitel beschreibt die Realisierung eines eigenen CAST Werkzeugs. Dieses bekommt den Namen „Virtuelle Testautomatisierung für CNC-Systeme“ (ViTau). Der Umfang der ViTau Implementierung umfasst GUI, Testframework und

Datenhaltung. Darin enthalten sind die entworfenen Konzepte der Herstellerunabhängigkeit, die Testtreiber, die Komparatoren, die Sprache zur Erstellung der Testfälle und die Bibliothek zur Wiederverwendung realisiert werden.

Zunächst wurden geeignete Plattformen ausgewählt, um die Realisierung für GUI, Testframework und Datenhaltung umzusetzen. Höchste Priorität für die beiden Derivate mit ViTau haben eine schnelle Austauschbarkeit von Modulen in Testframework und Datenhaltung sowie eine von Menschen lesbare Beschreibung der Module. Für das Framework wurde daher die Entscheidung getroffen, es mit einer Skriptsprache umzusetzen. Da die Skriptsprache *Python* eine hohe Verbreitung besitzt und viele Bibliotheken zur Ergänzung bereitgestellt werden, wurde diese Sprache als Realisierungsplattform gewählt. Für die Datenhaltung wurde das weit verbreitet Format Extensible Markup Language (XML) gewählt und z. B. kein Datenbanksystem. Abbildung 48 zeigt die Benutzeroberfläche von ViTau. Als Realisierungsplattform hierfür wurde das C# *Windows Presentation Framework* gewählt, um eine optimale Unterstützung auf Windowssystemen zu gewährleisten.

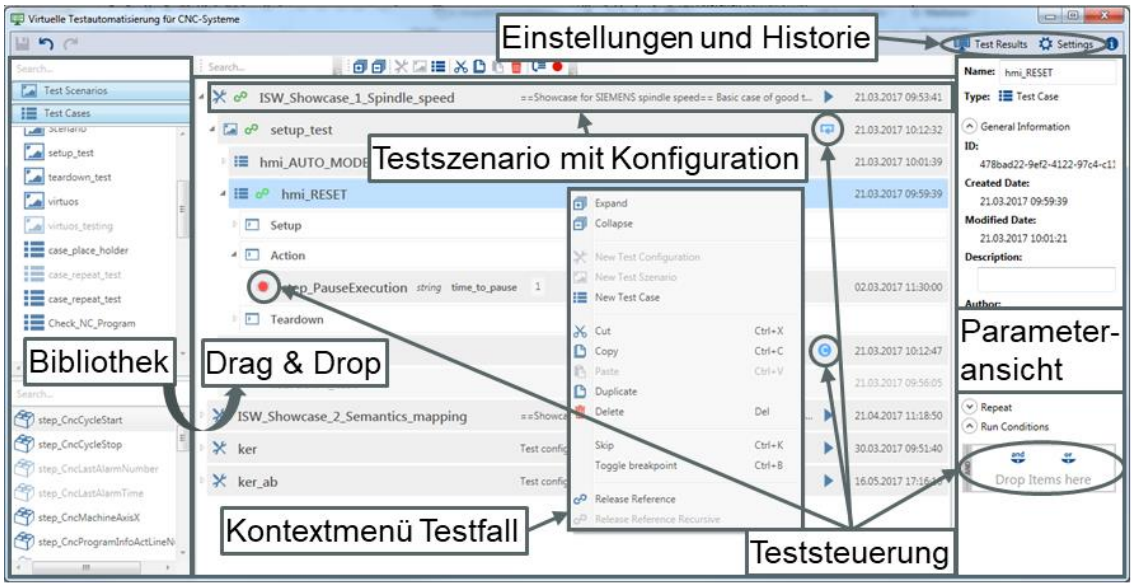


Abbildung 48 Die GUI des entwickelten Testwerkzeugs ViTau mit Erläuterungen zu einzelnen Funktionen

In Abbildung 49 ist der fertige Aufbau des Demonstrators der Virtuellen Steuerungstestbench in der Variante 2 zu sehen.

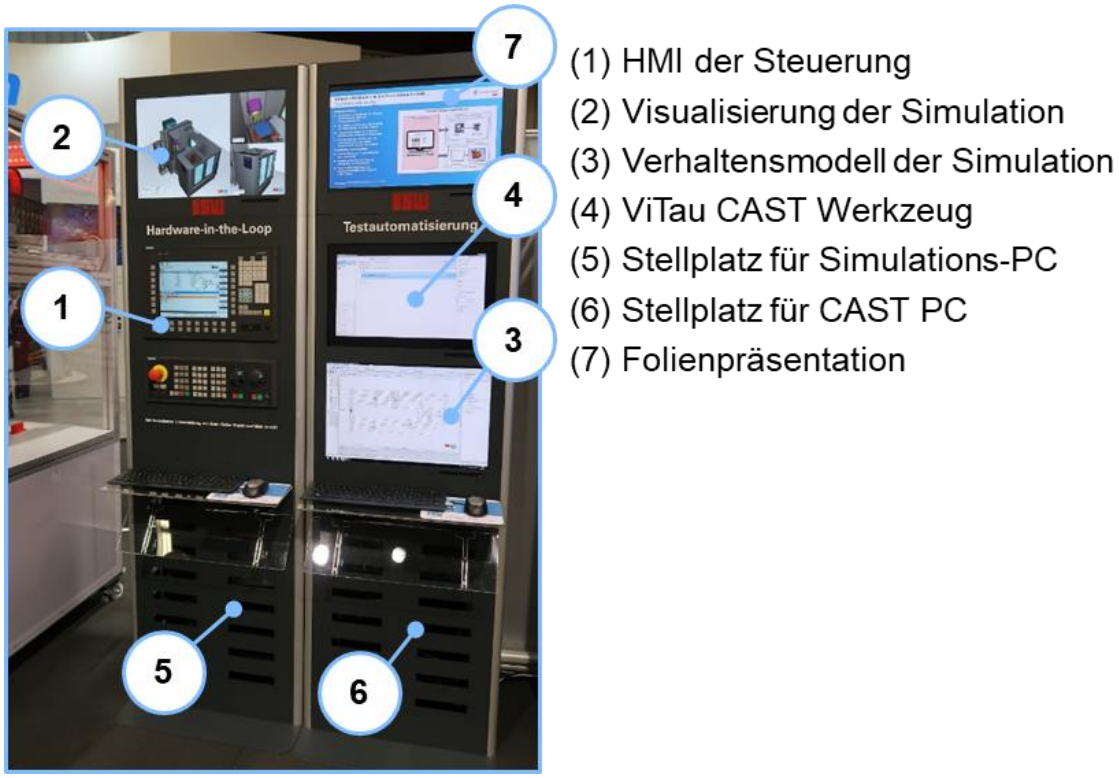


Abbildung 49 Aufnahmen des Demonstrators zur ganzheitlichen Testautomatisierung unter Nutzung eines RT-HiL Systems

7.2 Realisierung mit einem bestehenden Testwerkzeug

In diesem Kapitel wird die Realisierung des CAST Werkzeugs, basierend auf dem Testwerkzeug expecco der Fa. eXept, vorgestellt. Es wird zunächst darauf eingegangen, warum die Wahl bei bestehenden CAST Werkzeugen aus anderen Anwendungsdomänen auf expecco fällt. Anschließend wird darauf eingegangen, welche Implementierungen notwendig sind, um expecco für automatisierte Fertigungssysteme einsetzen zu können.

Aus der großen Menge an verfügbaren CAST Werkzeugen[13] wird mittels einer Liste an K.O.-Kriterien eine Vorauswahl getroffen. Bei Erfüllung einer der folgenden Kriterien wird das Werkzeug nicht weiter betrachtet:

- Es handelt sich um ein Testwerkzeug für Hochsprachenapplikationen.
- Es wird keine Interaktion mit einer GUI ermöglicht (nicht Web-GUI).
- Es können keine weiteren Schnittstellen eingebunden werden.
- Der Testbericht kann nicht automatisiert erzeugt werden.
- Das Werkzeug ist veraltet oder wird nicht mehr gewartet.
- Der Testentwurf ist nicht grafisch möglich.

Nach Sichtung von 42 CAST Werkzeugen kommen die folgenden acht Werkzeuge für eine Umsetzung in Betracht:

1. EggPlant Functional[14] von Eggplant
2. *expecco*[15] von eXept
3. MESSINA[16] von expleo
4. Ranorex Studio[17] von Ranorex
5. *Squish*[18] von froglogic
6. *TestComplete*[19] von SmartBear
7. TPT[20] von PikeTec
8. TTworkbench[21] von Spirent

Die Entscheidung für das Werkzeug expecco wurde aus zwei Gründen gefällt. Erstens ist es das einzige Werkzeug, das eine grafische Erstellung der Testfälle mittels Blockschaltbildern erlaubt und gleichzeitig die verlangten Abläufe und

---

13 Genutzte Auflistungen, zuletzt aufgerufen am 03.06.20
https://www.testtoolreview.de/de/testing-tools/
https://www.testing-board.com/testautomatisierung-tools/
https://de.wikipedia.org/wiki/Liste_von_Software_für_automatisierte_Softwaretests

14 https://www.eggplantsoftware.com, zuletzt aufgerufen am 03.06.20

15 https://www.exept.de/produkte/expecco.html, zuletzt aufgerufen am 03.06.20

16 https://www.expleo-germany.com/produkte/messina/, zuletzt aufgerufen am 03.06.20

17 https://www.ranorex.com, zuletzt aufgerufen am 03.06.20

18 https://www.froglogic.com/squish/, zuletzt aufgerufen am 03.06.20

19 https://smartbear.de/product/testcomplete/overview/, zuletzt aufgerufen am 03.06.20

20 https://piketec.com/tpt/, zuletzt aufgerufen am 03.06.20

21 https://www.spirent.com/products/ttworkbench, zuletzt aufgerufen am 03.06.20

Verzweigungen in der Sprache abbilden kann. Zusätzlich ist die textbasierte Programmierung möglich, diese wurde als Wunschanforderung definiert. Der zweite Grund ist die Offenheit gegenüber Erweiterungen des Werkzeugs mittels eigener Plugins. Diese Möglichkeit schien bei keinem der anderen Werkzeuge so einfach zu funktionieren und ist die wichtigste Grundvoraussetzung zur Implementierung der ganzheitlichen Testautomatisierung.

Aus der getroffenen Auswahl ergeben sich folgende offene Arbeitspunkt zur Realisierung mit expecco:

- Testtreiberintegration für die Schnittstellen zum RT-HiL System inklusive des virtuellen Bedieners
- Integration der Herstellerunabhängigkeit der Testschritte
- Erstellung einer Bibliothek für das Anwendungsgebiet der Fertigungssysteme inklusive des virtuellen Bedieners

Die nachfolgende Abbildung zeigt die realisierten Bibliotheken für die ViST Variante 2 (links) und Variante 3 (rechts). Unter den Screenshots der Bibliothek ist angegeben, wie viele Testschritte in den jeweiligen Bibliotheken enthalten sind.

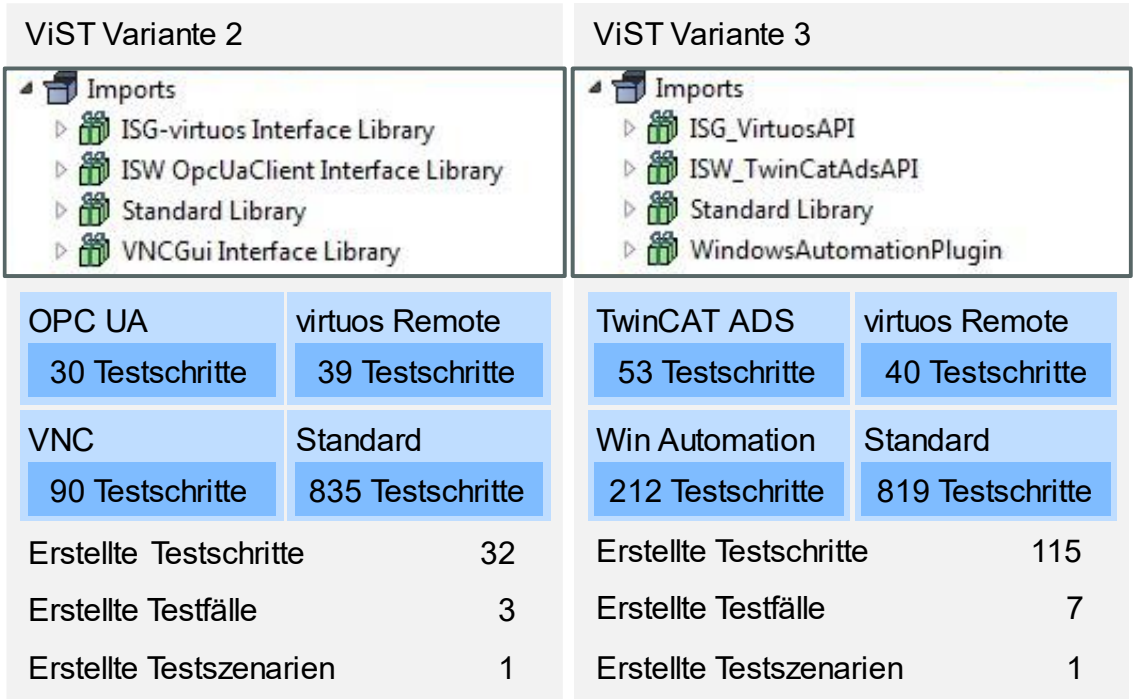


Abbildung 50 Übersicht zu den erstellten Bibliotheken für die Automatisierung mittels expecco

7.3 Validierung der ganzheitlichen Testautomatisierung und Diskussion zu Vorteilen und Grenzen der Methodik

Wie in der Implementierung der CAST Werkzeuge gezeigt wurde, ist die Methodik in dieser Arbeit unabhängig von der technologischen Realisierung entworfen worden. Mittels der vier vorgestellten Demonstratoren konnte gezeigten werden, dass es die ganzheitliche Testautomatisierung allgemein ermöglicht (in Klammern sind die berücksichtigten Anforderungen genannt):

- Den Zugriff auf Steuerung und Simulation zu automatisieren (/A1/, /A2/, /A3/),
- die HMI eines Steuerungssystems vollautomatisch zu bedienen (/A1/, /A2/),
- die Nebenzeiten des Tests für die Vorbereitung und Nachbereiten des RT-HiL (Laden von Programmen, Setzen eines Zustands der Simulation, etc.) zu automatisieren (/A1/, /A8/),
- die Auswertung und Aufbereitung der Daten aus der Testdurchführung für Berichtszwecke zu automatisieren (/A8/, /A9/),
- die Erstellung von Testfällen und Testszenarien mittels einer grafischen Modellierungssprache (/A5/),
- die Erstellung von Testfällen unabhängig vom Zielsystem RT-HiL zu formulieren und zur Ausführungszeit zu übersetzen (/A4/),
- die Erstellung von Testfällen und Testszenarien mit dem Konzept der Wiederverwendung zu unterstützen (/A7/),
- durch fehlerbasierte Erweiterung der Simulationsmodelle im RT-HiL die Anzahl an Testszenarien je Maschine zu erhöhen (/A6/).

Für die praktische Validierung der ganzheitlichen Testautomatisierung werden zwei Szenarien gewählt: Zum einen soll die Anwendung des CAST Werkzeugs, welches den größten Teil der Methodik abbildet, untersucht werden. Zum anderen soll eine Validierung bei der praktischen Anwendung in einem Unternehmen betrachtet werden. Zunächst werden die CAST Werkzeuge mit dem bisher manuellen Vorgehen im Testprozess verglichen.

7.3.1 Praktische Validierungen durch Anwendung auf ein Testszenario

Im weiteren Verlauf des Abschnitts wird die praktische Validierung durch Umsetzung und Durchführung eines Testszenarios an zwei Demonstratorvarianten (ViST Variante 1 und Variante 2) beschrieben und mit dem manuellen Testprozess verglichen.

Es wird ein typisches Testszenario an einer Werkzeugmaschine gewählt, das unabhängig von einem spezifischen Hersteller oder Maschinentyp funktioniert. Als Testszenario wird das Auslösen einer Notaus-Situation an einer Werkzeugmaschine gewählt. Abbildung 51 zeigt das in einem Ablaufdiagramm definierte Testszenario. Die entscheidende Funktionalität bei diesem Testfall ist das Wiederanfahren der Maschine nach Löschen des Not-Aus Zustands. Je nach Maschine und Anwendungsfall sind hier unterschiedliche Verhalten definiert (Pritschow 2006, S. 253). Der Zeitpunkt beim Auslösen eines Not-Aus Testszenarios kann ebenfalls entscheidend sein, denn bspw. bedarf das Auslösen eines Nothalts in einem Bearbeitungszentrum während eines Werkzeugwechsels einer anderen Strategie zur Wiederherstellung des Zustands „betriebsbereit“ als der Not-Aus während einer Werkstückbearbeitung. Zum Vergleich der CAST Werkzeuge sind Fehlerszenarien nicht relevant, deshalb wird der Neustart des NC-Programms verlangt.

Der Testfall „Emergency stop“ ist so gewählt, dass bei der Testdurchführung alle Schnittstellen zwischen CAST Werkzeug und RT-HiL angesprochen werden müssen. Nicht Teil des Testfalls aber Teil des Testszenarios ist es, das Testprotokoll auszuwerten und darauf basierend einen Testbericht zu erstellen. Beide CAST Werkzeuge machen dies automatisch. Beim manuellen Test fand keine explizite Protokollierung der Testdurchführung statt.

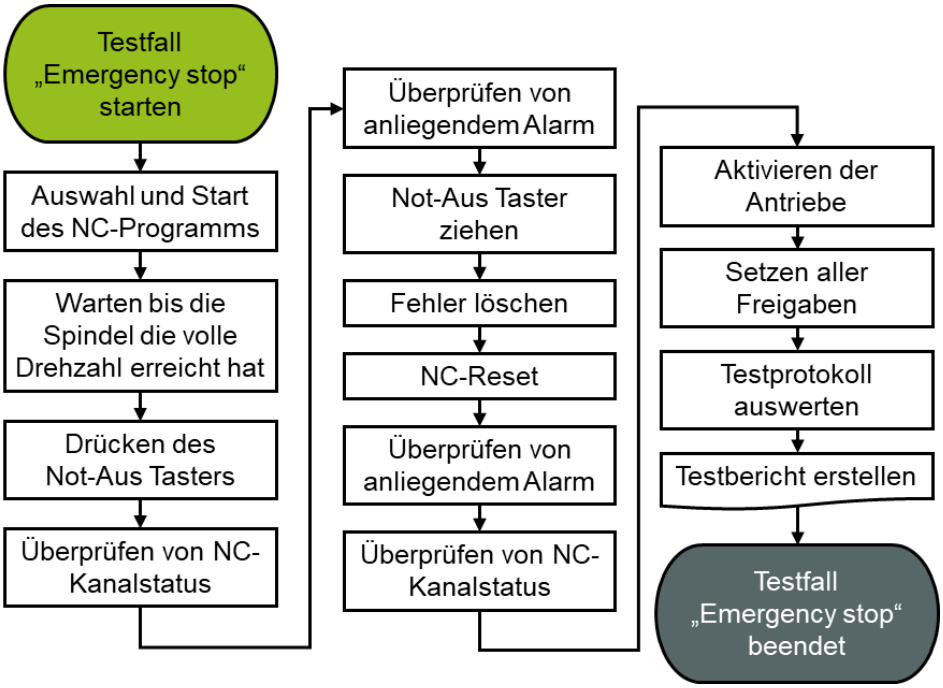


Abbildung 51 Ablaufdiagramm für den Testfall "Emergency stop"

Das für die Ausführung des Testszenarios betrachtete System ist ein RT-HiL Aufbau mit einem virtuellen Bearbeitungszentrum *CP8000* der Fa. *Gebr. Heller GmbH &Co KG,* welches freundlicherweise während der Erstellung dieser Arbeit zur Verfügung stand. Abbildung 52 zeigt die schematische Darstellung der beiden Aufbauten.

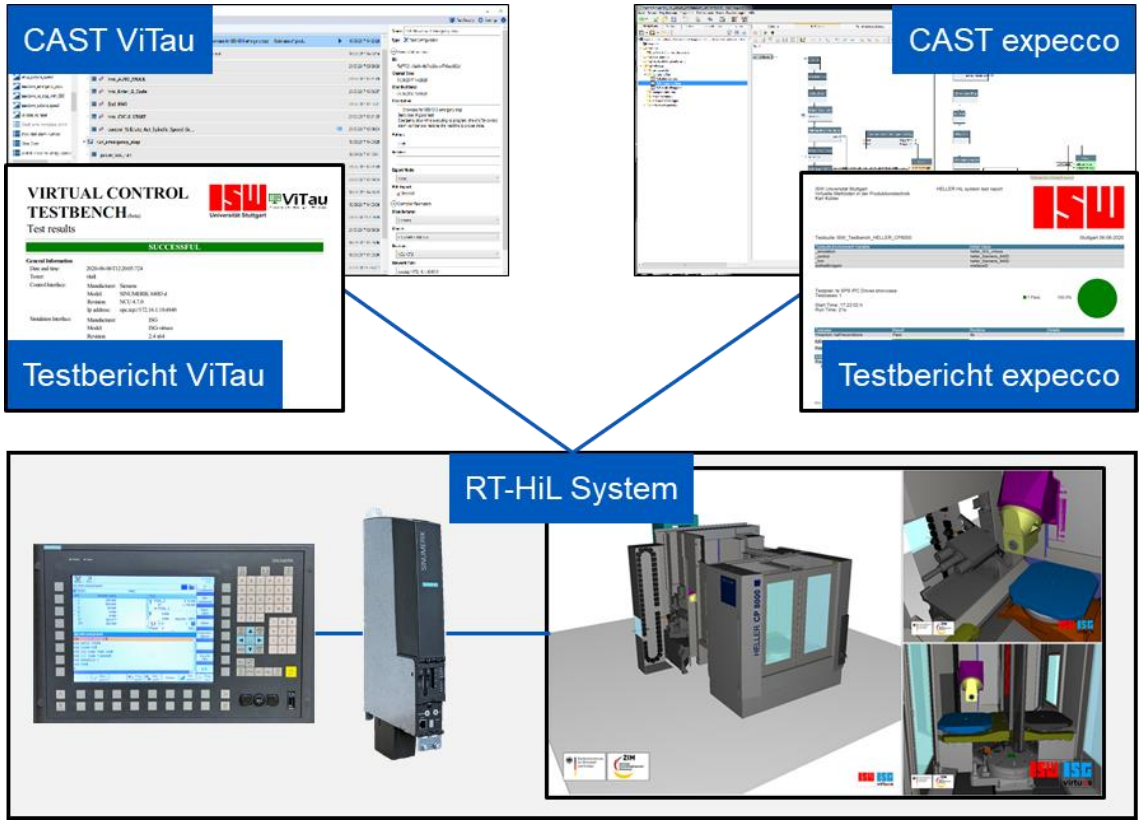


Abbildung 52 Aufbau der Variante1 (links) und Variante 2 (rechts) des Demonstrators zum direkten Vergleich der CAST Implementierungen; oben: Screenshots der erstellten Testszenarien sowie die von den CAST Werkzeugen generierten Testberichte; unten: RT-HiL System aus HMI, Steuerung und virtuellem Bearbeitungszentrum.

Abbildung 53 zeigt, neben den jeweils im CAST Werkzeug spezifizierten Testfällen, auch die zu bedienenden Elemente auf der virtuellen Steuertafel und die zu erwartende Fehlermeldung auf der HMI der Steuerung, siehe blaue Kreise.

Abbildung 53 Testszenario "Emergency stop"; oben: die Reaktion am RT-HiL System, NC HMI zeigt den Fehler "Not-Halt"; mittig: die virtuelle Steuertafel zeigt in den blauen Kreisen an, dass der Nothalt ausgelöst ist, die Freigaben zurückgesetzt wurden und die Antriebe stromlos geschalten wurden; unten: das modellierte Testszenario in den beiden Werkzeugen ViTau und expecco.

Für die praktische Validierung wurde das Testszenario „Emergency stop“ je zehn Mal mit der ViST Variante 1 (CAST Werkzeug ViTau), der Variante 2 (CAST Werkzeug expecco) und manuell ausgeführt. Für den Vergleich wurden die Zeitdauern für die Durchführung und Auswertung je Variante erfasst. Bei den CAST Werkzeugen wurden die Zeiten aus dem Testprotokoll ermittelt. Bei der manuellen Durchführung wurde die Zeit ebenfalls manuell gestoppt. Der Tester für die manuelle Testdurchführung ist geübt im Umgang mit dem System.

Aus den erfassten Zeiten wurden, zum einen, die Mittelwerte und, zum anderen, die Spannweite errechnet, siehe Abbildung 54.

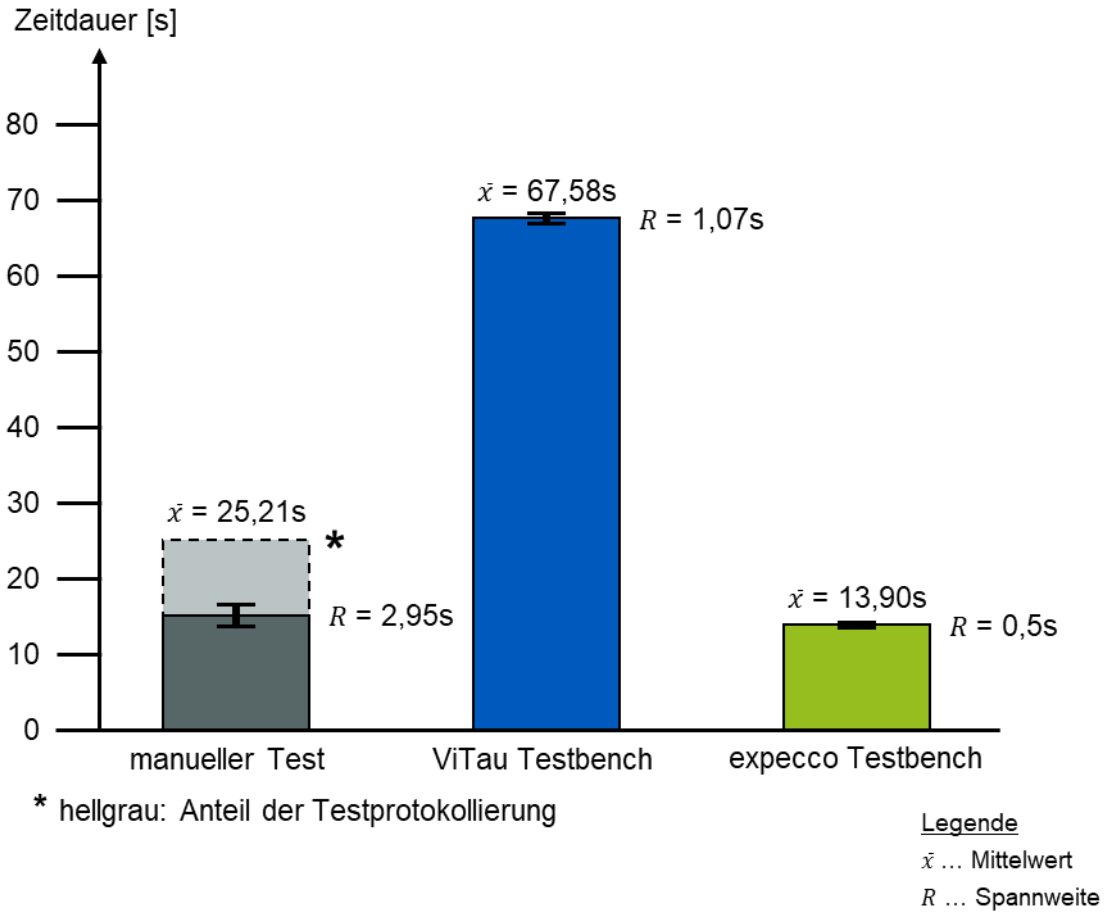


Abbildung 54 Vergleich der Zeitdauer bei der Durchführung und Auswertung im Testszenario „Emergency stop“ an den drei Testkonfigurationen

Bei der Diskussion der Ergebnisse ist zu beachten, dass beim manuellen Test die Protokollierung der Testdurchführung im Rahmen eines vorgefertigten Testberichts stattfand. Der Testbericht beinhaltet die Schritte der Testdurchführung und wird mit „i.O.“ oder „n.i.O.“ kommentiert. Bei einem auftretenden Fehlverhalten sind die beobachteten Werte und Anomalien in einem dafür vorgesehenen Feld zu notieren. Das manuelle Ausfüllen des Testberichts geschieht, ohne auftretende Fehler, nach Ende der Testdurchführung, daher konnte die Zeit für die Erstellung des Testberichts separat, mit einer Zeitdauer von ca. 10 s, ermittelt werden. Die Protokollierung aller Schritte und Signale geschieht bei den CAST

Werkzeugen automatisch. Die Generierung des Testberichts mittels ViTau dauerte ca. 0,5 s und erzeugt einen elfseitigen Bericht. Mittels expecco wurde ein zwölfseitiger Bericht erstellt, innerhalb von 2 s.

Folgende Beobachtungen können im Hinblick auf den Vergleich der verschiedenen Testdurchführungen und -auswertungen gemacht werden:

- Das ausgereifte CAST Werkzeug (expecco) ist am effizientesten bei der Testdurchführung und -auswertung; der Zeitvorteil gegenüber dem manuellen Test beträgt ca. 45 %, ist jedoch immer davon abhängig, wie viele Bedienhandlungen bei der Testdurchführung stattfinden.
- Automatisierung heißt nicht direkt eine schnellere Durchführung, siehe Vergleich „manueller Test“ und „ViTau Testbench“; das Werkzeug muss optimiert sein.
- Die zeitliche Streuung bei der Testdurchführung ist bei der manuellen Prüfung am höchsten: Sie ist mehr als 2,5-fach, bzw. fast 6-fach, so hoch wie bei der Testautomatisierung; dabei sind tagesform- und personenabhängige Schwankungen noch nicht berücksichtigt.
- Der Aufwand für Protokollierung und Testberichterstellung beim manuellen Test hat einen überproportionalen Anteil am Zeitaufwand; im gemessenen Vergleich ist die Erstellung des Testberichts beim manuellen Vorgehen 5- bis 20-mal so zeitintensiv.
- Das Durchführen gleicher manueller Tätigkeiten beim Testen führt zu Nachlässigkeiten und mindert somit das Testergebnis und die Vergleichbarkeit.

7.3.2 Einsatz von ganzheitlicher Testautomatisierung zum Systemtest bei Messzyklen einer NC-basierten Steuerung

In diesem Kapitel wird ein Anwendungsszenario der ganzheitlichen Testautomatisierung bei einem Steuerungshersteller beschrieben. Der Steuerungshersteller bietet einen NC-Kern als kommerzielles Softwarepaket mit dazugehörigen Mess-, Kalibrierungs- und Bearbeitungszyklen an. In diesem Anwendungsszenario geht es darum, wie mittels der ganzheitlichen Testautomatisierung ein Nutzen

für den Systemtest der Mess- und Kalibrierungszyklen erreicht wird. Der Aufbau und die Anwendung der ganzheitlichen Testautomatisierung fand intern durch den Steuerungshersteller selbst statt. Vorbild für den Einsatz war der Demonstrator in Variante 3 aus der vorliegenden Arbeit. Dieses Kapitel berichtet über die Ausgangssituation, das Vorgehen und die Ergebnisse der ganzheitlichen Testautomatisierung. Als Quelle dienen geführte Interviews und schriftliche Dokumentationen sowie die Testergebnisse. Als CAST Werkzeug wurde das auf expecco basierende Werkzeug ISG-dirigent eingesetzt.

Ausgangssituation

Der NC-Kern inklusive Zyklen wird neben Werkzeugmaschinen auch vermehrt zur Steuerung von Robotern in der Handhabung und Freiformflächenbearbeitung eingesetzt. Dabei kann nativer G-Code nach DIN 66025 für die Programmierung verwendet werden und mehrere Achsen im Verbund bewegt werden. Zum Erhöhen der Fertigungsgenauigkeit von Werkstücken in der NC-basierten Bearbeitung stehen für den Anwender des NC-Kerns Mess- und Kalibrierungszyklen zur Verfügung. Mit diesen Zyklen werden Messmittel kalibriert, Werkzeuge und Werkstücke eingemessen sowie die Nullpunktverschiebung des Werkstückkoordinatensystems ermittelt. Die Messzyklen umfassen manuell, halbautomatisch und vollautomatisch durchzuführende Abläufe. (ISG Industrielle Steuerungstechnik GmbH 2021)

Bspw. wird mit dem Zyklus „SysMeasWcs8“ die Nullpunktverschiebung des Werkstückkoordinatensystems anhand des Mittelpunkts eines Kreises (bspw. in einer Bohrung) anhand von vier Messungen vollautomatisch eingemessen. Die Syntax des Befehls und dessen Parameter lauten (ISG Industrielle Steuerungstechnik GmbH 2021, S. 109f):

„L CYCLE [NAME= SysMeasWcs8.ecy @P2=.. @P7=.. @P9=.. @P10=.. @P13=.. @P33=..]“

Zyklusparameter	Beschreibung
@P2	Bohrungsdurchmesser
@P7	Nullpunktverschiebung, die beschrieben werden soll (WCS) • G54 bis G59 • Eingabe ohne Adresse „G“
@P9 (optional)	Verschieben des Messpunktes entlang der X-Achse
@P10 (optional)	Verschieben des Messpunktes entlang der Y-Achse
@P13 (optional)	Sicherheitsabstand zum Vorpositionieren (Wird @P13 nicht angegeben, wird die gesamte Messstrecke im Messvorschub gefahren)
@P33	Überfahrbereich • Voreingestellt 5mm

Abbildung 55 Überblick und Beschreibung der Parameter für den Messzyklus (ISG Industrielle Steuerungstechnik GmbH 2021, S. 110)

Problemstellung für den Test

In der Entwicklung des Steuerungsherstellers steht eine reale Testumgebung zur Verfügung, bestehend aus einem KUKA KR90 Sechsachs-Roboter, einer KUKA KRC4 Steuerung, mit dem integrierten NC-Kern, Messtastern und Beispielwerkstücken. An der realen Testumgebung werden neue Mess- und Kalibrierungszyklen entwickelt und bestehende Zyklen getestet. Die Vielfalt an Konfigurationen, in welchen der NC-Kern bei Kunden eingesetzt wird, kann mit der einen realen Testumgebung längst nicht mehr abgedeckt werden. Zudem steht die reale Testumgebung nicht exklusiv dem Test zur Verfügung, weswegen es zu zeitlichen Engpässen kommt. Bei wichtigen Entwicklungen werden heute als Alternative die abschließenden Tests direkt beim Kunden vor Ort an einer realen Testumgebung durchgeführt. Vor den abschließenden Tests wird nur vom Entwickler modulweise getestet.

Für das Messen und Kalibrieren stehen aktuell 56 Zyklen im NC-Kern zur Verfügung. Nach jeder Änderung an einem Zyklus oder der Änderung eines abhängigen Moduls im NC-Kern selbst (Module „Messfunktion“, „Koordinatensysteme“, „Antriebssysteme“, „SPS-Schnittstelle“) sollten die Zyklen getestet werden. Der aktuelle Aufwand von ca. einem Arbeitstag zum Test aller Zyklen führt dazu, dass

erst kurz vor Freigabe an die Kunden alle Zyklen getestet werden. Der Aufwand fällt nur für den Systemtest an, Entwickler- und Modultests finden bereits im Zuge der Entwicklung statt. Dieses Vorgehen führt zu Zeitdruck und Mehrarbeit, wenn Fehler so kurz vor Freigabe entdeckt werden. Aufgrund des gestiegenen Aufwands einer 100 %igen Testung aller Zyklen vor der Freigabe und der nicht ausreichenden realen Testumgebung hat sich der Steuerungshersteller für den Einsatz der ganzheitlichen Testautomatisierung für den Bereich der Zyklenentwicklung als Pilotprojekt entschieden.

Vorgehen

Für die Realisierung der ganzheitlichen Testautomatisierung in einem Systemtestszenario wurde vom Steuerungshersteller eine Virtuelle Steuerungstestbench von Grund auf neu aufgebaut: Ein RT-HiL System mit einer Simulation eines Stäubli TX200 Roboters und der Kopplung mit der Stäubli CS8 Steuerung und einer Beckhoff TwinCAT CNC mittels Bussystem (CANopen over EtherCAT). Für die Modellierung der Simulation der Roboterapplikation war es entscheidend, das Verhalten der Messmittel nachzubilden. Dazu wurde insbesondere eine Kollisionserkennung zum Erkennen des Kontakts zwischen taktilem Messmittel und Werkstück realisiert. Der Screenshot in Abbildung 56 zeigt die 3D-Visualisierung der Simulation und die Ausführung des Testszenarios für den Messzyklus SysMeasWcs8.

In der automatisierten Vorbereitung des Testszenarios werden das benötigte Werkstück und das benötigte Werkzeug (Messmittel) in die Simulation geladen. In die Steuerung wird mittels GUI Automation das benötigte Programm geladen, welches den Zyklusaufruf enthält. Zuletzt muss, wie bei Mess- und Kalibrierungszyklen üblich, der Tool Center Point positioniert werden, bevor der Zyklus gestartet werden kann. Die Positionierung kann automatisch, per NC-Programm, oder manuell, per Handbetrieb, erfolgen und wird ebenfalls vom automatisierten Testfall übernommen.

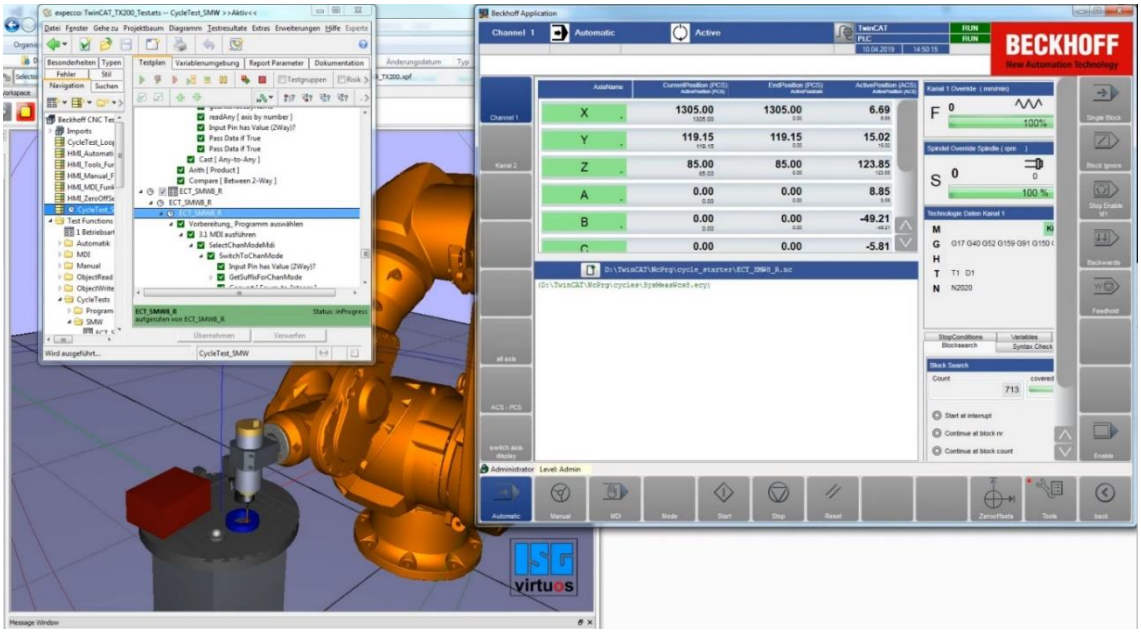


Abbildung 56 Screenshot bei laufender Testdurchführung an der virtuellen Testumgebung zum Test von Messzyklus SysMeasWcs8 (mit freundlicher Genehmigung der ISG Steuerungstechnik GmbH)

Neben dem Gutfall, dass der Zyklus entsprechend der Spezifikation ausgeführt wird, werden auch bekannte Fehlerfälle getestet. Dazu zählen die beiden Fehlerfälle „P-ERR-13413“ und „P-ERR-13414“ laut Dokumentation:

- „Wird während der Messung kein Messsignal ausgelöst, stoppt die Maschine mit der Fehlermeldung: „Es wurde kein Werkstück gefunden, bitte überprüfen Sie die Messstrecke.“.“
- „Wird während der Positionierung eine Messung erkannt, stoppt die Maschine mit einer Fehlermeldung: „Es gab eine Kollision während der Positionierung, Fahrbereich überprüfen.“.“

Ergebnisse

Der Systemtest als Testart für die Zyklen ist, neben dem Entwicklertest, wichtig, da die Abhängigkeiten im späteren kundenspezifischen Zielsystem mitgetestet werden. Der Test geschieht unter den gleichen Bedingungen, wie später die Software, die beim Kunden im Einsatz ist. Durch die Anwendung der ganzheitlichen Testautomatisierung beim Systemtest der Zyklen werden folgende Vorteile erzielt:

- Die komplette Testdurchführung aller Zyklen-Testfälle nach jeder Änderung am Code der Steuerung (Regressionstest) geschieht zuverlässig und gleichzeitig ohne Mehraufwand.
- Der zeitliche Aufwand bei automatisierter Durchführung reduziert sich von bisher ca. 2 h je Zyklus auf ca. 5 min zum Starten des Tests und die Prüfung des generierten Protokolls.
- Der konkrete Ablauf beim Testen, z. B. das manuelle Positionieren des Messmittels, erfolgt auf immer gleiche Art und Weise.
- Die Protokollierung der Testdurchführungen geschieht immer und einheitlich.
- Die Testdurchführungen können jederzeit von Nicht-Experten gestartet werden.
- Kostenreduktion durch ausfallende Reisekosten und Einsparung einer realen Testumgebung.
- Testfälle werden ohne Aufsicht durchgeführt, Beschädigungen sind am RT-HiL System ausgeschlossen.
- Bei engem Zeitplan vor der Freigabe werden die Nacht- und Wochenendzeiten zur Testdurchführung genutzt.

Der Einsatz der ganzheitlichen Testautomatisierung in dem vorgestellten Pilotprojekt wird als sehr erfolgreich gewertet. Der Steuerungshersteller prüft aktuell, den Einsatz zu erweitern. Der erweiterte Einsatz von ganzheitlicher Testautomatisierung auf Bearbeitungszyklen, Kundensystemen und Testarten bietet weiteres Verbesserungspotential in der Qualitätssicherung, wie bspw. ein einheitliches Vorgehen beim Test in der gesamten Entwicklung, Wiederverwendung von erstellten Testfällen und zertifizierte Abnahmeprotokolle.

7.3.3 Vorteile und Grenzen einer ganzheitlichen Testautomatisierung

Basierend auf den Ergebnissen der Validierung wird in Abbildung 57 eine Übersicht zum Vergleich zwischen dem aktuellen, manuellen Testprozess und der ganzheitlichen Testautomatisierung gezeigt. Im weiteren Verlauf dieses Abschnitts werden Vorteile und Grenzen der neuen Testmethodik diskutiert.

Kriterium \ Ansatz	Manueller Testprozess	Ganzheitliche Testautomatisierung
Manuelle Tätigkeiten	Erstellen, Durchführen und Bewerten der Tests	Erstellen der Tests
Aufwand f. Testentwurf	Manuell und immer wieder	Durch Modularisierung und Wiederverwendung
Aufwand f. Testdurchführung	Manuell	Durch Automatisierung
Aufwand f. Testauswertung	Manuell	Durch automatische Durchführung
Aufwand f. Hilfsmittel (Tracing, Forcing)	Manuell	Durch automatische Anwendung
Anzahl nicht getesteter Szenarien	Weniger Tests pro Zeiteinheit	Die Testabdeckung erhöht sich
Aufwand f. reale Inbetriebnahme	Potenziell mehr unentdeckte Fehler	Durch mehr Test-szenarien vorab
Zusatzkosten	Simulationswerkzeug, Schulung, Personaleinsatz	Simulationswerkzeug, Testwerkzeug, Schulung, Personaleinsatz

Legende
höher/mehr
geringer/weniger

Abbildung 57 Vergleich des manuellen Testprozesses mit ganzheitlicher Testautomatisierung anhand Kriterien entlang der Einführung, Verwendung und Qualitätsverbesserung. Der Text je Kriterium liefert die Begründung für die Bewertung.

Durch den Aspekt der Automatisierung von Testentwurf, Testdurchführung und Testabschluss wird Zeit in der Entwicklung, bzw. bei der Abnahme, frei, welche auf die Behebung der Fehler und Optimierung des Fertigungssystems verwendet werden kann. Pro Woche erhöht sich im Dauerlauf zudem die Testzeit, da über die Nächte und das Wochenende Testszenarien durchgeführt werden können, ohne Personal abzustellen. Sporadische Fehler werden nur so, im Dauerlauf, gefunden.

Das exakte und schnelle Logging während der Testdurchführung ist bei der Fehlersuche und Behebung hilfreich. Durch aufwandsarme erneute Testdurchführung nach einer Fehlerbehebung wird das Regressionstesten befördert und Änderungen sind ebenfalls verifiziert.

Durch den Einsatz einer RT-HiL Simulationsumgebung für das Fertigungssystem ist es möglich, einen Teil der Testdurchführung in die Echtzeitumgebung zu verlagern und dort auszuführen. De facto wird dieser Vorteil bei der Einführung der fehlerbasierten Erweiterung der Simulationsmodelle genutzt: Die Fehlerszenarien werden als Teil der Simulation modelliert und vom Testfall angestoßen. Damit wird eine höhere Testabdeckung der Automatisierungssoftware erreicht, da Fehlerszenarien umfangreich und mit zeitlich korrektem Verhalten getestet werden. Als Resultat werden die Steuerungsprogramme robuster.

Am Ende eines Projekts oder Auftrags steht die zeitgerechte Abnahme und Auslieferung des Fertigungssystems im Vordergrund. Diese können nun durch die automatisierte Erstellung von Reports und Abnahmeprotokollen unterstützt werden. Die meist geringe Zeit für die Durchführung und Auswertung der Tests wird durch Testautomatisierung optimal genutzt, ohne den Einsatz zusätzlicher Kapazitäten. Eine hohe Qualität in Bezug auf die Automatisierung und die Liefertreue des Fertigungssystems führt meist zu einer Imagesteigerung und, damit verbunden, zu einem Wettbewerbsvorteil.

Die Grenzen der neuen Testmethodik werden im Folgenden diskutiert. Durch die softwaretechnische und kommunikationstechnische Trennung von CAST Werkzeug und RT-HiL System müssen für Tests mit echtzeitkritischer Ausführung Erweiterungen am Simulationsmodell durchgeführt werden. Dies macht die Datenhaltung zwischen Versionen der Testfälle und dem Simulationsmodell aufwändiger. Die praktische Validierung zeigt, dass meist ein Zeitfenster von weit mehr als einem Steuerungstakt zum Auslösen, z. B. des Nothalts, in der Testdurchführung zur Verfügung steht.

Alle Ergebnisse eines Tests und deren Aussagekraft sind bei der ganzheitlichen Testautomatisierung an einer virtuellen Testumgebung von der Modellgüte der

Simulation abhängig. Das bedeutet, dass nicht abgebildete Zusammenhänge oder Fehler in den Simulationsmodellen das Ergebnis des Tests verfälschen.

Realisiert und damit validiert im Rahmen dieser Arbeit wurde die Verwendung von verschiedenen CAST Werkzeugen, nicht aber die Verwendung verschiedener Simulationswerkzeuge. Die neue Testmethodik wurde auf höchste Aussagekraft hin entworfen, daher kam ein Werkzeug zur Simulation in deterministischer Echtzeit zum Einsatz. Bestehen keine Echtzeitanforderungen an den Ansatz, können auch nichtechtzeitfähige Simulationswerkzeuge zum Einsatz kommen.

Per Definition kann das automatisierte Blackbox Testing keine Aussage über die Testabdeckung in den Steuerungsprogrammen machen.

Da es sich bei der ganzheitlichen Testautomatisierung um einen Ansatz zum System-, Abnahme- und Regressionstest handelt, können Fehler im Pflichtenheft oder darin vergessene Funktionen nicht entdeckt werden (Strassacker 1997, S. 117f). Die angewandten Testarten decken Abweichungen zum Pflichtenheft, zur Spezifikation sowie sporadische Fehler auf.

Für die Realisierung und die langfristige Nutzung eines virtuellen Bedieners ist ein nichtgrafischer Zugriff zu bevorzugen. Diese Beobachtung konnte in Versuchen gewonnen werden. Die Auswertung der Oberfläche auf grafischer Basis ist nicht so zuverlässig und nicht so robust wie mittels kontextbasierter Oberflächenautomatisierung. Schwierigkeiten treten z. B. nach einem Softwareupdate seitens des Steuerungsherstellers auf, welches kleine grafische Änderungen durchführt; oder das Ändern der Beschriftung eines Elements, bspw. durch Umschalten der Sprache oder Schriftart, oder auch die Änderung der Auflösung des Bildschirms führt zu Problemen bei der Wiedererkennung.

7.4 Zusammenfassung

Im letzten Inhaltskapitel dieser Arbeit wurde die Implementierung und Validierung des Beitrags zur ganzheitlichen Testautomatisierung für automatisierte Fertigungssysteme durchgeführt und dargestellt. Bei der Implementierung stand die Demonstration der Allgemeingültigkeit der Lösung im Vordergrund. Dazu wurden vier Realisierungen der Lösung demonstriert. Es konnte gezeigt werden, dass

die Methodik unabhängig von der Realisierung des CAST Werkzeugs umsetzbar ist und auf verschiedene Fertigungssysteme angewandt werden kann. Zur Validierung wurde in zwei CAST Werkzeugen dasselbe Testszenario umgesetzt und zur Durchführung gebracht. Im Vergleich mit dem manuellen Vorgehen war die Protokollierung und Erstellung des Testberichts automatisiert deutlich schneller und umfangreicher. Die zeitliche Streuung zwischen einzelnen Testdurchführungen war beim automatisierten Test ebenfalls deutlich geringer. Die komplette Zeitdauer beim automatisierten Test ist nur dann schneller als der manuelle Test, wenn ein optimiertes Werkzeug eingesetzt wird. Der Zeitvorteil bei der automatischen Ausführung ist dabei stark von der Zusammensetzung des Testszenarios abhängig: Bedienhandlungen können meist nicht schneller vom CAST Werkzeug durchgeführt werden, dafür aber die Verarbeitung großer Datenmengen.

Ein Steuerungshersteller hat die ganzheitliche Testautomatisierung in einem Pilotprojekt eingesetzt. Ziel war es, die Systemtests der eigenen Messzyklen automatisiert in einem Kundenszenario prüfen zu können. Die Beschreibung der Ergebnisse zeigt, dass die ganzheitliche Testautomatisierung in diesem Fall von ihrer virtuellen Umgebung zum schadensfreien Testen und der ständigen Verfügbarkeit profitiert. Durch die ganzheitliche Testautomatisierung erreicht der Steuerungshersteller zudem eine 100 %ige Testdurchführung aller Testszenarien seiner Zyklen ohne Mehraufwand, sowie eine vergleichbare Durchführung und Auswertung der Testszenarien, unabhängig vom Testpersonal. Dabei verringert sich der Aufwand je Testszenario für einen Zyklus von ca. 2 h auf ca. 5 min.

Im letzten Teil dieses Kapitels werden die Vorteile und Grenzen der Methodik zur ganzheitlichen Testautomatisierung, wie sie in dieser Arbeit entworfen wurde, diskutiert.

8 Zusammenfassung und Ausblick

Bedingt durch steigende Anforderungen seitens der Märkte und Kunden führte die zeitliche Entwicklung zu einem immer komplexeren Aufbau von Fertigungssystemen im Maschinenbau. Die steigende Komplexität senkt die Zuverlässigkeit und erschwert die Entwicklung der Systeme. Für diese Arbeit wird daher die Hypothese aufgestellt, dass die aktuelle Form des Testprozesses nicht ausreichend ist, um den künftigen Herausforderungen bei der Entwicklung von softwarelastigen, automatisierten Fertigungssystemen zu begegnen. Unter der Voraussetzung, dass die Hypothese standhält, wird die Forschungsfrage der Arbeit definiert als „Wie muss eine neue Methodik aussehen, welche den Testprozess von automatisierten Fertigungssystemen verbessert?“.

Zunächst werden in der Arbeit die Grundlagen zu den Bereichen industrielle Steuerungstechnik, Virtuelle Inbetriebnahme, zum Begriff des Fehlers und zu Softwaretest und Qualität vermittelt. Anschließend werden die bestehenden Defizite des aktuellen Testprozesses herausgearbeitet und dargelegt. Die Defizite können wie folgt zusammengefasst werden: Kein strukturiertes und methodisches Vorgehen und keine Werkzeugunterstützung in den Phasen des Testprozesses. Weiterhin macht die manuelle Testdurchführung Testergebnisse schwer vergleichbar und führt zu Fehlern. Aus den Defiziten werden Anforderungen an eine neue, zu erarbeitende Methodik gestellt. Die maßgeblichen Anforderungen sind eine ganzheitliche Betrachtung der Tätigkeiten im Rahmen des Testprozesses sowie die Automatisierung der Tätigkeiten.

In der Recherche zum Stand der Technik werden Arbeiten aus der Wissenschaft und Industrie zu Testautomatisierung und Qualitätssicherung von Steuerungsprogrammen im fertigungstechnischen Kontext untersucht. Die wissenschaftlichen Arbeiten werden anschließend gegen die Anforderungen geprüft. Keine der Arbeiten erfüllt die gestellten Anforderungen. Insbesondere die Anforderungen bezüglich einer herstellerunabhängigen Definition der Testfälle und eines Dauerlaufs bei der Testdurchführung sind nicht erfüllt. Daraus leitet sich der Handlungs-

bedarf und die konkrete Zielsetzung der Arbeit ab: eine ganzheitliche Testautomatisierung zu konzeptionieren, umzusetzen und zu evaluieren, welche in kausaler Folge die Qualität der Fertigungssysteme verbessert.

Zunächst erfolgt bei der Erarbeitung der Methodik ein Grobentwurf, der die Einordnung im Vorgehensmodell der Entwicklung betrachtet und Testwerkzeugtypen auf ihre Eignung hin untersucht. Zudem wird im Grobentwurf die Architektur der neuen Testumgebung erarbeitet. Im anschließenden Feinentwurf werden die einzelnen Methoden der ganzheitlichen Testautomatisierung entwickelt und mit den Anforderungen abgeglichen. Die konkreten Lösungen sind dabei:

- Eine fehlerbasierte Erweiterung von Simulationsmodellen zum gezielten Test von Schlechtfallszenarien
- Ein virtueller Bediener zur modellbasierten Generierung und robusten Ausführung von Oberflächenautomation
- Softwaretechnische und kommunikationstechnische Schnittstellen für den automatisierten Zugriff auf Steuerung und Simulation eines HiL Systems
- Eine herstellerunabhängige Definition von Testfällen mittels Keyword-Driven Testing
- Die Betrachtung einer grafischen Sprache für den Testentwurf
- Ein Vorgehen beim Testentwurf und der Wiederverwendung nach dem Bibliothekskonzept

Die beschriebenen Teilmethoden der gesamten Methodik zur ganzheitlichen Testautomatisierung werden im Anschluss implementiert und prototypisch eingesetzt. Die Implementierung erfolgt auf zwei Arten: Zum einen wird ein eigenes CAST Werkzeug umgesetzt, zum anderen wird ein bestehendes CAST Werkzeug angepasst und erweitert. Die beiden unterschiedlichen Wege zur Implementierung zeigen die universelle Gültigkeit der neuen Testmethodik. Beide Implementierungen werden anschließend mit der manuellen Testdurchführung vergleichend validiert. Die ganzheitliche Testautomatisierung erzielt deutliche Verbesserungen bei der zeitlichen Wiederholgenauigkeit (bis zu Faktor 6) und eine Aufwandsreduzierung bei Protokollierung und Testberichterstellung (bis zu Faktor 20). Bei den beiden CAST Werkzeugen zur Automatisierung wird ersichtlich,

dass ein optimiertes Werkzeug die notwendige Zeit für die Testdurchführung und -auswertung nochmals erheblich (> 50 %) verkürzt.

Im industriellen Einsatz wurde die ganzheitliche Testautomatisierung bei einem Steuerungshersteller für den Systemtest Messzyklen eingesetzt. Die in diesem Szenario erreichten Verbesserungen waren: Eine testerunabhängige Reproduzierbarkeit der Testdurchführung und -protokollierung, sowie die mühelose 100 %ige Abdeckung aller Testfälle bei jedem Regressionstest. Dabei verringerte sich der Aufwand je Testszenario für einen Zyklus von ca. 2 h auf ca. 5 min.

Die in dieser Arbeit erarbeitete und validierte Lösung erfüllt demnach alle aufgestellten Anforderungen und beantwortet die gestellte Forschungsfrage nach einem Ansatz, welcher den Testprozess von softwarelastigen, automatisierten Fertigungssystemen verbessert. Die Lösung stellt einen ganzheitlichen Ansatz dar: Neben dem Schwerpunkt einer automatisierten Testdurchführung und -auswertung, durch Einsatz eines CAST Werkzeugs, werden die Testumgebung, die Testerstellung, Testmethoden und das Testpersonal berücksichtigt.

Der weiterhin manuelle Aufwand im Testprozess beinhaltet Testentwurf und Testrealisierung. Es existieren Arbeiten, z. B. die von (Tatsumi 1987; Feldmann et al. 2014; Kormann 2016; Rösch 2016), um Testfälle und Testdaten automatisiert zu generieren. Die Verknüpfung einer dieser Arbeiten mit der vorliegenden Arbeit würde den manuellen Aufwand im gesamten Testprozess bei der Softwareentwicklung von Fertigungssystemen weiter reduzieren.

Weiterhin erscheint es interessant zu untersuchen, ob die Kopplung der in dieser Arbeit entwickelten Methodik zur ganzheitlichen Testautomatisierung zusammen mit dem Ansatz der testgetriebenen Entwicklung aus der Softwaretechnik (Beck 2003) gewinnbringend einsetzbar ist. Dazu sollten die Arbeiten von (Hametner 2013) zur testgetriebenen Definition und Ausführung von Testfällen im Umfeld der Automatisierungstechnik und (Hackenberg 2018) zur Übertragung und Anpassung von testgetriebener Entwicklung auf cyber-physische Systeme Berücksichtigung finden.

Auf Basis der in dieser Arbeit entwickelten Methodik zur ganzheitlichen Automatisierung von Systemtests an Fertigungssystemen können weitere Einsatzfelder für die Automatisierung von Entwicklungs- und Qualitätssicherungsaufgaben im Maschinen- und Anlagenbau in Betracht gezogen werden:

- Für die automatisierte Qualitätsbewertung von Simulationsmodellen (Barth et al. 2020; Barth et al. 2021) für den Maschinen- und Anlagenbau.
- Für die automatisierte Optimierung von Prozessparametern der Fertigung an der Systemsimulation, bspw. für Tiefziehpressen zum Erhalt der Bauteilqualität (Drossel et al. 2016). Voraussetzung ist eine Systemsimulation, die neben den Maschinenmodellen auch ein echtzeitfähiges Prozessmodell beinhaltet. Das CAST Werkzeug muss um Methoden, bspw. der Künstlichen Intelligenz, erweitert werden, um optimale Einstellungen der Prozessparameter zu identifizieren.
- Für die angeleitete Durchführung und anwendergerechte Auswertung von Schulungsszenarien in virtuellen Umgebungen (Hönig et al. 2021). Dazu stellt das CAST Werkzeug das passende Schulungsszenario ein, überprüft die Schritte des Bedieners und leitet durch das Szenario. Am Ende setzt das CAST Werkzeug die Umgebung wieder in den Ausgangszustand zurück und kann die Leistung des Teilnehmers bewerten.
- Für die automatisierte Absicherung von Agenten, welche im Rahmen von autonomen und selbstadaptierenden Maschinen- und Anlagensystemen (Möhring et al. 2020) zum Einsatz kommen.
- Für automatisierte Benchmarkvorgänge zwischen Steuerungssystemen, Simulationsumgebungen und gesamten Fertigungssystemen.

Für die Einführung in der Industrie und die standardmäßige Nutzung von ganzheitlicher Testautomatisierung wäre zu untersuchen, welche Überzeugungsarbeit im Unternehmen und in den entsprechenden Entwicklungsprozessen zu leisten ist. Nicht zuletzt wird die allgemeine Entscheidung zur Einführung, ähnlich wie bei der VIBN, über eine Kosten-Nutzen-Analyse gefällt werden.

Anhang

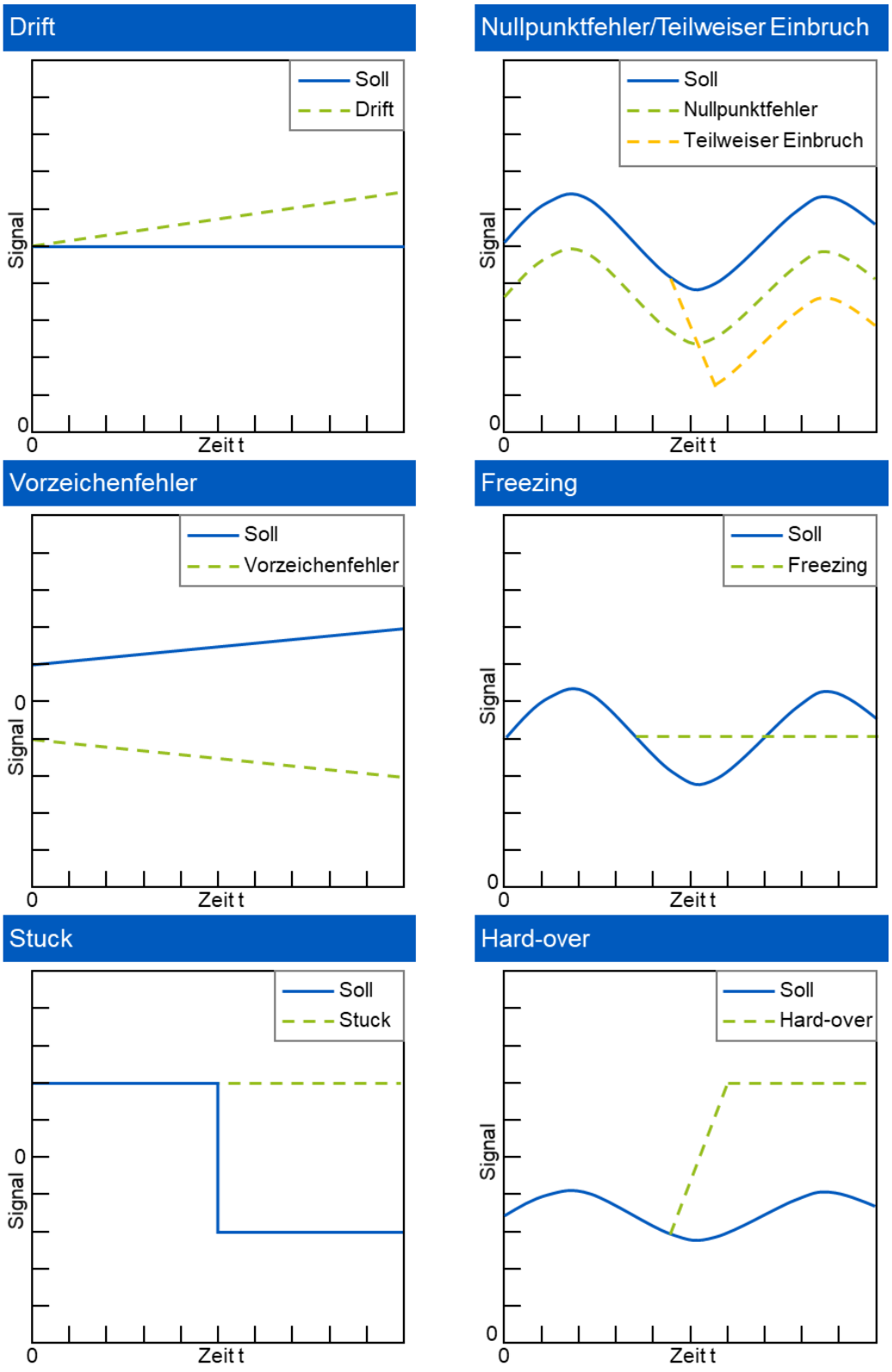


Abbildung 58 Erster Teil der identifizierten Signalverläufe für fehlerbasierte Erweiterung von Simulationsmodellen

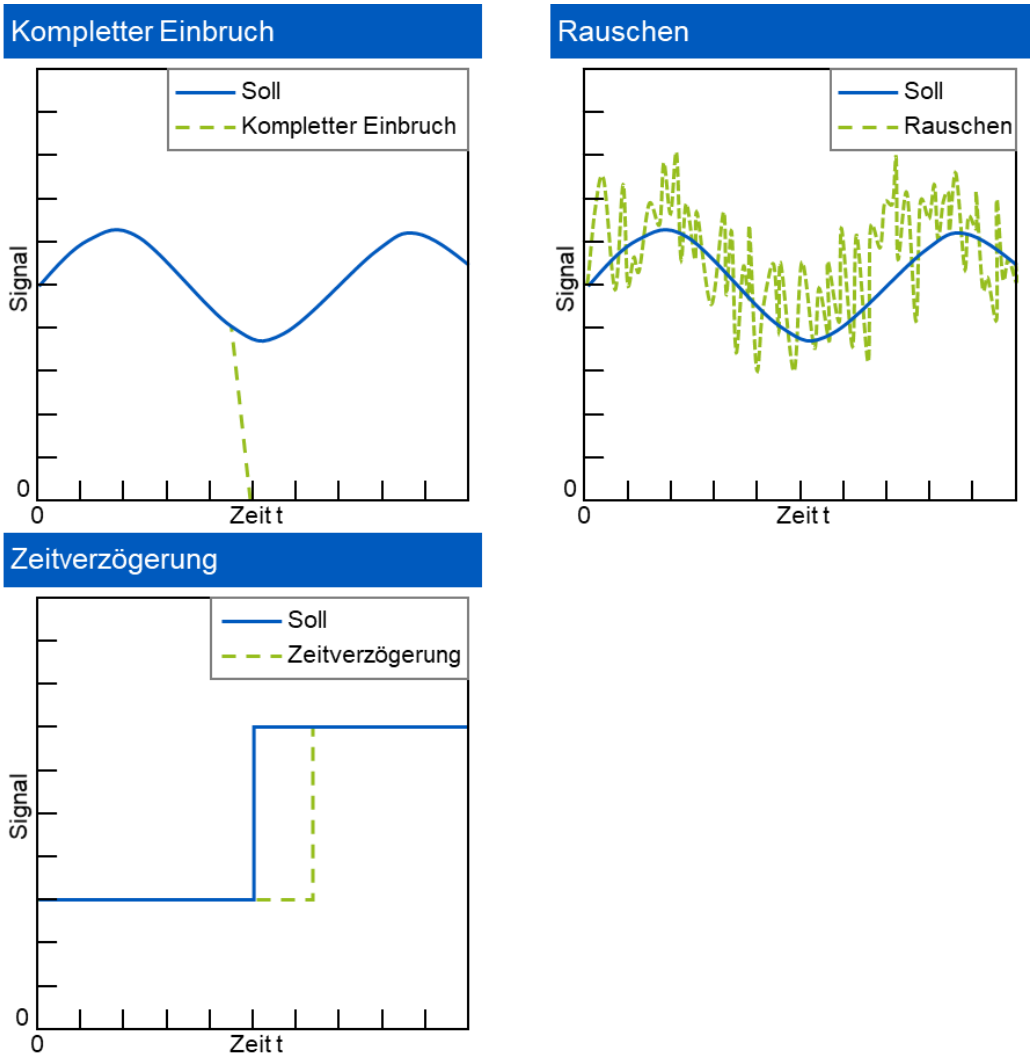


Abbildung 59 Zweiter Teil der identifizierten Signalverläufe für fehlerbasierte Erweiterung von Simulationsmodellen

Literaturverzeichnis

3S-Smart Software Solutions GmbH 2017
3S-Smart Software Solutions GmbH, 2017.
CODESYS Test Manager: Datenblatt, Version 4.2.
Verfügbar unter: https://store.codesys.com/codesys-test-manager.html
Zugriff am: 06.03.2022

Arcuri 2008
Arcuri, Andrea, 2008.
On the Automation of Fixing Software Bugs.
In: *Companion of the 30th international conference on software engineering,*
10.-18. Mai, Leipzig, Deutschland, S. 1003–1006

Balzert 2008
Balzert, Helmut, 2008.
Softwaremanagement.
2. Aufl.
Heidelberg : Spektrum Akad. Verl.
Lehrbücher der Informatik / Helmut Balzert ; 3.
ISBN 9783827411617
Verfügbar unter: http://deposit.d-nb.de/cgi-bin/dok-serv?id=3030756&prov=M&dok_var=1&dok_ext=htm

Balzert et al. 2011
Balzert, Helmut; Liggesmeyer, Peter, 2011.
Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb.
3. Aufl.
Heidelberg : Spektrum Akademischer Verlag.
Lehrbücher der Informatik.
ISBN 978-3-8274-1706-0
Verfügbar unter: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10494391DOI:
10.1007/978-3-8274-2246-0

Barth et al. 2020
Barth, Mike; Kübler, Karl; Heinzerling, Till; Rosen, Roland; Jäkel, Jens, 2020. Eine systematische Bewertung der Qualität von Simulationsmodellen: Identifikation und Clustering von Qualitätskriterien.
atp magazin (06-07), S. 68–75

Barth et al. 2021
Barth, Mike; Rosen, Roland; Kübler, Karl; Jäkel, Jens; Ristic, Mirjana; Heinzerling, Till; Scheifele, Christian, 2021. Eine systematische Bewertung der Qualität von Simulationsmodellen: Teil 2: Bewertungsmetrik und Validierung.
atp magazin (04), S. 56-63

Bauer 1995 Bauer, Roman, 1995.
Hard- und Software im Entwicklungsprozeß integrierter Produkte: Die Anwendung des Just-in-Time-Konzeptes in Forschung & Entwicklung.
Gabler Edition Wissenschaft.
Wiesbaden, s.l. : Deutscher Universitätsverlag.
ISBN 3824461161
Verfügbar unter: http://dx.doi.org/10.1007/978-3-663-08539-3DOI: 10.1007/978-3-663-08539-3

Beck 2003 Beck, Kent, 2003.
Test-driven development: by example : Addison-Wesley Professional

Beckhoff Automation GmbH & Co. KG Beckhoff Automation GmbH & Co. KG.
Beckhoff Information System: TwinCAT ADS.
Verfügbar unter: https://infosys.beckhoff.com/index.php?content=./content/1031/tcadscommon/html/tcadscommon_introads.htm&id=
Zugriff am: 06.03.2022

Bender 1999 Bender, Klaus (Hrsg.), 1999.
Echtzeitsimulation zum Test von Maschinensteuerungen: [Abschlußdatum des BMBF SEMI Vorhabens: Oktober 1998].
München : Herbert Utz Verlag GmbH.
ISBN 3896754823
Verfügbar unter: http://edok01.tib.uni-hannover.de/edoks/e01fbdig09/593806514.pdf

Bender 2005 Bender, Klaus (Hrsg.), 2005.
Embedded Systems: Qualitätsorientierte Entwicklung.
Berlin : Springer.
ISBN 978-3-540-27370-7

Benington 1983 Benington, Herbert D., 1983. Production of Large Computer Programs.
IEEE Annals of the History of Computing **5** (4), S. 350–361
DOI: 10.1109/MAHC.1983.10102

Bertsche et al. 2009 Bertsche, Bernd; Göhner, Peter; Jensen, Uwe; Schinköthe, Wolfgang; Wunderlich, Hans-Joachim, 2009.
Zuverlässigkeit mechatronischer Systeme: Grundlagen und Bewertung in frühen Entwicklungsphasen.
Berlin, Heidelberg : Springer Berlin Heidelberg.
ISBN 978-3-540-85089-2
DOI: 10.1007/978-3-540-85091-5

Biallas 2016 Biallas, Sebastian, 2016.
Verification of programmable logic controller code using model checking and static analysis.
Aachen, Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Dissertation, 2016

Bibliographisches Institut GmbH 2019 Bibliographisches Institut GmbH, 2019.
Duden | Bestätigung | Rechtschreibung, Bedeutung, Definition, Herkunft.
Berlin.
Verfügbar unter: https://www.duden.de/rechtschreibung/Bestaetigung

Bock 2015 Bock, Hans-Peter, 2015.
Fehlertolerante numerische Steuerung, Zugl.: Stuttgart, Univ., Diss., 2014.
Stuttgart : Fraunhofer-Verl.
Stuttgarter Beiträge zur Produktionsforschung 45.
ISBN 9783839608906
Verfügbar unter: http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-101203

Bosch Rexroth AG 2011 Bosch Rexroth AG (Hrsg.), 2011.
Rexroth IndraMotion MTX 12VRS OPC-Kommunikation: Projektierungsanleitung

Bourque et al. 2014 Bourque, Pierre; Fairley, Richard E., 2014.
Guide to the software engineering body of knowledge SWEBOK v3.0.
Version 3.0.
[Los Alamitos, Calif.] : IEEE Computer Society.
ISBN 978-0-7695-5166-1

Brecher et al. 2010 Brecher, C; Verl, A; Lechler, A; Servos, M., 2010. Open control systems: state of the art.
Production Engineering **4** (2-3), S. 247–254
DOI: 10.1007/s11740-010-0218-5

Bröhl et al. 1995 Bröhl, Adolf-Peter; Dröschel, Wolfgang (Hrsg.), 1995.
Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden.
2. Aufl.
München, Wien : R. Oldenbourg Verlag.
ISBN 3-486-23470-6

Bucsics et al. 2015 Bucsics, Thomas; Baumgartner, Manfred; Seidl, Richard; Gwihs, Stefan, 2015.
Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken.
2nd ed.
Heidelberg : dpunkt.verlag.
ISBN 9783864901942

Bürger 2005 Bürger, Thomas, 2005.
Durchgängige analytische Qualitätssicherung für numerische Steuerungssoftware.
Stuttgart, Universität Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW), Dissertation, 2005

Bursac 2016 Bursac, Nikola, 2016.
Model Based Systems Engineering zur Unterstützung der Baukastenentwicklung im Kontext der Frühen Phase der Produktgenerationsentwicklung.
Karlsruhe, Karlsruher Institut für Technologie (KIT), Dissertation, 2016.
http://digbib.ubka.uni-karlsruhe.de/volltexte/1000054484

Business Wire 2016 Business Wire, 2016.
FANUC Brings New Look to Machine Tool Controls with iHMI at IMTS 2016.
Rochester Hills, MI.
Verfügbar unter: https://www.businesswire.com/news/home/20160920006403/en/FANUC-Brings-New-Machine-Tool-Controls-iHMI
Zugriff am: 06.03.2022

Buwalda 1999 Buwalda, Hans, 1999.
Testing with action words.
In: Fewster, Mark; Graham, Dorothy (Hrsg.)*: Software test automation*.
Reading, MA : Addison-Wesley, S. 446–464
ISBN 0-201-33140-3

Coupek et al. 2016 Coupek, D; Lechler, A; Verl, A., 2016.
Cloud-based control for downstream defect reduction in the production of electric motors.
In: *2016 International ESARS-ITEC,*
02.11.2016 - 04.11.2016, Toulouse, France

Daniel 2015 Daniel, Christian, 2015. Virtuelle Komponenten verkürzen Inbetriebnahme.
MM MaschinenMarkt (41), S. 40–42
Verfügbar unter: https://www.isg-stuttgart.de/fileadmin/user_upload/news/MM41-15S40-42.pdf
Zugriff am: 06.03.2022

Daniel 2018 Daniel, Christian, 2018. Digitale Zwillinge aus virtuellen Baugruppen: Simulationslösung reduziert Inbetriebnahmezeiten und Projektentwicklungskosten.
VDI-Z integrierte Produktion **160** (4), S. 1–2

Denkena et al. 2007 Denkena, Berend; Brecher, Christian (Hrsg.), 2007.
Ramp-Up/2 - Anlaufoptimierung durch Einsatz virtueller Fertigungssysteme.
1. Aufl.
Frankfurt am Main : VDMA-Verl.
ISBN 9783816305460

Dijkstra 1972 Dijkstra, Edsger W., 1972. The humble programmer, S. 859–866.
Communications of the ACM **15** (10), 1972

Dornseiff et al. 2001 Dornseiff, M; Stahl, M; Sieger, M; Sax, Eric, 2001. Durchgängige Testmethoden für komplexe Steuerungssysteme.
In: *Elektronik im Kraftfahrzeug,*
27.-28. September, Baden-Baden, S. 347–366

Dr. Johannes Heidenhain GmbH 2014 Dr. Johannes Heidenhain GmbH (Hrsg.), 2014.
RemoTools SDK virtualTNC: Software-Schnittstelle zur Interaktion mit TNC-Steuerungen, Informationen für den Maschinenhersteller

Drossel et al. 2016 Drossel, Welf-Guntram; Ihlenfeldt, Steffen; Penter, Lars; Zorn, Wolfgang; Hellmich, Arvid; Grüning, Tina, 2016. Das Presswerk auf dem Weg in das Cyber-physische Zeitalter.
ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb **111** (7-8), S. 460–465
DOI: 10.3139/104.111566

Eigner et al. 2014 Eigner, Martin; Roubanov, Daniil; Zafirov, Radoslav, 2014.
Modellbasierte virtuelle Produktentwicklung.
Berlin, Heidelberg : Springer Berlin Heidelberg.
ISBN 978-3-662-43815-2
DOI: 10.1007/978-3-662-43816-9

Eigner et al. 2017 Eigner, Martin; Koch, Walter; Muggeo, Christian (Hrsg.), 2017.
Modellbasierter Entwicklungsprozess cybertronischer Systeme.
Berlin, Heidelberg : Springer Berlin Heidelberg.
ISBN 978-3-662-55123-3
DOI: 10.1007/978-3-662-55124-0

Fanuc Corporation 2016 Fanuc Corporation (Hrsg.), 2016.
FANUC FOCAS1/2 Library: Handbuch Version 4.9

Feldmann et al. 2014 Feldmann, Stefan; Rösch, Susanne; Legat, Christoph; Vogel-Heuser, Birgit, 2014.
Keeping requirements and test cases consistent.
In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN),*
27.07.2014 - 30.07.2014, Porto Alegre RS, Brazil, S. 726–732

Frank et al. 2016 Frank, Tobias; Koop, Harry, 2016. Bessere Qualität bei geringeren Kosten.
DESIGN&ELEKTRONIK (10), S. 41–43

Freyer 2017 Freyer, Beate, 2017.
Warum ohne die Virtuelle Inbetriebnahme heute nichts geht.
Verfügbar unter: https://www.machineering.de/blog/unserer-experten-berichten/article/warum-ohne-die-virtuelle-inbetriebnahme-heute-nichts-geht/
Zugriff am: 09.08.2018

Friedrich Braun GmbH Friedrich Braun GmbH.
Lagerausfälle – Ursachen erkennen & Ausfälle vermeiden.
Münster.
Verfügbar unter: https://www.braun-waelzlager.de/de/aktuelles/lagerausfalle-und-ihre-ursachen/
Zugriff am: 06.03.2022

Frieß 2014 Frieß, Michael, 2014.
Effizienter Projektverlauf mit Robotersimulation und Virtueller Inbetriebnahme.
In: *6. mav Innovationsforum,*
01. April, Böblingen

Früh et al. 2015 Früh, K. F; Schaudel, Dieter; Maier, Uwe; Bleich, René (Hrsg.), 2015.
Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen.
5. komplett überarb. Aufl.
München : DIV Dt. Industrieverl.
ISBN 978-3-8356-3372-8

Frühauf et al. 2007 Frühauf, Karol; Ludewig, Jochen; Sandmayr, Helmut, 2007.
Software-Prüfung: Eine Anleitung zum Test und zur Inspektion.
6., überarb. und aktualisierte Aufl.
Zürich : vdf Hochsch.-Verl. an der ETH.
vdf-Lehrbuch Informatik.
ISBN 9783728130594
Verfügbar unter: http://www.vdf.ethz.ch/info/showDetails.asp?isbnNr=3059

Galin 2004 Galin, Daniel, 2004.
Software quality assurance: From theory to implementation.
Harlow : Pearson/Addison Wesley.
ISBN 0201709457

Gamblin 2014 Gamblin, Russell, 2014.
Machine tools: Specification, purchase, and installation.
New York : McGraw-Hill Ed.
ISBN 978-0-07-181222-1

Gausemeier et al. 2013 Gausemeier, Jürgen; Dumitrescu, Roman; Steffen, Daniel; Czaja, Anja; Wiederkehr, Olga; Tschirner, Christian, 2013.
Systems Engineering in der industriellen Praxis: Studie.
Paderborn.
Verfügbar unter: https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Seniorprofessur_Gausemeier/systemsengineerings/Studie_Systems-Engineering.pdf
Zugriff am: 06.03.2022

Getriebebau NORD GmbH & Co. KG 2016 *Motoren: Betriebs- und Montageanleitung*.
Bargteheide

GfK Marktforschung et al. 2000 GfK Marktforschung; Fraunhofer-Institut für Experimentelles Software Engineering IESE; Fraunhofer-Institut für Systemtechnik und Innovationsforschung ISI, 2000. *Analyse und Evaluation der Softwareentwicklung in Deutschland: Eine Studie für das Ministerium für Bildung und Forschung*, 01.12.2000.
Nürnberg
Verfügbar unter: https://www.iese.fraunhofer.de/content/dam/iese/de/documents/Softwareentwicklung_Deutschland_Evasoft_tcm122-7431.pdf
Zugriff am: 06.03.2022

Graham et al. 1993 Graham, J. A; Higgott, J. B; Hynes, A. J., 1993. Bridging the gap between CASE and CAST.
In: *Proceedings 1993 Software Engineering Standards Symposium,*
30 Aug.-3 Sept. 1993, Brighton, UK, S. 317–324

Gräßler 2017 Gräßler, Iris, 2017.
A new V-Model for interdisciplinary product engineering.
In: *59th Ilmenau Scientific Colloquium,*
11. - 15. September, Ilmenau

Gühmann et al. 2002 Gühmann, Clemens; Riese, J., 2002. Testautomatisierung in der Hardware-in-the-Loop Simulation.
In: *Tagungsband der AUTOREG 2002,*
15.-16. April, Mannheim, S. 511–527

Haberfellner 2015 Haberfellner, Reinhard (Hrsg.), 2015.
Systems Engineering: Grundlagen und Anwendung.
13., aktualisierte Aufl.
Zürich : Orell Füssli.
ISBN 978-3-280-04068-3

Hackenberg 2018 Hackenberg, Georg, 2018.
Test-driven conceptual design of cyber-physical manufacturing systems.
München, Technische Universität München, Dissertation, 2018

Hackenberg et al. 2015
Hackenberg, Georg; Richter, Christoph; Zäh, Michael, 2015.
IMoMeSA - Abschlussbericht: Integrierte modellbasierte Entwicklung mechatronischer Systeme im Maschinen- und Anlagenbau.
München, Technische Universität München, TUM, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Abschlussbericht, 2015.
https://mediatum.ub.tum.de/attfile/1253882/incoming/2015-May/570232.pdf

Hametner 2013
Hametner, Reinhard, 2013.
Test driven software development for improving the quality of control software for industrial automation software.
Wien, Technische Universität Wien, Institut für Automatisierungs- und Regelungstechnik, Dissertation, 2013

Hampp 2010
Hampp, Tilmann, 2010.
Ein Kosten-Nutzen-Modell für die Softwareprüfung.
Stuttgart, Universität Stuttgart, Institut für Softwaretechnologie, Dissertation, 2010

Helfen et al. 2006
Helfen, Markus; Lauer, Michael; Trauthwein, Hans Martin, 2006.
SAP-Lösungen testen: Methodik, Funktionstests mit eCATT, Performancetests.
1. Aufl.
Bonn : Galileo Press.
SAP Press.
ISBN 3-89842-721-8

Herfs 2010
Herfs, Werner Josef, 2010.
Modellbasierte Software in the Loop Simulation von Werkzeugmaschinen.
Aachen, Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Werkzeugmaschinenlabor (WZL), Dissertation, 2010

Herkommer 2015
Herkommer, Günter, 2015.
Neue HMI-Ära eingeläutet.
Verl.
Verfügbar unter: https://www.computer-automation.de/steuerungsebene/bedienen-beobachten/artikel/125241/
Zugriff am: 06.03.2022

Herrman et al. 2015 Herrman, Christoph; Reinhart, Gunther; Schuh, Günther; Spengler, Thomas; Vietor, Thomas; Drescher, Benny; Gäde, Maren; Klein, Thorsten; Richter, Timo; Schönemann, Malte; Spiegelberger, Bernd; Vogels, Till, 2015. Strategien, Methoden und Werkzeuge für die Entwicklung mechatronischer Produkte: Ergebnisse und Synergiepotenziale aus den Forschungsprojekten GiBWert, MEPROMA und SynProd.
ZWF – Zeitschrift für wirtschaftlichen Fabrikbetrieb **110** (5), S. 251–255

Hoffmann 2013 Hoffmann, Dirk W., 2013.
Software-Qualität.
2., aktualisierte und korrigierte Aufl.
Berlin : Springer Vieweg.
eXamen.press.
ISBN 978-3-642-35700-8

Hönig et al. 2021 Hönig, Jana; Schnierle, Marc; Scheifele, Christian; Spielmann, Tobias; Münster, Carlos; Roth, Armin; Röck, Sascha; Verl, Alexander, 2021. Mixed-Reality-in-the-Loop Simulation: von Produktionssystemen zur Aus- und Weiterbildung.
atp magazin (01)

Huckle 2015 Huckle, Thomas, 2015.
Collection of software bugs.
München.
Verfügbar unter: https://www5.in.tum.de/persons/huckle/bugse.html
Zugriff am: 06.03.2022

Huuck 2003 Huuck, Ralf, 2003.
Software verification for programmable logic controllers.
Kiel, Christian-Albrechts-Universität zu Kiel, Dissertation, 2003.
http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_726

Isermann 2008 Isermann, Rolf, 2008.
Mechatronische Systeme.
Berlin, Heidelberg : Springer Berlin Heidelberg.
ISBN 978-3-540-32336-5
DOI: 10.1007/978-3-540-32512-3

ISG Industrielle Steuerungstechnik GmbH 2018
ISG Industrielle Steuerungstechnik GmbH (ISG), 2018. *Rhythmus für Ihre Stars: Das Testautomatisierungstool für Steuerungssoftware*, Produktbroschüre ISG-dirigent. Stuttgart, Deutschland.
Verfügbar unter: https://www.isg-stuttgart.de/fileadmin/user_upload/2018_11_06_Website_ISG-dirigent.pdf
Zugriff am: 06.03.2022

ISG Industrielle Steuerungstechnik GmbH 2021
ISG Industrielle Steuerungstechnik GmbH (ISG), 2021. *Dokumentation ISG-kernel: Handbuch Zyklen - Kalibrieren und Messen*.
Stuttgart.
Verfügbar unter: https://ftp.isg-stuttgart.de/download/kernel/documentation/de/Zyklen/cycles_calib_measure_de.pdf
Zugriff am: 06.03.2022

ITQ GmbH 2016
ITQ GmbH, 2016a. *Erfahrungsbericht: Effiziente Testautomatisierung mechatronischer Produkte*, 01.10.2016.
Garching bei München.
Verfügbar unter: https://www.itq.de/wp-content/uploads/2016/10/Erfahrungsbericht_effiziente_Testautomatisierung_mechatronischer_Produkte.pdf
Zugriff am: 06.03.2022

ITQ GmbH 2016
ITQ GmbH, 2016b. *Testautomatisierung: Nachhaltige Qualitätssicherung auch bei höchsten Echtzeitanforderungen*, 01.11.2016.
Garching bei München.
Verfügbar unter: https://www.itq.de/wp-content/uploads/2016/11/testautomatisierung.pdf
Zugriff am: 06.03.2022

Jan et al. 2017
Jan, Sana Ullah; Lee, Young-Doo; Shin, Jungpil; Koo, Insoo, 2017. Sensor Fault Classification Based on Support Vector Machine and Statistical Time-Domain Features.
IEEE Access **5**, S. 8682–8690
DOI: 10.1109/ACCESS.2017.2705644

Janisch Janisch, Josef. Kleiner Effekt – Große Wirkung: Was Sie schon immer über Hallsensoren wissen wollten, S. 2–5.
all-electronics **2006** (07)
Verfügbar unter: http://www.all-electronics.de/wp-content/uploads/migrated/article-pdf/75149/ei06-07-067.pdf
Zugriff am: 02.09.2019

JUMO GmbH & Co. KG 2009 JUMO GmbH & Co. KG (Hrsg.), 2009.
Aufbau und Anwendung von Thermoelementen.
Fulda

Kaiser 2013 Kaiser, Lydia, 2013.
Rahmenwerk zur Modellierung einer plausiblen Systemstruktur mechatronischer Systeme.
Paderborn, Universität Paderborn, Heinz-Nixdorf-Institut, Dissertation, 2013

Kaner 2003 Kaner, Cem, 2003.
An introduction to scenario testing.
Melbourne, FL.
Verfügbar unter: http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf
Zugriff am: 06.03.2022

Keinert et al. 2015 Keinert, Matthias; Lechler, Armin, 2015.
OPC UA Informationsmodell für CNC-Datenschnittstellen.
In: *Automation 2015,*
11.-12. Juni 2015, Baden-Baden

Kohring 1993 Kohring, Arne, 1993.
Systematisches Projektieren und Testen von Steuerungssoftware für Werkzeugmaschinen, Zugl.: Aachen, Techn. Hochsch., Diss., 1993.
Aachen : Shaker.
Berichte aus der Produktionstechnik 93,13.
ISBN 3861115875

Kollmann et al. 1954 Kollmann, K; Bussmann, K. H; Dahl, A; Morchutt, G; Arp, B; Pahl, Obering F; Link, E; Pietsch, P; Eisenach, A; Meitzner, H; Bensinger, W. D; Martyrer, E; Benz, W; Maurer, A; Maier, A; Fuhrmann, E., 1954. *Riementriebe, Kettentriebe, Kupplungen: Vorträge u. Diskussionsbeiträge der Fachtagung „Antriebselemente“, Essen 1953*.
Wiesbaden, s.l. : Vieweg+Teubner Verlag.
Schriftenreihe Antriebstechnik 12.
ISBN 978-3-663-02896-3
Verfügbar unter: http://dx.doi.org/10.1007/978-3-663-02896-3DOI: 10.1007/978-3-663-02896-3

Kormann 2016 Kormann, Benjamin, 2016.
Automatische Testdatengenerierung zur Absicherung fehlerbehafteter reaktiver Automatisierungssysteme.
Clausthal, Technische Universität Clausthal, Dissertation, 2016

Kormann et al. 2012 Kormann, Benjamin; Vogel-Heuser, Birgit; Friedrich, Markus, 2012.
Befragung deutscher Maschinenbauunternehmen zum Thema Softwaretest.
In: *Automation 2012,*
13.-14. Juni 2012, Baden-Baden

Korotkiy 2010 Korotkiy, Dmitry, 2010.
Universelle Testsystem-Architektur in der Mechatronik, Zugl.: München, Techn. Univ., Diss., 2010.
1. Aufl.
Göttingen : Sierke Verlag.
Informationstechnik im Maschinenwesen.
ISBN 9783868442380

Krebser et al. 2018 Krebser, Gerhard; Kübler, Karl, 2018. Den Test automatisieren.
computer & automation (11), Sonderdruck

Kretschmer et al. 2016 Kretschmer, Felix; Lechler, Armin; Verl, Alexander, 2016.
Gelbe Seiten für Industrie 4.0.
In: *Automation 2016,*
07.-08. Juni2016, Baden-Baden

Kreusch 2002 Kreusch, Karsten, 2002.
Verifikation numerischer Steuerungen an virtuellen Werkzeugmaschinen, Zugl.: Cottbus, BTU, Diss., 2001.
Aachen : Shaker.
Berichte aus der Steuerungs- und Regelungstechnik.
ISBN 978-3-8322-0023-7

Kübler et al. 2016 Kübler, Karl; Verl, Alexander, 2016.
Testautomatisierung im Kontext der virtuellen Inbetriebnahme von Werkzeugmaschinen.
In: Krüger, Jörg; Uhlmann, Eckart; Verl, Alexander; Klemm, Peter; Brecher, Christian (Hrsg.)*: Industrie 4.0 - Visionen der Steuerungstechnik/im MES-Umfeld*.
Düsseldorf : VDI Verlag, S. 16–27
ISBN 978-3-18-369402-0

Kübler et al. 2018 Kübler, Karl; Krebser, Gerhard, 2018. Steuerungssoftware automatisiert prüfen.
MM MASCHINENMARKT DER FOKUS **124** (4), S. 28–31

Kübler et al. 2020 Kübler, Karl; Schopper, Dominik; Riedel, Oliver; Rudolph, Stephan, 2020. Towards an Automated Product-Production System Design - Combining Simulation-based Engineering and Graph-based Design Languages.
Procedia Manufacturing **52**, S. 258–265
DOI: 10.1016/j.promfg.2020.11.043

Kufner 2012 Kufner, Annika, 2012.
Automatisierte Erstellung von Maschinenmodellen für die Hardware-in-the-Loop-Simulation von Montagemaschinen.
Stuttgart, Universität Stuttgart, ISW, Dissertation, 2012.
http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-72958

Laible 2005 Laible, Ulrich, 2005.
Aufbau numerischer Steuerungssysteme für sicherheitskritische Anwendungen.
Stuttgart, Universität Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW), Dissertation, 2005

Legat et al. 2014 Legat, Christoph; Mund, Jakob; Campetelli, Alarico; Hackenberg, Georg; Folmer, Jens; Schütz, Daniel; Broy, Manfred; Vogel-Heuser, Birgit, 2014. Interface behavior modeling for automatic verification of industrial automation systems' functional conformance.
at - Automatisierungstechnik **62** (11), S. 815–825
DOI: 10.1515/auto-2014-1126

Liggesmeyer 2009 Liggesmeyer, Peter, 2009.
Software-Qualität: Testen, Analysieren und Verifizieren von Software.
2. Aufl.
s.l. : Spektrum Akademischer Verlag.
ISBN 9783827420565

Linder 2008 Linder, Paul, 2008.
Constraintbasierte Testdatenermittlung für Automatisierungssoftware auf Grundlage von Signalflussplänen.
Stuttgart, Universität Stuttgart, Institut für Automatisierungs- und Softwaretechnik, Dissertation, 2008.
http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-35871

Linke 2011 Linke, Jörn, 2011. Der SPS-Benchmark.
Computer Automation (9), S. 55–58

Lohmeyer 2013 Lohmeyer, Quentin, 2013.
Menschzentrierte Modellierung von Produktentstehungssystemen unter besonderer Berücksichtigung der Synthese und Analyse dynamischer Zielsysteme.
Karlsruhe, Karlsruher Institut für Technologie (KIT), Institut für Produktentwicklung (IPEK), Dissertation, 2013.
DOI: 10.5445/IR/1000035102

Lohse et al. 2007 Lohse, Wolfram; Herfs, Werner; Brecher, Christian; Eppler, Claus, 2007.
Anwendungsorientierte Mechatroniksimulation mit realen und virtuellen Steuerungen.
In: Denkena, Berend; Brecher, Christian (Hrsg.)*: Ramp-Up/2 - Anlaufoptimierung durch Einsatz virtueller Fertigungssysteme*.
1. Aufl.
Frankfurt am Main : VDMA-Verl., S. 100–128
ISBN 9783816305460

Ludewig et al. 2010 Ludewig, Jochen; Lichter, Horst, 2010.
Software Engineering: Grundlagen, Menschen, Prozesse, Techniken.
2., überarb., aktualisierte und erg. Aufl.
Heidelberg : Dpunkt-Verl.
ISBN 9783898646628

McCall et al. 1977 McCall, Jim A; Richards, Paul K; Walters, Gene F., 1977.
Factors in software quality: Concept and definition of software quality, Volume 1.
Mass., USA : United States Air Force

Menzel et al. 2004 Menzel, Thomas; Stöppler, Guido; Wagner, Peter; Wünsch, Georg, 2004. Simulation erhöht Produktivität und Verfügbarkeit von Werkzeugmaschinen.
ZWF – Zeitschrift für wirtschaftlichen Fabrikbetrieb **99** (10), S. 534–537
DOI: 10.3139/104.100846

Mertke et al. 2001 Mertke, Thomas; Frey, Georg, 2001.
Formal verification of plc-programs generated from signal interpreted petri nets.
In: *Proceedings of the 2001 Conference on Systems, Man and Cybernetics,*
07.-10. Oktober, Tucson, Arizona, USA, S. 2700–2705

Metzner 2020 Metzner, Anja, 2020.
Software-Engineering: kompakt.
München : Carl Hanser Verlag.
ISBN 978-3-446-45949-6

Mewes 1996 Mewes, Jürgen, 1996. Qualitätssicherung der Automatisierungssoftware.
Industrie Management **12** (4), S. 50–52

Mewes 2005 Mewes, Jürgen, 2005.
Virtuelle Inbetriebnahme mit realen Automatisierungssystemen und virtuellen Maschinen.
Henningsdorf.
Verfügbar unter: https://www.winmod.de/de/uploads/Mewes_virtuelle%20Inbetriebnahme-vortrag2005.pdf
Zugriff am: 09.08.2018

Möhring et al. 2020 Möhring, H.-C; Wiederkehr, P; Erkorkmaz, K; Kakinuma, Y., 2020. Self-optimizing machining systems. *CIRP Annals - Manufacturing Technology* **69** (2), S. 740–763
DOI: 10.1016/j.cirp.2020.05.007

Möller 1996 Möller, Karl Heinrich, 1996.
Ausgangsdaten für Qualitätsmetriken — Eine Fundgrube für Analysen.
In: Ebert, Christof; Dumke, Reiner (Hrsg.)*: Software-Metriken in der Praxis*.
Berlin, Heidelberg : Springer Berlin Heidelberg, S. 105–116
ISBN 978-3-642-88195-4

Montenegro 1999 Montenegro, Sergio, 1999.
Sichere und fehlertolerante Steuerungen: Entwicklung sicherheitsrelevanter Systeme.
München, Wien : Carl Hanser Verlag.
ISBN 3446212353

Montenegro 2000 Montenegro, Sergio (Hrsg.), 2000.
Fehlertoleranz und Sicherheit.
Erfurt

Montenegro 2001 Montenegro, Sergio, 2001.
Methoden und Techniken zur Entwicklung sicherheitsrelevanter Systeme.
In: Dencker, Peter (Hrsg.)*: Ada und Software-Qualität*.
Aachen : Shaker, S. 187–208
ISBN 3-8265-9245-X

Myers et al. 2012 Myers, Glenford J; Badgett, Tom; Sandler, Corey, 2012.
The art of software testing: [now covers testing for usability, smartphone apps, and agile development environments].
3. ed.
Hoboken, NJ : Wiley.
ISBN 9781118031964

Nau 2004 Nau, Matthias, 2004.
Elektrische Temperaturmessung: Mit Thermoelementen und Widerstandsthermometern.
Fulda : JUMO.
messen - regeln - registrieren.
ISBN 978-3-935742-06-1

Norm ANSI/IEEE 610.12 (R2002) 1990 ANSI/IEEE 610.12 (R2002):1990. *IEEE standard glossary of software engineering terminology.*

Norm DIN 55350-11 2008 DIN 55350-11:2008-05. *Begriffe zum Qualitätsmanagement: Ergänzung zu DIN EN ISO 9000:2005*

Norm DIN 66025-1 1983 DIN 66025-1:1983-01. *Programmaufbau für numerisch gesteuerte Arbeitsmaschinen: Allgemeines*

Norm DIN 66025-2 1988 DIN 66025-2:1988-09. *Programmaufbau für numerisch gesteuerte Arbeitsmaschinen: Wegbedingungen und Zusatzfunktionen*

Norm DIN EN ISO 13849-2 2013 DIN EN ISO 13849-2:2013-02. *Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen: Validierung*

Norm DIN EN ISO 9000 2015 DIN EN ISO 9000:2015-11. *Qualitätsmanagementsysteme: Grundlagen und Begriffe*

Norm DIN EN 61131-3 2013 DIN EN 61131-3:2013-12. *Speicherprogrammierbare Steuerungen: Programmiersprachen*

Norm IEEE 1061-1998 2009 IEEE 1061-1998:2009-12. *IEEE Standard for a Software Quality Metrics Methodology.*

Norm ISO/IEC/IEEE 24765 2017 ISO/IEC/IEEE 24765:2017-09. *Systems and Software Engineering: Vocabulary*

Norm ISO/IEC/IEEE 29119-1 2013 ISO/IEC/IEEE 29119-1:2013-09. *Software and Systems Engineering: Software Testing : Concepts and Definitions*

Norm ISO/IEC/IEEE 29119-5 2016 ISO/IEC/IEEE 29119-5:2016-11. *Software and Systems Engineering: Software Testing: Keyword-driven Testing*

Norm ISTQB/GTB Version 2.3 2014 ISTQB/GTB Version 2.3:2014-04. *ISTQB/GTB Standardglossar der Testbegriffe: Deutsch - Englisch*

Norm OPC 40502 2017 OPC 40502:2017-07.
OPC UA for Computerized Numerical Control (CNC) Systems.

Norm VDI 2206 2004 VDI 2206:2004-06.
Entwicklungsmethodik für mechatronische Systeme.

Norm VDI 2221 Blatt 1 2019 VDI 2221 Blatt 1:2019-11.
Entwicklung technischer Produkte und Systeme: Modell der Produktentwicklung

Norm VDI 3633 Blatt 1 2014 VDI 3633 Blatt 1:2014-12.
Simulation von Logistik-, Materialfluss und Produktionssystemen: Grundlagen

Norm VDI 4499 Blatt 1 2008 VDI 4499 Blatt 1:2008-02.
Digitale Fabrik: Grundlagen

Norm VDI 4499 Blatt 2 2011 VDI 4499 Blatt 2:2011-05.
Digitale Fabrik: Digitaler Fabrikbetrieb

Norm VDI/VDE 3681 2005 VDI/VDE 3681:2005-10.
Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik.

Norm VDI/VDE 3693 Blatt 1 2016 VDI/VDE 3693 Blatt 1:2016-08.
Virtuelle Inbetriebnahme: Modellarten und Glossar

Norm VDI/VDE 3694 2014 VDI/VDE 3694:2014-04.
Lastenheft/Pflichtenheft für den Einsatz von Automatisierungssystemen.

Norm VDI/VDE 3695 Blatt 3 2010 VDI/VDE 3695 Blatt 3:2010-12.
Engineering von Anlagen - Evaluieren und optimieren des Engineerings: Themenfeld Methoden

Obstbaum et al. 2017 Obstbaum, Meri; Wurstbauer, Ulrich; König, Christoph; Wagner, Thomas; Kübler, Carsten; Fäßler, Victor, 2017.
From a Graph to a Development Cycle: MBSE as an Approach to reduce Development Efforts.
In: *Deutscher Luft- und Raumfahrtkongress*.
Bonn : Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V

Oestreicher 1986 Oestreicher, Thomas, 1986.
Rechnergestützte Projektierung von Steuerungssystemen.
Karlsruhe, Universität Karlsruhe, Institut für Werkzeugmaschinen und Betriebstechnik, Dissertation, 1986

OPC Foundation 2022 OPC Foundation, 2022.
OPC UA Information Models.
Verfügbar unter: https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models
Zugriff am: 06.03.2022

Pieloth 2013 Pieloth, Roman, 2013. Virtuelle Maschine ermöglicht schnelle Inbetriebnahme.
MaschinenMarkt (41)
Verfügbar unter: http://www.maschinenmarkt.vogel.de/virtuelle-maschine-ermoeglicht-schnelle-inbetriebnahme-a-420292/
Zugriff am: 06.03.2022

Pohlmann 2018 Pohlmann, Uwe, 2018.
A model-driven software construction approach for cyber-physical systems.
Paderborn, Universität Paderborn, Dissertation, 2018.
DOI: 10.17619/UNIPB/1-313

Porteck 2018 Porteck, Stefan, 2018. Allzeit bereit: Smartwatches für jeden Anlass.
c't (3), S. 84–89

Prefi 2003 Prefi, Thomas, 2003.
Qualitätsorientierte Unternehmensführung, Zugl.:
Aachen, Techn. Hochsch., Habil.-Schr., 2002.
1. Aufl.
Aachen : P3 - Ingenieurges. für Management u. Organisation.
P3-Schrift Schriftenreihe Qualitätsmanagement 2003,1.
ISBN 3-9806305-5-2

Preiml 2013 Preiml, Harald, 2013.
Heitec: Erfahrungen mit der virtuellen Inbetriebnahme in der Anlagen-Projektierung.
Vaterstetten.
Verfügbar unter: https://www.digital-engineering-magazin.de/heitec-erfahrungen-mit-der-virtuellen-inbetriebnahme-der-anlagen-projektierung
Zugriff am: 01.04.2019

Pritschow 2006 Pritschow, Günter, 2006.
Einführung in die Steuerungstechnik: Mit 388 Bildern und 40 Tabellen.
München [u.a.] : Carl Hanser Verlag.
ISBN 3-446-21422-4

Pritschow et al. 2004 Pritschow, Günter; Röck, Sascha, 2004. “Hardware in the Loop” Simulation of Machine Tools.
CIRP Annals - Manufacturing Technology **53** (1), S. 295–298
DOI: 10.1016/S0007-8506(07)60701-X

Puntel Schmidt 2017 Puntel Schmidt, Philipp, 2017.
Methoden zur simulationsbasierten Absicherung von Steuerungscode fertigungstechnischer Anlagen.
Hamburg, Helmut-Schmidt-Universität, Universität der Bundeswehr; Helmut-Schmidt-Universität, Dissertation, 2017.
http://nbn-resolving.de/urn:nbn:de:gbv:705-opus-31583

Puntigam 2007 Puntigam, Franz, 2007.
Skriptum zu Objektorientierte Programmierung: Wintersemester 2007/2008.
Wien.
Verfügbar unter: http://www.complang.tuwien.ac.at/franz/objektorientiert/skript07-1seitig.pdf

Reimann 2017 Reimann, Guido, 2017.
IT-Report 2017: Simulationswerkzeuge auf dem Vormarsch.
Frankfurt am Main.
Verfügbar unter: https://sud.vdma.org/viewer/-/v2article/render/18302246
Zugriff am: 04.03.2019

Reiss 2014 Reiss, Steven P., 2014.
The Challenge of Helping the Programmer during Debugging.
In: *Proceedings of 2014 Second IEEE Working Conference on Software Visualization,*
29.-30. September, Victoria, BC, Kanada, S. 112–116

Renieres et al. 2003 Renieres, Manos; Reiss, Steven. P., 2003.
Fault localization with nearest neighbor queries.
In: *Proceedings of 18th IEEE International Conference on Automated Software Engineering,*
6-10 Oktober, Montreal, Que., Kanada, S. 30–39

Röck 2007 Röck, Sascha, 2007.
Echtzeitsimulation von Produktionsanlagen mit realen Steuerungselementen.
Stuttgart, Universität Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW), Dissertation, 2007

Rösch 2016 Rösch, Susanne, 2016.
Model-based testing of fault scenarios in production automation, Internationaler Fachverlag für Wissenschaft & Praxis, Dissertation, 2016

Ross 1977 Ross, D. T., 1977. Structured Analysis (SA): A Language for Communicating Ideas.
IEEE Transactions on Software Engineering **SE-3** (1), S. 16–34
DOI: 10.1109/TSE.1977.229900

Royce 1970 Royce, Winston Walker, 1970.
Managing the development of large software systems.
In: *In Procedings of IEEE WESCON*, S. 1–9

Rumpe 2012 Rumpe, Bernhard, 2012.
Agile Modellierung mit UML.
Berlin, Heidelberg : Springer Berlin Heidelberg.
ISBN 978-3-642-22429-4
DOI: 10.1007/978-3-642-22430-0

Schaefer 2012 Schaefer, Ina, 2012. Software Engineering 1: Software- und Systementwurf - Softwarearchitektur, Vorlesung 5 WS 2012/2013.
Vorlesung TU Braunschweig, 22.11.2012
Verfügbar unter: https://www.isf.cs.tu-bs.de/cms/teaching/2012w/se1/VL5.pdf
Zugriff am: 06.03.2022

Scheifele et al. 2007 Scheifele, Dieter; Eger, Ulrich; Röck, Sascha; Sekler, Peter, 2007.
Potentiale der Hardware-in-the-Loop Simulation für Maschinen und Anlagen.
In: Verl, Alexander; Bender, Klaus; Schuhmacher, Walter (Hrsg.)*: Elektrische Automatisierung - Systeme und Komponenten*.
Berlin : VDE-Verl., S. 555–565
ISBN 978-3800730698

Scheifele et al. 2018 Scheifele, Christian; Verl, Alexander; Riedel, Oliver, 2018. Echtzeit-Co-Simulation für die virtuelle Inbetriebnahme: Echtzeitfähige Integrations-Plattform für digitale Zwillinge.
atp magazin (11-12), S. 44–55

Schlechtendahl et al. 2014 Schlechtendahl, Jan; Kretschmer, Felix; Lechler, Armin; Verl, Alexander, 2014. Communication Mechanisms for Cloud based Machine Controls.
Procedia CIRP **17**, S. 830–834
DOI: 10.1016/j.procir.2014.01.074

Schmitt et al. 2015 Schmitt, Robert; Pfeifer, Tilo, 2015.
Qualitätsmanagement.
München : Carl Hanser Verlag GmbH & Co. KG.
ISBN 978-3-446-43432-5
DOI: 10.3139/9783446440821

Schneider Electric Industries SAS 2016 *Inductive proximity sensors: OsiSense XS*, Catalogue. V2.0.
Rueil-Malmaison, Frankreich

Schumann et al. 2011 Schumann, Marco; Schenk, Michael; Schmucker, Ulrich; Saake, Gunter, 2011.
Digital Engineering - Herausforderungen, Ziele und Lösungsansätz.
In: *Tagungsband 14. IFF-Wissenschaftstage,*
28.-30. Juni, Magdeburg, S. 193–199

Seifert 1992 Seifert, Hans-Jörg, 1992.
Modellgestützte Diagnose komplexer Produktionssysteme : ein Beitrag zur Erhöhung der Verfügbarkeit kapitalintensiver Fertigungsanlagen.
Bochum, Ruhr-Universität Bochum, Institut für Automatisierungstechnik, Dissertation, 1992

Seipel 2012 Seipel, Eberhard, 2012.
Virtuelle Blasformanlage beschleunigt Prozesse.
Bonn.
Verfügbar unter: https://www.kautex-group.com/de/ueber-uns/newsletter/k-news-ausgabe-2-2012/virtuelle-blasformanlage-beschleunigt-prozesse.html
Zugriff am: 01.04.2019

Siemens AG 2008 Siemens AG (Hrsg.), 2008.
SINUMERIK 840D sl/840D/840Di sl: Zyklen, 01.01.2008.
Nürnberg
Verfügbar unter: https://cache.industry.siemens.com/dl/files/376/24763376/att_80969/v1/Programmieranleitung_Zyklen_de-DE.pdf
Zugriff am: 06.03.2022

Siemens AG 2013 Siemens AG (Hrsg.), 2013.
SINUMERIK 840D sl / 828D: Messzyklen, 01.03.2013.
Nürnberg
Verfügbar unter: https://cache.industry.siemens.com/dl/files/299/74473299/att_28112/v1/BNMsl_0313_de_de-DE.pdf
Zugriff am: 06.03.2022

Siemens AG 2014 Siemens AG (Hrsg.), 2014.
SINUMERIK 840D sl/828D SINUMERIK Integrate for Engineering Access MyMachine / OPC UA: Inbetriebnahmehandbuch

Simon 2007 Simon, Alexander, 2007.
Steuerungsbasierte NC-Simulation.
In: Denkena, Berend; Brecher, Christian (Hrsg.)*: Ramp-Up/2 - Anlaufoptimierung durch Einsatz virtueller Fertigungssysteme*.
1. Aufl.
Frankfurt am Main : VDMA-Verl., S. 37–53
ISBN 9783816305460

Sommer 2018 Sommer, Philipp (Hrsg.), 2018.
iWindow - Intelligentes Maschinenfenster.
Als Manuskript gedruckt.
Düsseldorf : VDI Verlag.
ISBN 978-3-18-369702-1

Sommerville 2018 Sommerville, Ian, 2018.
Software Engineering.
10., aktualisierte Auflage.
Hallbergmoos : Pearson Deutschland GmbH.
it - informatik.
ISBN 9783868943443

Spiegelberger 2014 Spiegelberger, Bernd, 2014.
Engineering im Kielwasser von Industrie 4.0.
Heidelberg.
Verfügbar unter: https://www.all-electronics.de/engineering-im-kielwasser-von-industrie-4-0/
Zugriff am: 06.03.2022

Spillner et al. 2012 Spillner, Andreas; Linz, Tilo, 2012.
Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester ; Foundation Level nach ISTQB-Standard.
5. überarb. und aktualisierte Aufl.
Heidelberg : dpunkt Verl.
Safari Tech Books Online.
ISBN 978-3-86490-024-2
Verfügbar unter: https://ebookcentral.proquest.com/lib/subhh/detail.action?docID=1049740

Spillner et al. 2014 Spillner, Andreas; Roßner, Thomas; Winter, Mario; Linz, Tilo, 2014.
Praxiswissen Softwaretest: Testmanagement ; Aus- und Weiterbildung zum Certified Tester ; Advanced Level nach ISTQB-Standard.
4. überarbeitete u. erweiterte Auflage.
Heidelberg : dpunkt.
iSQI-Reihe.
ISBN 978-3-86490-052-5
Verfügbar unter: http://gbv.eblib.com/patron/FullRecord.aspx?p=1714253

Statistisches Bundesamt 2021 Statistisches Bundesamt, 2021.
Produktionswert im Maschinenbau in Deutschland nach Sektoren in den Jahren 2019 und 2020.
Hamburg.
Verfügbar unter: https://de.statista.com/statistik/daten/studie/236826/umfrage/produktionswert-im-maschinenbau-in-deutschland-nach-sektoren/
Zugriff am: 12.02.2022

Statistisches Bundesamt et al. 2021 Statistisches Bundesamt; Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA), 2021a.
Produktionswert im deutschen Maschinenbau in den Jahren 1991 bis 2020 und Prognose bis 2022.
Hamburg.
Verfügbar unter: https://de.statista.com/statistik/daten/studie/77231/umfrage/produktionswert-und-umsatz-im-deutschen-maschinenbau-seit-2000/
Zugriff am: 12.02.2022

Stetter et al. 2008 Stetter, Rainer; Erben, Manuel, 2008. Automatisches Testen bei SPS-Steuerungssoftware. *atp-Spezial*, S. 31–34

Stöppler et al. 2005 Stöppler, Guido; Menzel, Thomas; Douglas, Steve, 2005. Hardware-in-the-loop simulation of machine tools and manufacturing systems. *IEE Computing & Control Engineering* **16** (1), S. 10–15 DOI: 10.1049/cce:20050101

Strassacker 1997 Strassacker, Dirk, 1997. *Testumgebung für die Implementierung und Inbetriebnahme eines adaptierbaren Leitsteuerungssystems*, Dissertation. Stuttgart : Springer. ISW Forschung und Praxis, Berichte aus dem Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart 117. ISBN 3540631615 Verfügbar unter: http://dx.doi.org/10.1007/978-3-642-51507-1DOI: 10.1007/978-3-642-51507-1

Sünder et al. 2013 Sünder, Christoph; Vyatkin, Valeriy; Zoitl, Alois, 2013. Formal Verification of Downtimeless System Evolution in Embedded Automation Controllers. *ACM Transactions on Embedded Computing Systems* **12** (1), S. 1–17 DOI: 10.1145/2406336.2406353

Tatsumi 1987 Tatsumi, Keizo, 1987. Test case design support system. In: *Proceedings of ICQC‘87,* 20.-23. Oktober, Tokyo, Japan, S. 615–620

Thaller 2000 Thaller, Georg Erwin, 2000. *Software-Qualität: Der Weg zu Spitzenleistungen in der Software-Entwicklung*. Berlin : VDE-Verl. EDV-Praxis. ISBN 3800724944

Thaller 2002 Thaller, Georg Erwin, 2002. *Software-Test: Verifikation und Validation.* 2., aktualisierte und erw. Aufl. Hannover : Heise. ISBN 3882291982

Ulewicz 2018
Ulewicz, Sebastian, 2018.
Test coverage assessment for semi-automatic system testing and regression testing support in production automation.
München, Technische Universität München, Dissertation, 2018

Ulrich et al. 1995
Ulrich, Hans; Probst, Gilbert J. B., 1995.
Anleitung zum ganzheitlichen Denken und Handeln: Ein Brevier für Führungskräfte.
4., unveränd. Aufl.
Bern : Haupt.
ISBN 978-3258051826

Utting et al. 2007
Utting, Mark; Legeard, Bruno, 2007.
Practical model-based testing: A tools approach.
Amsterdam : Morgan Kaufmann/Elsevier.
Safari Tech Books Online.
ISBN 9780123725011
Verfügbar unter: http://proquest.safaribooksonline.com/9780123725011

Verband Deutscher Maschinen- und Anlagenbau e.V. 2004
Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA) (Hrsg.), 2004.
Baukastenbasiertes Engineering mit Föderal: Ein Leitfaden für Maschinen- und Anlagenbauer : VDMA Verl.
ISBN 3-8163-0478-8

Verband Deutscher Maschinen- und Anlagenbau e.V. 2010
Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA) (Hrsg.), 2010.
Aquimo: Adaptierbares Moldellierungswerkzeug und Qualifizierungsprogramm für den Aufbau firmenspezifischer mechatronischer Engineeringprozesse ; ein Leitfaden für Maschinen- und Anlagenbauer.
Frankfurt, M. : VDMA Verlag GmbH.
ISBN 9783816305897

Verband Deutscher Maschinen- und Anlagenbau e.V. 2015
Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA) (Hrsg.), 2015.
MEPROMA: Anforderungen und Methoden im mechatronischen Engineering : Leitfaden für Anwender im Maschinen-und Anlagenbau.
Frankfurt am Main : VDMA Verlag GmbH.
ISBN 9783816306788

Verband Deutscher Maschinen- und Anlagenbau e.V. 2017
Simulation und Visualisierung: Erste Ergebnisse der Trendumfrage 2017.
Frankfurt am Main

Verband Deutscher Maschinen- und Anlagenbau e.V. 2020
Verband Deutscher Maschinen- und Anlagenbau e.V. (VDMA) (Hrsg.), 2020.
Leitfaden Virtuelle Inbetriebnahme: Handlungsempfehlungen zum wirtschaftlichen Einstieg.
Frankfurt am Main : VDMA Verlag GmbH

Verein zur Weiterentwicklung des V-Modell XT e.V. 2006
Verein zur Weiterentwicklung des V-Modell XT e.V., 2006.
V-Modell XT: Das deutsche Referenzmodell für Systementwicklungsprojekte, Version 2.3.
Verfügbar unter: http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.3/V-Modell-XT-Gesamt.pdf
Zugriff am: 06.03.2022

Vogel Communications Group 2016
Vogel Communications Group, 2016.
Qualität von SPS-Programmen in Minuten checken, elektrotechnik Automatisierung.
Verfügbar unter: https://www.elektrotechnik.vogel.de/qualitaet-von-sps-programmen-in-minuten-checken-a-522834/
Zugriff am: 06.03.2022

Vogel-Heuser et al. 2014
Vogel-Heuser, Birgit; Diedrich, Christian; Fay, Alexander; Jeschke, Sabine; Kowalewski, Stefan; Wollschlaeger, Martin; Göhner, Peter, 2014. Challenges for Software Engineering in Automation.
Journal of Software Engineering and Applications **07** (05), S. 440–451
DOI: 10.4236/jsea.2014.75041

Vogel-Heuser et al. 2017
Vogel-Heuser, Birgit; Fischer, Juliane; Feldmann, Stefan; Ulewicz, Sebastian; Rösch, Susanne, 2017. Modularity and architecture of PLC-based software for automated production Systems: An analysis in industrial companies.
Journal of Systems and Software **131**, S. 35–62
DOI: 10.1016/j.jss.2017.05.051

Vollmer 1994
Vollmer, Patrick, 1994. Software-Test für speicherprogrammierbare Steuerungen.
Werkstatt und Betrieb **127** (6), S. 463–464

Vyatkin et al. 2009 Vyatkin, V; Hanisch, H.-M; Pang, Cheng; Yang, Chia-Han, 2009. Closed-Loop Modeling in Future Automation System Engineering and Validation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **39** (1), S. 17–28 DOI: 10.1109/TSMCC.2008.2005785

Wälde 2002 Wälde, Klaus, 2002. *Sicherstellung der Softwarequalität von Anwendungsmodulen und Systemplattformen in offenen Steuerungssystemen*, Dissertation. Stuttgart : Jost-Jetter. ISW-Forschung und -Praxis Nr. 143. ISBN 3931388891

Weck et al. 2005 Weck, Manfred; Brecher, Christian, 2005. *Werkzeugmaschinen 1: Maschinenarten und Anwendungsbereiche.* 6., neu bearb. Aufl. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg. VDI-Buch. ISBN 978-3-540-22504-1 Verfügbar unter: http://dx.doi.org/10.1007/978-3-540-28085-9DOI: 10.1007/3-540-28085-5

Weck et al. 2006 Weck, Manfred; Brecher, Christian, 2006a. *Werkzeugmaschinen 3: Mechatronische Systeme, Vorschubantriebe, Prozessdiagnose.* 6., neu bearbeitete Auflage. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg. VDI-Buch. ISBN 978-3-540-22506-5 Verfügbar unter: http://dx.doi.org/10.1007/978-3-540-32506-2DOI: 10.1007/3-540-32506-9

Weck et al. 2006 Weck, Manfred; Brecher, Christian, 2006b. *Werkzeugmaschinen 4: Automatisierung von Maschinen und Anlagen.* 6 Auflage. Berlin [u.a.] : Springer. ISBN 3-540-22507-2

Weidauer 2013 Weidauer, Jens, 2013.
Elektrische Antriebstechnik: Grundlagen, Auslegung, Anwendungen, Lösungen.
3. Aufl.
Erlangen : Publicis Publ.
ISBN 978-3-89578-431-6
Verfügbar unter: http://gbv.eblib.com/patron/FullRecord.aspx?p=1221203

Weißenberger et al. 2015 Weißenberger, Benedikt; Flad, Stefan; Chen, Xinyu; Rösch, Susanne; Voigt, Tobias; Vogel-Heuser, Birgit, 2015.
Model driven engineering of manufacturing execution systems using a formal specification.
In: *Proceedings of 2015 ETFA,*
08.-11. September 2015, Luxembourg, Luxembourg, S. 1–8

Weyrich et al. 2018 Weyrich, Michael; Zeller, Andreas; Maschler, Benjamin; Jung, Tobias, Krause, Jan; Krüning, Kai; Köllner, Christian; Siegrist, Daniel, 2018. Test vernetzter Systeme für die Industrie 4.0: Auszug aus dem Statusreport des VDI/VDE-Fachausschusses 7.25.
atp edition **60** (03), S. 26–33

Witte 2016 Witte, Frank, 2016.
Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung.
1. Auflage.
Wiesbaden : Springer Vieweg.
ISBN 9783658099633
Verfügbar unter: http://dx.doi.org/10.1007/978-3-658-09964-0DOI: 10.1007/978-3-658-09964-0

Wittel et al. 2013 Wittel, Herbert; Muhs, Dieter; Jannasch, Dieter; Voßiek, Joachim, 2013.
Roloff/Matek Maschinenelemente.
Wiesbaden : Springer Fachmedien Wiesbaden.
ISBN 978-3-658-02326-3
DOI: 10.1007/978-3-658-02327-0

Wünsch 2008 Wünsch, Georg, 2008.
Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme.
München, Technische Universität München, Lehrstuhl für Werkzeugmaschinen und Fertigungstechnik, Dissertation, 2008

Zäh et al. 2005 Zäh, Michael F; Wünsch, Georg, 2005. Schnelle Inbetriebnahme von Produktionssystemen: Qualitätssicherung von automatisierten Maschinen durch Simulation. *wt Werkstattstechnik online* **95** (09), S. 699–704

Zäh et al. 2006 Zäh, Michael F; Wünsch, Georg; Hensel, Thomas; Lindworsky, Alexander, 2006a. Feldstudie - Virtuelle Inbetriebnahme. *wt Werkstattstechnik online* **96** (10), S. 767–771

Zäh et al. 2006 Zäh, Michael F; Wünsch, Georg; Hensel, Thomas; Lindworsky, Alexander, 2006b. Nutzen der virtuellen Inbetriebnahme: Ein Experiment. *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* **101** (10), S. 595–599 DOI: 10.3139/104.101070

Zeller 2009 Zeller, Andreas, 2009. *Why programs fail: A guide to systematic debugging.* 2. ed. Amsterdam : Elsevier. ISBN 978-3-89864-620-8 Verfügbar unter: http://www.sciencedirect.com/science/book/9780123745156

Zetzsche 2019 Zetzsche, Michael, 2019. Simulation. In: *3. Stuttgarter Innovationstage,* 12.-13. Februar 2019, Stuttgart, S. 148–171

Universität Stuttgart

Seit Jahrzehnten wird daran entwickelt und geforscht, den Automatisierungsgrad von Fertigungssystemen zu erhöhen. Im Trend steigender und intelligenterer Automatisierung ergibt sich der Effekt von deutlich mehr softwarebasierten Lösungen. Der höheren Dynamik beim Entwickeln und Nachbessern an der Software steht ein steigender Aufwand zur Qualitätssicherung gegenüber: Komplexere Testabläufe und größere Datenmengen bedeuten einen gestiegenen Testumfang bei gleichzeitig höherer Frequenz an Testdurchführungen. Diese Tendenzen machen eine hohe Testabdeckung und eine hohe Testtiefe mittels manueller Testprozesse unmöglich.

In dieser Arbeit wird eine ganzheitliche Testautomatisierung entwickelt, um den Testprozess für Fertigungssysteme zu verbessern. Zielgröße ist die Schnittmenge aus hoher Aussagekraft der Testergebnisse und geringem Aufwand bei der Anwendung der neuen Testmethodik. Anhand zweier Validierungsszenarien werden Verbesserungen zum bisherigen Testprozess ermittelt. Quantitativ verkürzt die Methodik die Zeitdauer bei der Testdurchführung mit einem Testwerkzeug um ca. 45 %. In einem Systemtestszenario wurde der Gesamtaufwand mit Testentwurf und -abschluss um ca. 95 % gesenkt.

Fraunhofer VERLAG

ISBN 978-3-8396-1817-2

9 783839 618172