Schlussbericht

zu dem IGF-Vorhaben

Steigerung der Zuverlässigkeit von Maschinen und Anlagen durch automatisiertes Testen von Fehlerbehandlungsroutinen in der Steuerungssoftware (ZuMaTra)

der Forschungsstelle
Lehrstuhl für Automatisierung und Informationssysteme, Technische Universität München

Das IGF-Vorhaben 16906 N der Forschungsvereinigung Elektrotechnik beim ZVEI e. V. wurde über die

im Rahmen des Programms zur Förderung der Industriellen Gemeinschaftsforschung (IGF) vom

aufgrund eines Beschlusses des Deutschen Bundestages gefördert.

Garching 30.01.2015		Prof. Dr.-Ing. Birgit Vogel-Heuser
Ort, Datum		Name und Unterschrift des/der Projektleiter(s) an der/den Forschungsstelle(n)

Steigerung der Zuverlässigkeit von Maschinen und Anlagen durch automatisiertes Testen von Fehlerbehandlungsroutinen in der Steuerungssoftware (ZuMaTra)

Kurzfassung:

Die Zuverlässigkeit von Maschinen wird wesentlich von deren Steuerungssoftware bestimmt, da diese Ausnahmesituationen, die z.B. in Folge von Sensor-/Aktorausfällen eintreten können, behandelt. Die fehlerbehandelnden Softwareanteile und insbesondere Anteile zur Erkennung von Fehlern bilden dabei einen Großteil der Steuerungssoftware, werden jedoch aufgrund mangelnder Zeit bzw. fehlender Werkzeuge und Verfahren oft nur unzureichend getestet. Ob diese für die Zuverlässigkeit der Maschine oder Anlage notwendigen Funktionen korrekt funktionieren, wird dann erst bei Eintritt der jeweiligen Ausnahmesituationen im laufenden Betrieb festgestellt. Um hohe Ausfallkosten und Schäden infolge fehlerhafter Ausnahmebehandlungen zu vermeiden, ist daher ein Funktionsnachweis bereits vor oder während der Inbetriebnahme anzustreben. Eine Lösung hierfür ist ein Verfahren, welches die notwendigen Testfälle zur Überprüfung der Ausnahmebehandlungen automatisiert erzeugt. Ziel des Projekts war daher die Erforschung eines Verfahrens, welches mit minimalem Engineering-Aufwand und maximalem Automatisierungsgrad eine effektive Überprüfung der notwendigen Ausnahmebehandlungsroutinen in der Steuerungssoftware ermöglicht. Um das Verhalten des Steuerungscodes auf Sensor-/Aktorausfälle testen zu können, wurde ein neuer Ansatz für die Fehlerinjektion in den Steuerungscode entwickelt und zusammen mit existierenden Techniken zur automatischen Generierung und Ausführung von Testfällen auf die Software-Entwicklung in der Fertigungstechnik angewandt. Fehlerinjektion wird im Projekt definiert als das absichtliche Herbeiführen eines Fehlers in einem System, um dessen Reaktion darauf auszuwerten. Bislang existierten im Umfeld der Software-Entwicklung für Speicherprogrammierbare Steuerungen (SPS) nur unzureichende Methoden der Fehlerinjektion.
Um den Aufwand der Testerstellung zu minimieren wurde ein modellbasierter Ansatz gewählt. Nach einer Anforderungsanalyse wurde das Format eines Weg-Zeit-Diagramms als Basis für den Ansatz gewählt, für die Modellierung von fertigungstechnischen Anlagen weiterentwickelt und für die Testfallgenerierung optimiert. Basierend auf der Weg-Zeit-Diagramm-Notation wurde ein Algorithmus entwickelt, der die wichtigsten Fehlerszenarien in der Fertigungstechnik durch Manipulation von Sensoren abdeckt und eine Testausführung durch software-implementierte Fehlerinjektion für SPSen möglich macht. Die Ergebnisse der einzelnen Testläufe dienen zugleich als Dokumentation und Nachweis der korrekten Funktion der Software. Der Ansatz für die Testfallgenerierung und –automatisierung konnte innerhalb eines Workshops mit Hilfe eines Funktionsmusters in Form eines Editors und Testfallgenerierungsalgorithmus mit den Mitgliedern des Projektausschusses durchgeführt und evaluiert werden. Dementsprechend konnte nicht nur der Nachweis der Machbarkeit, sondern auch der Nachweis der Anwendbarkeit des Ansatzes für die Domäne der Fertigungstechnik erbracht werden. Das Forschungsziel wurde nachweislich erreicht.

Berichtsumfang:	65 S., 38 Abb., 4 Tab., 41 Lit.
Forschungsvereinigung:	ZVEI - Zentralverband Elektrotechnik- und Elektronikindustrie e. V
Forschungsstelle:	Lehrstuhl für Automatisierung und Informationssysteme, Technische Universität München
Leiterin:	Prof. Dr.-Ing. Birgit Vogel-Heuser
Bearbeiterin und Verfasserin:	Susanne Rösch

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung / Motivation

Die Software nimmt im Maschinen- und Anlagenbau eine zentrale Rolle ein und stellt einen wesentlichen Teil der Funktionserbringung dar. In zunehmendem Maße werden mehrere verteilte Softwarekomponenten zur Umsetzung einer Funktion benötigt, was zu einer steigenden Komplexität führt. Die Beherrschung der Komplexität dieser Funktionsübernahme der Software innerhalb des Gesamtsystems erfordert die Integration der softwarerelevanten Aspekte während der Systemplanungs- und entwicklungsphase. In [Mo00] wird gezeigt, dass Ausfälle in fehlertoleranten Systemen lediglich in 8 % aller Fälle auf Hardwareschäden, hingegen 65 % auf Softwarefehler zurückzuführen sind.

Die Entwicklung qualitativ hochwertiger Software ist heute mangels Werkzeugen zur durchgängigen Entwicklung enorm aufwendig, erfordert viel Zeit und zieht dementsprechend hohe Kosten nach sich. Durch den immateriellen Charakter von Steuerungssoftware und die nur schwer sichtbare Entwicklungskomplexität der Software fehlt häufig die Bereitschaft seitens des Kunden, die zur Gewährleistung der Softwarequalität notwendigen Anstrengungen im Engineering, zu bezahlen, wodurch der Kostendruck bei den Herstellern der Maschinen und Anlagen steigt. Nichtsdestotrotz sind die Kunden nicht bereit Einbußen was die Zuverlässigkeit der Maschinen und Anlagen betrifft hinzunehmen.

Zuverlässigkeit setzt sich u.a. aus den Aspekten Reife, Fehlertoleranz und Wiederherstellbarkeit zusammen (s. ISO/IEC 9126 bzw. DIN 66272). Diese Aspekte sind dabei wie folgt definiert:

- Reife: Geringe Versagenshäufigkeit durch Fehlerzustände
- Fehlertoleranz: Fähigkeit, ein spezifiziertes Leistungsniveau bei Softwarefehlern oder Nichteinhaltung ihrer spezifizierten Schnittstelle einzuhalten,
- Wiederherstellbarkeit: Fähigkeit, bei einem Versagen das Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen.

Die Zuverlässigkeit von Maschinen und Anlagen wird wesentlich von deren Steuerungssoftware bestimmt, da diese Ausnahmesituationen, die z.B. in Folge von Sensor-/Aktorausfällen eintreten können, behandelt. Die fehlerbehandelnden Softwareanteile bilden einen Großteil der Steuerungssoftware, werden jedoch aufgrund mangelnder Zeit bzw. fehlender Werkzeuge und Verfahren oft nur unzureichend getestet. Fehlerbehandlungsroutinen werden zwar mit hohem Aufwand erstellt, jedoch oft nicht explizit getestet, da die „Gut-Funktion“ der Maschine/Anlage den Kern der Abnahme durch den Kunden darstellt. Die „Gut-Funktionen“ werden daher kontinuierlich im Rahmen der Inbetriebnahme und in Vorbereitung eines Abnahmetests geprüft. Darüber hinaus bleibt in der Inbetriebnahme oft nicht die Zeit, Fehlerbehandlungsroutinen im notwendigen Umfang zu testen. Zur Sicherstellung der Systemzuverlässigkeit müssten jedoch auch diese Softwareanteile getestet werden, die im Falle eines möglichen Hardware-Ausfalls greifen sollten. Aufgrund des insgesamt vorherrschenden Kostendrucks, insbesondere für die Softwareanteile von Maschinen und Anlagen, ist es jedoch nicht möglich, erweiterte manuelle Testverfahren einzusetzen. Umfangreichere Tests können nur dann kostendeckend durchgeführt werden, wenn diese im Wesentlichen automatisch ablaufen können. Ob diese für die Zuverlässigkeit der Maschine oder Anlage notwendigen Funktionen korrekt funktionieren, wird dann erst bei Eintritt der jeweiligen Ausnahmesituationen im laufenden Betrieb festgestellt. Um hohe Ausfallkosten und Schäden infolge fehlerhafter Ausnahmebehandlungen zu vermeiden, ist jedoch ein Funktionsnachweis bereits vor oder während der Inbetriebnahme anzustreben. Eine Lösung hierfür ist ein Verfahren, welches die notwendigen Testfälle zur Überprüfung der Ausnahmebehandlungen automatisiert erzeugt. Die somit generierten Testfälle müssen zur effizienten Abarbeitung in ihrer Anzahl und Komplexität beschränkt werden. Ziel dieses Vorhabens ist daher die Erforschung eines Verfahrens, welches mit minimalem Engineering-Aufwand und maximalem Automatisierungsgrad eine effektive Überprüfung der

notwendigen Ausnahmebehandlungsroutinen in der Steuerungssoftware ermöglicht. Um das Verhalten des Steuerungscodes auf Sensor-/Aktorausfälle testen zu können, sollen neue Ansätze der Fehlerinjektion in den Steuerungscode entwickelt und zusammen mit existierenden Techniken zur automatischen Generierung und Ausführung von Testfällen auf die Software-Entwicklung im Maschinen- und Anlagenbau angewendet werden. Bislang existieren im Umfeld der Software-Entwicklung für Speicherprogrammierbare Steuerungen (SPS) keine passenden Methoden der Fehlerinjektion für Sensor-/Aktorausfälle. Da die Zuverlässigkeit ihrer Produkte für klein- und mittelständische Maschinen- und Anlagenbauer einen entscheidenden Wettbewerbsfaktor darstellt, der heute noch notwendige zeitliche Aufwand für die Qualitätssicherung jedoch kaum tragbar ist, soll bei diesem Projekt insbesondere die praktische Anwendbarkeit für KMU berücksichtigt werden. Die Ergebnisse sollen außerdem Möglichkeiten für weitere Werkzeugentwicklungen schaffen, die eine langfristige Basis für eine Steigerung der Zuverlässigkeit und Sicherheit von Maschinen und Anlagen bieten und somit einen Wettbewerbsvorteil ermöglichen.

1.2 Projekt ZuMaTra

1.2.1 Projektziele

Ziel des Projekts „Steigerung der Zuverlässigkeit von Maschinen und Anlagen durch automatisiertes Testen von Fehlerbehandlungsroutinen in der Steuerungssoftware“ (ZuMaTra) ist eine in der Praxis anwendbare und wissenschaftlich fundierte Methode bereitzustellen, wodurch eine (teil-)automatische Überprüfung der korrekten Implementierung bzw. Funktion der Behandlung von Ausnahmesituationen ermöglicht wird.

Um die Praxistauglichkeit des Ansatzes sicherzustellen, wurden zu Beginn des Projekts zum einen eine ausführliche Anforderungsanalyse unter den Unternehmen durchgeführt zum anderen einige Anwendungsbeispiele aus den Unternehmen auf weitere Anforderungen und Randbedingungen untersucht. Darüber hinaus wurden bestehende Ansätze auf deren Anwendbarkeit hin überprüft.

Um die Herausforderungen im Kontext einer Überprüfung von Fehlerbehandlungsroutinen der Steuerungssoftware im Maschinen-/Anlagenbau zu beherrschen, wurden im Projekt zentrale, dafür notwendige Aspekte, wie die modellbasierte Testfallgenerierung und die statische Codeanalyse, in einem Ansatz vereint.

Dazu sollte auf Basis der ermittelten Anforderungen ein Modellierungsansatz entwickelt werden, der eine automatische Testfallgenerierung basierend auf ebenfalls ermittelten und im Projekt definierten Fehleroperatoren ermöglicht. Diese Fehleroperatoren sind verknüpft mit Mustern in der Verhaltensmodellierung der Software einer Maschine (z.B. steigende Flanke eines binären Sensors) und erlauben so die Identifikation von potenziellen Fehlerszenarien, die durch einen Testfall zu überprüfen sind. Darauf aufbauend wurde ein Verfahren zur automatischen Generierung sowie Ausführung identifizierter Testfälle entwickelt.

Die bei der Anforderungsermittlung aufgedeckten Randbedingungen sollten außerdem die Überprüfung von Fehlerbehandlungen durch statische Codeanalyse ermöglichen.

In dem hier vorgestellten Ansatz ist die Überprüfung von IEC 61131-3 Steuerungssoftware [IEC03] das grundsätzliche Anwendungsfeld. Die untersuchte Ausführung der vorher erstellten Testfälle muss in Interaktion mit der Steuerung stehen und unter Echtzeitbedingungen abgearbeitet werden. Um die Fehlerbehandlungsroutinen des Steuerungscodes überprüfen zu können, werden Fehler automatisch in das System injiziert. Dabei kann die Interaktion abhängig von einem jeweiligen Einsatzszenario und der Kritikalität der Anlage gegenüber einer realen oder simulierten Maschine stattfinden. Um die Robustheit fehlertoleranter Systeme testen zu können, müssen potentiell auftretende Hardware-Fehler (Ausnahmesituationen) in Tests erzwungen werden. Die aktuell existierenden Ansätze der Fehlerinjektion werden für das Erzeugen von Speicherfehlern verwendet, um während des Testdesigns berücksichtigte Fehler zu erzwingen. In ZuMaTra werden diese Technik in Testfällen verwendet, die beispielsweise auf

Fehleranalysen (bspw. FMEA [FME09]) oder anderen geeigneten Notationen für die Anwendungsbeispiele basieren.

Die Fehlerinjektion soll gegenüber einer simulierten oder auch realen Anlage durchgeführt werden können. Je nach Art der generierten Testfälle setzen diese ähnliche oder unterschiedliche Startzustände des Gesamtsystems voraus.

Um das Potential des Ansatzes aufzuzeigen, wurde ein Funktionsmuster, welches sowohl die Codeanalyse als auch der Test von Fehlerbehandlungsroutinen unterstützt, umgesetzt. Das Funktionsmuster konnte für eine ausführliche Evaluierung der Ergebnisse dienen.

1.2.2 Innovativer Beitrag der angestrebten Forschungsergebnisse

Bislang existieren keine Ansätze oder Werkzeuge, die es erlauben die Fehlerbehandlungsroutinen von Maschinen und Anlagen automatisiert und damit kostengünstig zu testen. Der Einsatz der angestrebten Forschungsergebnisse ermöglicht letztendlich eine effizientere Entwicklung, da durch die Testautomatisierung die Effizienz bei sowohl der Testfallerstellung als auch der Testdurchführung erheblich erhöht wird. Die bislang erst während des Betriebs aufgetretenen Softwarefehler werden somit schon während der Entwicklung erkannt und korrigiert. Die Entwicklung eines solchen Ansatzes bietet für kleine und mittelständische Maschinen- und Anlagenbauer erstmalig die Möglichkeit, die Steuerungssoftware ihrer Anlage auch für Ausfallsituationen mit geringem Mehraufwand zu testen und damit die Zuverlässigkeit ihrer Maschinen und Anlagen erheblich zu steigern. Dies hat unmittelbar zur Folge, dass die Inbetriebnahmezeiten verkürzt und der Zuverlässigkeitsnachweise strukturiert dokumentiert werden kann. Anbieter von Software-Werkzeugen, insbesondere Hersteller von Programmierumgebungen für Steuerungen, können die erforschten Verfahren und Algorithmen für die Entwicklung innovativer Produkte heranziehen.

Darüber hinaus können Dienstleister und Ingenieurbüros (größtenteils kleine Unternehmen), die sich auf die Erstellung von Steuerungssoftware spezialisiert haben, ihr Angebot erweitern bzw. die Qualität der angebotenen Dienstleistungen erheblich steigern. Hierdurch können sie sich gegenüber der Konkurrenz aus Ländern mit geringerem Lohnniveau deutlich absetzen. Für eine praxisnahe und erfolgsversprechende Umsetzung bedarf es einer Betrachtung aus unterschiedlichen Domänen. Die Konstellation des projektbegleitenden Ausschusses (PA) aus Unternehmen des Maschinen- und Anlagenbaus, Softwareentwicklern, Beratungsunternehmen im Maschinen- und Anlagenbaum, Gerätetechniker und Betreiber (Gerätehersteller verwenden Maschinen um ihre Produkte herzustellen) stellt die Ausgangsbasis für eine lückenlose Erarbeitung praxistauglicher Ergebnisse unter Berücksichtigung aller Anspruchsgruppen dar.

1.2.3 Zusammensetzung des Projektbegleitenden Ausschusses (PA)

Der projektbegleitende Ausschuss (PA) setzt sich aus Unternehmen aller betreffenden Anspruchsgruppen zusammen:

- Anbieter von Software-Werkzeugen und Beratung im Bereich Software
- Anbieter von Steuerungs-, Antriebs,- und Gerätetechnik
- Unternehmen des Maschinen- und Anlagenbaus mit Steuerungsprogrammierung
- Maschinenbetreiber als Kunde der Maschinen- und Anlagenbauer

Hierbei treten die nicht KMU in unterschiedlichen Rollen auf: Festo, Phoenix Contact und Bosch sind zum einen Anbieter von Steuerungskomponenten, produzieren diese aber auch selbst mit eigenen Maschinen, die von einer internen Maschinenbauabteilung entwickelt werden.

Der deutsche Automatisierungsmarkt wird heute im Wesentlichen durch einige wenige Hersteller bestimmt. Insbesondere die Produkte der Firma Siemens dominieren. Jedoch konnten in den letzten Jahren auch kleinere Unternehmen einen starken Anstieg ihrer Marktanteile

verzeichnen. Dies trifft insbesondere auch auf Anbieter von Steuerungskomponenten wie bspw. Beckhoff, Phoenix Contact oder Schneider zu. Viele dieser Anbieter stützen sich auf das Programmiersystem CODESYS der Firma 3S-Smart-Software-Solutions. Durch die Einbeziehung der Firmen 3S-Smart-Software-Solutions und anderen Anbietern wie z.B. Phoenix-Contact (die in Deutschland ebenso einen bedeutenden Marktanteil haben) wird in diesem Projekt eine hohe Abdeckung hinsichtlich der für Deutschland relevanten Anbieter von Programmierumgebungen für Steuerungen erreicht.

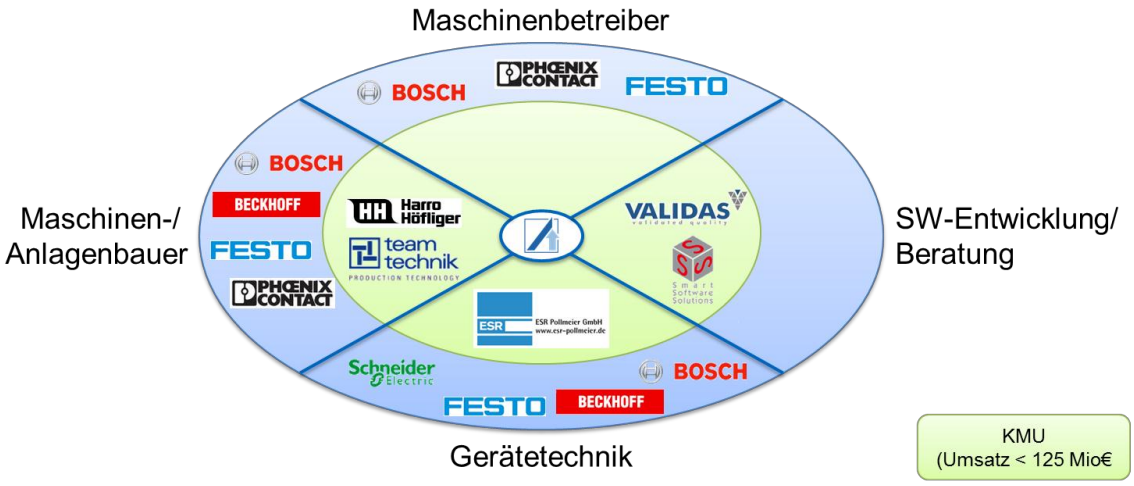


Abbildung 1: Struktur des projektbegleitenden Ausschusses

1.2.4 Arbeitspakete (AP)

Die Arbeitspakete wurden zur Erreichung der unter den Forschungszielen formulierten Zielstellungen eingeteilt:

AP 0: Vorgehensweise und Vorrecherchen im Projekt

Durch eine ausführliche Recherche und Bewertung existierender Technologien, Teillösungen und zu berücksichtigender Standards wurde die Grundlage für die weitere Ausarbeitung der Zielstellung und der weiteren Arbeitsschritte gelegt. Bei der Bearbeitung der einzelnen Arbeitsschritte wurde ein besonderer Fokus auf die praxisnahe Ausrichtung der Zielsetzung gelegt. Dabei wurden in den regelmäßigen Projekttreffen die Anforderungen und spezifischen Anwendungsfälle der Unternehmen im PA abgefragt und gemeinsam diskutiert. Mit dem Ziel den Aufwand der Unternehmen (KMU im speziellen) für an das Forschungsprojekt anknüpfende Entwicklungen niedrig zu halten, wurden die im wissenschaftlichen Umfeld analysierten existierenden oder neu entwickelten Methoden praxisgerecht aufbereitet. Um aufbauende Entwicklungen zu fördern, verbreitet die Forschungsstelle die Forschungsergebnisse praxisbezogen in Veröffentlichungen.

AP 1: Modellierungsverfahren für Komponenten und deren Ausfälle im Kontext des Steuerungscodes

Grundlage des AP 1 stellten die in AP 0 definierten Anforderungen und Randbedingungen dar. Einer Untersuchung, welche Notationen bei den Unternehmen ihren Einsatz finden und für die Testfallgenerierung geeignet sind, ergab, dass das Weg-Zeit-Diagramm grundsätzlich geeignet ist. Dies wurde entsprechend weiterentwickelt und angepasst um den Anforderungen voll zu genügen. Als Ergänzung zum Weg-Zeit-Diagramm wurde die „Failure Mode and Effects Analysis“ (FMEA), welche eine Betrachtung der Systemkomponenten mit den ihnen zuzuord-

nenden Ausfallwahrscheinlichkeiten und Kritikalitäten ermöglicht, gewählt. Um den Testaufwand zu reduzieren, kann so die Menge der zu erzeugenden Testfälle anhand dieser Parameter skaliert werden.

Es wurde erfolgreich ein Modellierungsansatz entwickelt.

Personaleinsatz HPA A: 6 PM

AP 2: Verfahren zur Extraktion eines sprachneutralen Kontrollflussmodells

Ziel dieses Arbeitspakets war die Erforschung eines Verfahrens welches den implizit z.B. durch Schleifen, Sprünge und Alternativ-Entscheidungen (IF-THEN-ELSE, CASE) definierten Kontrollfluss aus einem gegebenen Steuerungsprogramm extrahieren und in ein sprachneutrales Automatenmodell überführen kann. Für die Untersuchung mittels statischer Codeanalanalyse ist die einheitliche Repräsentation der unterschiedlichen IEC 61131-3 Programmiersprachen eine wichtige Herausforderung. Eine einheitliche Darstellung wurde insbesondere für die Sprachen Strukturierter Text (ST), Ablaufsprache (AS) (siehe Anhang CD: Meilenstein 2) und außerhalb des Projekts in einer assoziierten Bachelorarbeit für Funktionsblockdiagramme (FBD) [Ta14] untersucht.

Das Verfahren konnte erfolgreich entwickelt werden.

Personaleinsatz HPA A: 5 PM; HPA B: 2 PM

AP 3: Verfahren zur statischen Code-Analyse

Ziel dieses Arbeitspakets war die Erforschung eines statischen Code-Analyseverfahrens mit dem die in AP 0 identifizierten Regeln für die Überprüfung der Steuerungssoftware überprüft werden können. Hierfür wurden einfache Templates für die Spezifikation der Regeln und ein Algorithmus zur Überprüfung der spezifizierten Regeln am Kontrollfluss entwickelt.

Das Verfahren konnte erfolgreich für die Überprüfung von Codierrichtlinien, nicht jedoch für die Testfallgenerierung entwickelt werden.

Personaleinsatz HPA A: 4 PM; HPA B: 1 PM

AP 4: Automatisches Testfallgenerierungsverfahren

Ziel dieses Arbeitspakets war die Anwendung bzw. Adaption von Testfallgenerierungsverfahren. Das Generierungsverfahren entspricht allen nach AP 0 definierten relevanten Situationen, in denen ein Komponentenausfall ein alternatives Verhalten auslösen kann. Ein Testfall oder mehrere Testfälle liefern durch deren Ausführung die Möglichkeit einer Beurteilung, ob die Fehlererkennung und Fehlerbehandlung korrekt funktioniert.

Das automatische Testfallgenerierungsverfahren konnte erfolgreich entwickelt werden.

Personaleinsatz HPA A: 6 PM

AP 5: Automatisches Echtzeit-Testfallausführungsverfahren

In dem Gesamtansatz sollen Testfälle einerseits automatisch erzeugt werden (s. AP 4) und andererseits soll auch die Ausführung automatisiert werden. Dafür wurde in diesem Arbeitspaket ein Verfahren zur automatischen Initialisierung, Ausführung und Protokollierung von Testfällen erforscht, welches in Kontext von Echtzeit-Anforderungen auf industriellen Steuerungen angewendet werden kann. Eine zentrale Herausforderung stellt dabei die wiederholbare Herstellung notwendiger Ausgangszustände des Gesamtsystems (Maschine/Anlage und Software) dar, um für spätere Regressionstests Veränderungen in der Steuerungssoftware sicher beurteilen zu können.

Das automatische Echtzeit-Testfallausführungsverfahren konnte erfolgreich entwickelt werden.

Personaleinsatz HPA A: 2 PM

AP 6: Software-Funktionsmuster als Basis der Evaluierung

Ziel dieses Arbeitspakets war die softwaretechnische Umsetzung der in Arbeitspaket 1 bis 5 erforschten Verfahren im Sinne eines Funktionsmusters als Grundlage für die Evaluierung des Gesamtansatzes. Die softwaretechnische Umsetzung berücksichtigt soweit möglich - und für den grundsätzlichen Funktionsnachweis des Verfahrens (s. AP 7) notwendig - geltende Standards (PLCopen XML und UML).

Das Funktionsmuster konnte erfolgreich umgesetzt werden.

Personaleinsatz HPA A: 6 PM; HPA B: 12 PM

AP 7: Evaluation des Gesamtansatzes

Im abschließenden Arbeitspaket wurde das im Projekt entwickelte Gesamtkonzept an einem der Forschungseinrichtung und zwei in der Industrie vorhandenen Demonstratoren evaluiert. Diese Demonstratoren decken die Systemklassen diskreter und hybrider Prozess ab.

Basis für die Evaluation ist das in AP 6 erarbeitete Software-Funktionsmuster, welches die Verfahren anwendbar und bewertbar macht.

Fragestellungen des Evaluationsprozesses sind hierbei z.B.:

- Unter welchen Randbedingungen sind die vorgeschlagenen Verfahren technisch anwendbar?
- Wie hoch ist der zusätzliche Aufwand für den Maschinen-/Anlagenbauer?
- Welchen Einfluss hat die Anwendung der Lösung auf die Zuverlässigkeit der Maschine/Anlage?

Die Evaluation wurde erfolgreich durchgeführt.

Personaleinsatz HPA A: 4 PM

AP 8: Projektsteuerung, Vorstellung von Projektergebnissen, Anfertigung von Berichten

Dieses Arbeitspaket erstreckte sich über die gesamte Projektlaufzeit und wurde parallel zu allen anderen Arbeitspaketen bearbeitet. Neben den administrativen Aufgaben der Organisation und Kontrolle des Projektverlaufs wurden in diesem Arbeitspaket die Aufwendungen für die Vor-/Nachbereitungen von Projekttreffen mit dem Projektbeirat zusammengefasst. Die Ergebnisse der jeweiligen Aufgabenpakete wurden zentral organisiert und transparent an den projektbegleitenden Ausschuss kommuniziert.

Des Weiteren wurden in diesem Arbeitspaket alle Aufgaben zum Ergebnis- und Wissenstransfer an den PA und die Industrie bearbeitet. In diesen Aufgabenbereich fallen auch die Anfertigungen von Veröffentlichungen in Fachzeitschriften und Konferenzen, sowie die Präsentation der Forschungsergebnisse auf Messen.

Die Vorstellung der Projektergebnisse wurde in vollem Umfang erreicht.

Personaleinsatz HPA A: 3 PM

1.2.5 Arbeitsdiagramm

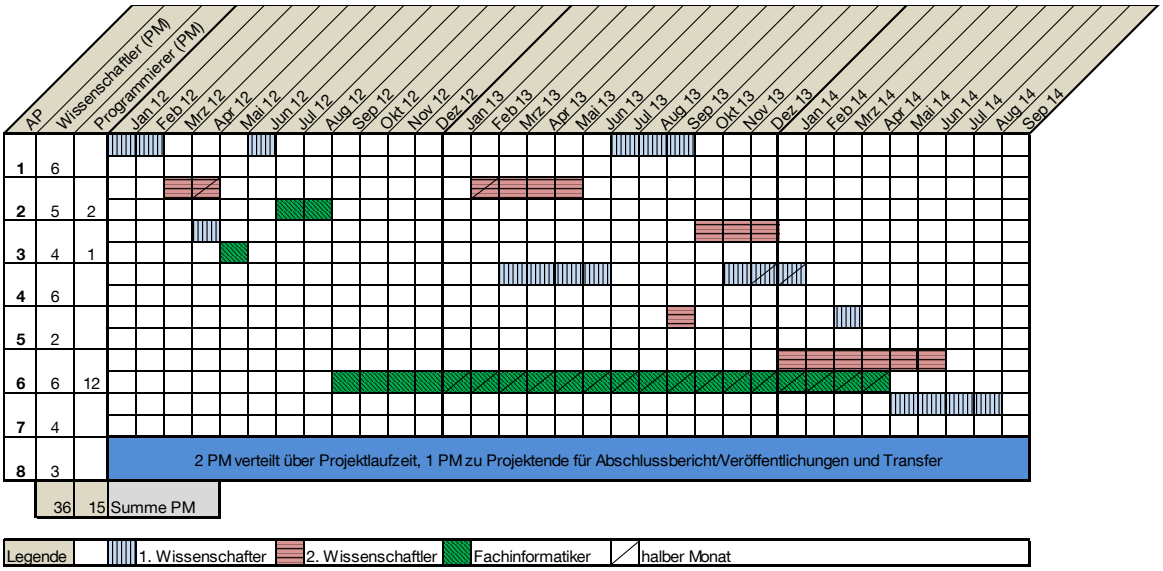


Abbildung 2: Projektplan (Laufzeit des Projektes 01.01.2012 – 30.09.2014)

1.2.6 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die in AP1 durchgeführten Arbeiten waren notwendig um die Grundlagen für ein anwendungsnahes und gut verständliches Modell für die Testfallgenerierung zu erreichen. Die Arbeit wurde von dem PA als angemessen bewertet (siehe Evaluation bezüglich des Modells in den Abschnitten 2.6.2, 2.6.3, 2.6.4).

Die in AP2 und AP3 durchgeführten Arbeiten waren notwendig um die in der Ist-Analyse gestellten Anforderungen (siehe 2.1) an die Überprüfung von Codierrichtlinien zu erfüllen. In AP2 wurde die Extraktion von IEC 61131-3 Code in einen Kontrollfluss untersucht, in AP3 die Überprüfung der Regeln realisiert.

Die in AP4 und AP5 geleisteten Arbeiten waren notwendig um aufbauend auf dem in AP1 entwickelten Modellierungsansatz Testfälle entsprechend der Anforderungen (siehe 2.1) zu generieren und auszuführen. Die Machbarkeitsstudien (siehe 2.6.2, 2.6.3) zeigen die erfolgreiche Umsetzung und damit die Angemessenheit der Arbeiten.

Das in AP6 umgesetzte Funktionsmuster trägt maßgeblich zur Evaluation und damit zur Ergebniseinschätzung der Industrieunternehmen bei. Die Veröffentlichung des Funktionsmusters auf einer Website (siehe Ergebnistransfer) zeigt die angemessene Umsetzung des Ansatzes und ermöglicht eine weitere Verbreitung des Ansatzes.

Die in AP7 durchgeführte Evaluation wurde über den im Antrag gestellten Anforderungen erfüllt und nicht nur an Laboranlagen sondern auch an Maschinen des Industrieunternehmen durchgeführt, womit eine deutlich bessere Ergebniseinschätzung in der Wirtschaft ermöglicht wurde.

Die in AP8 durchgeführten Arbeiten zur Veröffentlichung der Projektergebnisse waren für eine Verbreitung der Ergebnisse notwendig und wurden entsprechend den in Abschnitt 2.7 beschriebenen Maßnahmen umgesetzt.

2 Lösungsweg zur Erreichung des Forschungsziels

Im Folgenden werden die in den verschiedenen Arbeitspaketen entwickelten Konzepte und Funktionsmuster dargestellt. In Kapitel 2.1 werden die in AP 0 untersuchten Anforderungen für den Einsatz der Methodik in der Industrie und in Kapitel 2.2 die Untersuchung existierender Verfahren aufgezeigt. In Kapitel 2.3 wird das in AP 3 und 4 entwickelte Verfahren für die statische Codeanalyse erläutert. Kapitel 2.4 umfasst die Ergebnisse der entwickelten Methode für die Modellierung (AP 1), Testfallgenerierung (AP 4) und Testausführung (AP 5). Die Umsetzung der Methode in einem Funktionsmuster wird in Kapitel 2.5 dargelegt. Die Evaluierung (AP 7, Kapitel 2.6) und Zusammenfassung (Kapitel 3) schließen den Abschlussbericht ab.

2.1 Ist-Analyse im industriellen Umfeld zur Anforderungsverfeinerung

Im Folgenden werden aktuelle Vorgehensweisen anhand einer Ist-Analyse in Form eines Fragebogens und einer Untersuchung von Anwendungsbeispielen aus dem industriellen Umfeld des Projektbegleitende Ausschusses (PA) vorgestellt. Die Anwendungsbeispiele sind zum einen die Steuerungssoftware von vier Unternehmen des Projektausschusses zum anderen zwei Codierrichtlinien aus zwei Unternehmen.

Aus der Umfrage wurden die zentralen Anforderungen an den Ansatz des Projekts ZuMaTra abgeleitet und die Forschungsziele angepasst und verfeinert. Dabei wurden neun Mitglieder des PAs und somit eine Stichprobe von sowohl Dienstleistern, als auch Anwendern aus der Maschinenbauindustrie befragt.

Die Software-Anwendungsbeispiele und Codierrichtlinien wurden auf weitere Randbedingungen an den Ansatz und insbesondere auf Randbedingungen für einen automatisierten Test und die Codeanalyse untersucht.

2.1.1 Auswertung einer Umfrage mit Teilnehmern aus der Industrie

Im Folgenden werden die Ergebnisse der Umfrage mit 8 Teilnehmern aus 7 Unternehmen präsentiert. Die an der Umfrage teilnehmenden Unternehmen sind in der Abbildung 3 aufgelistet:

Abbildung 3: Teilnehmer der Umfrage

Momentan werden 25-30% der gemessenen Zeit an den gesamten Softwareentwicklungsstunden für das Testen der Steuerungssoftware aufgewendet. Daher ist es notwendig, dass für den zusätzlichen Test von Fehlerbehandlungsroutinen so wenig Zeit wie möglich aufgewendet werden muss. Folglich wurde folgende Anforderung formuliert:

A1: der Automatisierungsgrad der Testerstellung und –ausführung muss möglichst hoch sein.

In den Abbildungen 4 und 5 werden die Ergebnisse des aktuellen industriellen Testumfelds dargestellt: Während alle Unternehmen direkt am Arbeitsplatz Tests durchführen und über die

Hälfte vor Ort, wird am Teststand nur in wenigen Fällen getestet. Besonders bei der Entwicklung des Prototyps, bei Implementierung und während der Inbetriebnahme wird getestet, wogegen Tests direkt nach der Entwicklung unüblich sind. Dies ist vermutlich insbesondere auf fehlende Simulationsmodelle zurückzuführen. Der Ansatz muss also eine

A2: Durchführbarkeit sowohl gegen eine Simulation als auch gegen die reale Maschine

aufweisen.

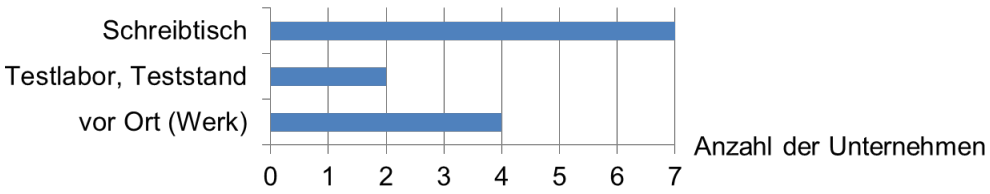


Abbildung 4: Testort

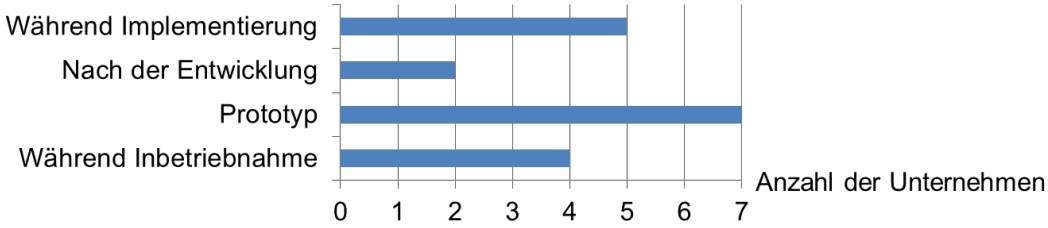


Abbildung 5: Tests nach Entwicklungsphase

Im zweiten Teil der Umfrage wurde der Abdeckungsgrad der Softwaretests analysiert. Dieser variiert stark (ø 40-60%) und liegt daher weit unter der Zielgröße von 100%. Die Möglichkeit zur manuellen, iterativen Detailierung der Tests ist erwünscht (siehe Abbildungen 6). Das Beschreibungsmittel muss daher eine möglichst

A3: freie Wahl des Abstraktionsgrades bei der Modellierung

aufweisen.

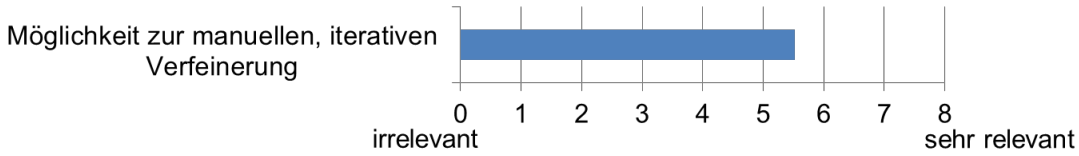


Abbildung 6: Bewertung der Möglichkeit zur manuellen und iterativen Verfeinerung der Tests.

Die Frequenz der Testanpassungen aufgrund geringer Änderungen und die daraus resultierende Verwaltung erzeugt einen nennenswerten Aufwand für die Unternehmen (siehe Abbildung 7). Daher folgt:

A4: die Neugenerierung von Testfällen und Anpassung von Modellen als Grundlage für die Generierung muss möglich sein.

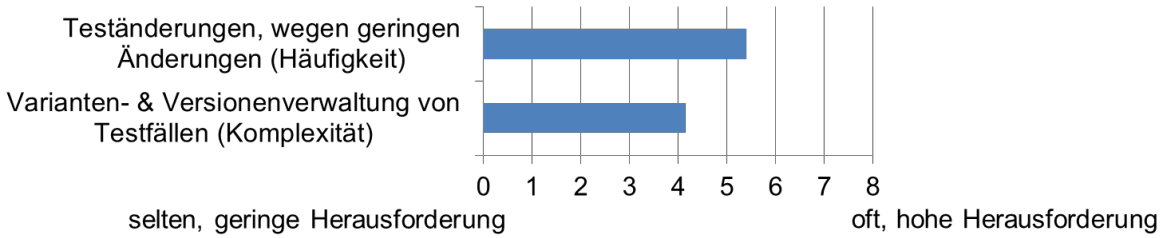


Abbildung 7: Häufigkeit der Softwareänderungen und Bewertung des Aufwands für die daraus resultierende Verwaltung

Im Hinblick auf die Detailierung des Testberichtes ist je nach Anwendungsbereich eine stichprobenartige bis vollständige Dokumentation aller Tests (v.a. Pflichtenheftanforderungen müssen abgedeckt sein) nötig. Die Bedeutung der Dokumentation bezüglich der Aspekte Bestimmung und

A5: Nachweis des Abdeckungsgrades der Anforderungen, sowie der Dokumentation für den Kunden

ist für die Teilnehmer von Relevanz (siehe Abbildung 8).

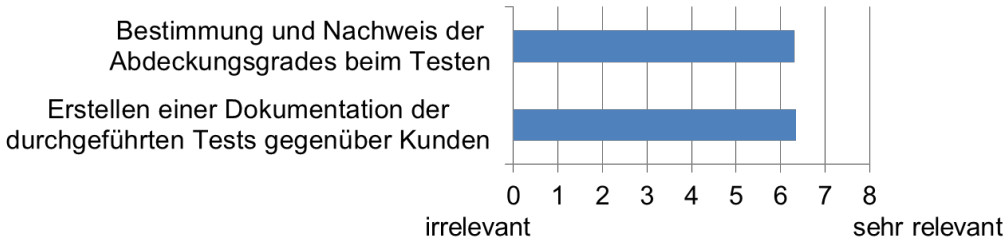


Abbildung 8: Bedeutung der Dokumentation

Bezüglich Relevanz von Simulation und Simulationsmodellen beim Testen wurde festgestellt, dass

- als Mehraufwand max. ~20% der Softwareentwicklungsstunden für ein Simulationsmodell möglich sind,
- die Simulation je nach Teilbereich realistisch, d.h. mit hohem Detailierungsgrad, sein muss oder nur relevante Maschinenzustände und Übergangszeiten beinhalten soll.

Die Simulation des Bedienerverhaltes ist wichtiger als die des Werkstücks für die Testfallausführung. Letzteres ist nur von mäßiger Relevanz (siehe Abbildung 9):

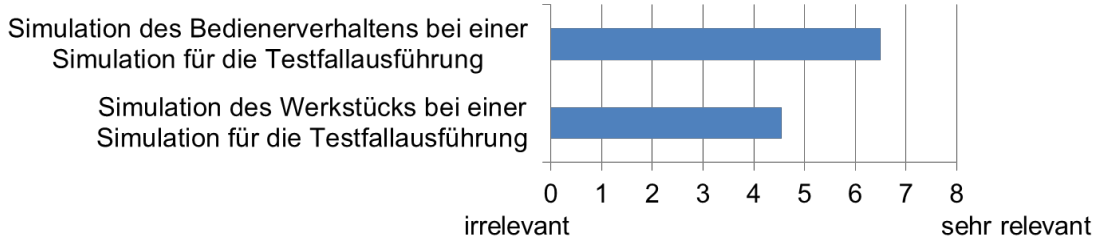


Abbildung 9: Relevanz der Simulation des Bedienerverhaltens und des Werkstücks

Die Aussage, dass neben dem Gut-Verhalten der Test des Schlecht-Verhaltens genauso wichtig ist, zeigt den Bedarf nach neuen Methoden für den Test dieser noch einmal in besonderer Weise auf (siehe Abbildung 10). Ziel muss es also sein eine Methodik für den

A6: Test von relevanten Fehlerszenarien

zu entwerfen.

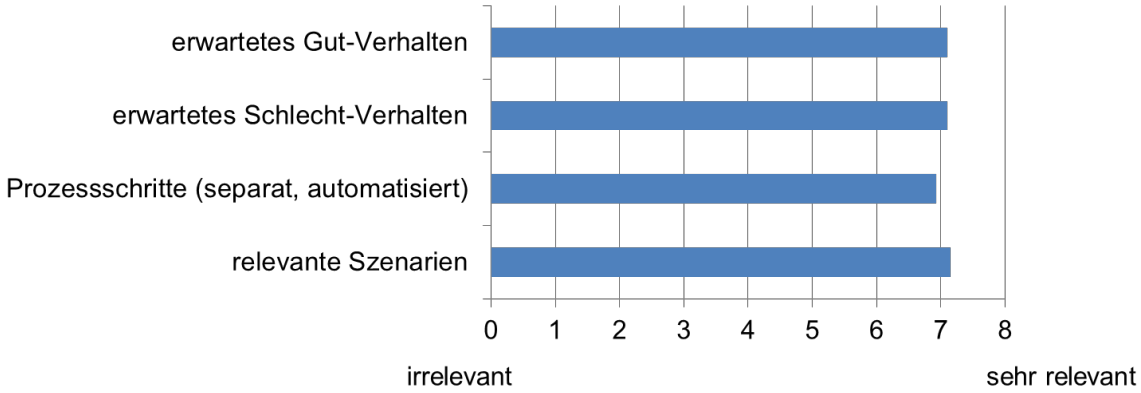


Abbildung 10: Testziel

Um die Einführungsbarrieren der neuen Methodik möglichst gering zu halten, wurde das Ziel verfolgt auf bereits etablierte Beschreibungssprachen zurückzugreifen. Funktionsbeschreibungen, textuelle Programmiersprachen, Schrittketten, Weg-Zeit-Diagramme und State Charts sind für mehr als die Hälfte der Unternehmen als Informationsquellen für die Testfallgenerierung relevant. Auch die beiden weiteren Informationsquellen, Anforderungstabelle und Ablaufsprache, werden von 3 der 7 Unternehmen eingesetzt. (siehe Abbildung 11).

Da das Weg-Zeit-Diagramm als graphisches Beschreibungsmittel leicht anpassbar und formalisierbar für die Testfallgenerierung ist, wurde diese Notation als Basis für die Testfallgenerierung gewählt. Die Anforderung nach einer

A7: Testfallgenerierung aus Weg-Zeit-Diagrammen

muss also, nach Konsens des projektbegleitenden Ausschusses, erfüllt werden.

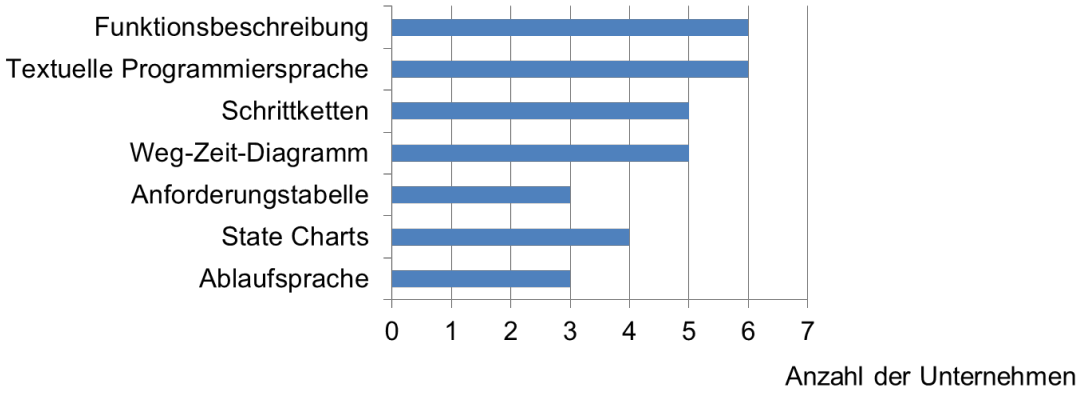


Abbildung 11: Informationsquellen zur Testfallgenerierung

Im Wesentlichen führen folgende Fehler in der Entwicklung zu Verzögerungen bei der Inbetriebnahme und zu Fehlverhalten im Betrieb:

- Nicht eindeutige Spezifikation von Kundenanforderungen
  - Fehlende Definition von Testfällen
  - Mängel in der Ablaufbeschreibung
- Nicht dokumentiertes Verhalten, Funktionen oder Abläufe
  - Schnittstellen zur Installation
  - Schnittstellen zwischen Hard- und Software (Sensorverhalten bei Über- und Unterspannung)
  - Verhalten bei unerwarteten Störungen
- Mangelhafte Auswahl oder Auslegung von Sensoren oder Aktoren
- Softwarefehler bei
  - Typumwandlungen
  - Scheduling-Vorgaben
  - oder bei unzureichenden Parameterüberprüfungen

Die Fehlerursachen sind gleichermaßen auf Prozessfehler, mechanische Fehler, Fehler in der Sensorik und Aktorik, der Software und Bedienerfehlern zurückzuführen (Abbildung 12).

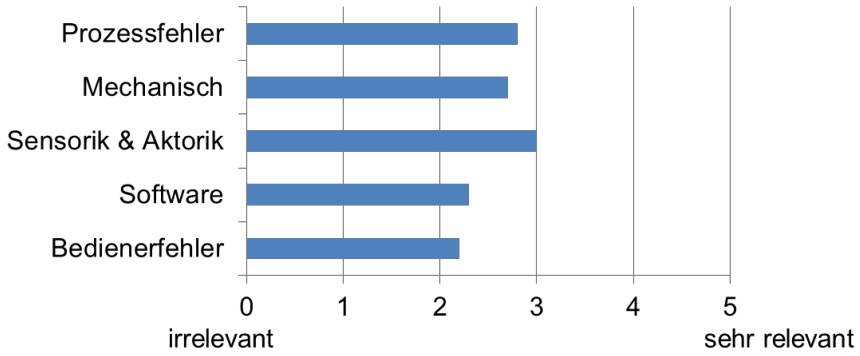


Abbildung 12: Lokalisierung typischer Fehlerarten

Als Werkzeuge zur Komponentenbeschreibung setzen 6 von 8 Teilnehmern (Prozess-)FMEA ein. Die darin enthaltenen Informationen werden in der Abbildung 13 beschrieben. 3 der 7 Teilnehmer beschreiben Informationen über die Abhängigkeit zu anderen Komponenten und über die Ausfallwahrscheinlichkeit. Die weiteren Aspekte (Schnittstellen zur Steuerungssoftware und die Ausfallbeobachtungswahrscheinlichkeit) werden kaum verwendet. Eine vertiefende Diskussion ergab, dass die FMEA hauptsächlich für die Untersuchung von Fehlern der Elektronik oder für Produktfehler eingesetzt wird, nicht jedoch für die Beschreibung von Fehlerursachen, die von der Steuerungssoftware behandelt werden müssen. Folglich wurde mit dem Projektausschuss eine

A8: optionale Verwendung der FMEA für die Priorisierung der Testfälle

postuliert. Der Aufwand einer durchgängigen Erstellung von FMEA für alle Komponenten wurde als zu aufwändig eingestuft.

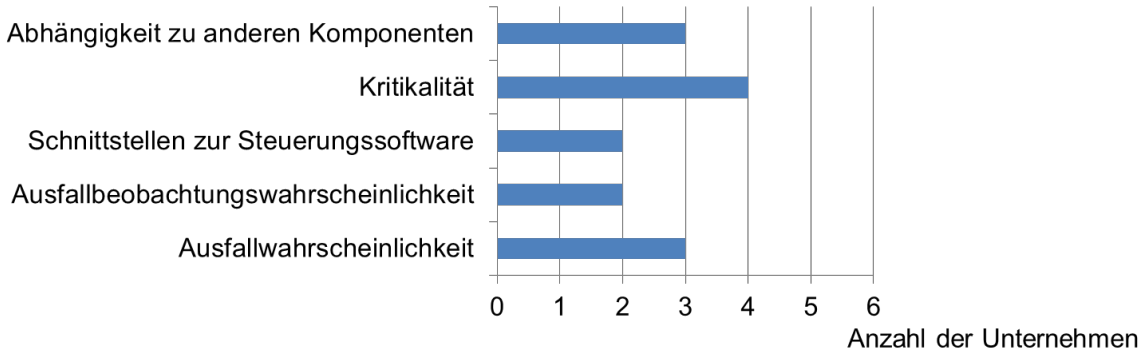


Abbildung 13: Informationen der Werkzeuge für die Komponentenbeschreibung

Zusammenfassend ergeben sich aus der Umfrage unter den Unternehmen folgende gesammelte Anforderungen:

A1: der Automatisierungsgrad der Testerstellung und –ausführung muss möglichst hoch sein.

A2: Durchführbarkeit sowohl gegen eine Simulation als auch gegen die reale Maschine

A3: freie Wahl des Abstraktionsgrades bei der Modellierung

A4: die Neugenerierung von Testfällen und Anpassung von Modellen als Grundlage für die Generierung muss möglich sein

A5: Nachweis des Abdeckungsgrades der Anforderungen sowie der Dokumentation für den Kunden

A6: Test von relevanten Fehlerszenarien

A7: Testfallgenerierung aus Weg-Zeit-Diagrammen

A8: optionale Verwendung der FMEA für die Priorisierung der Testfälle

2.1.2 Analyse von Anwendungsbeispielen aus der Industrie

Neben der allgemeinen Umfrage für die Ermittlung der Anforderungen wurden 4 Anwendungsbeispiele aus der Industrie für die Ermittlung weiterer Randbedingungen durchgeführt. Der Fokus bei der Untersuchung lag auf folgenden Gesichtspunkten:

- Allgemeiner Aufbau des Fehlermanagements und der Fehlerbehandlung
- Arten von Fehlern die behandelt werden
- Aufbau von Fehlererkennungsmechanismen
- Art der Fehlerbehandlung für die verschiedenen Fehler

Untersucht wurden dabei 4 Anwendungsbeispiele aus 4 unterschiedlichen Unternehmen, die sowohl kontinuierliche als auch diskrete Prozesse enthielten:

- Beispiel A (Siemens: Kontaktplan (KOP) + Anweisungsliste (AWL))
  - Transportsystem (mit zwei synchronisierten Bänder) mit Werkstückträgern: diskrete Prozesse
  - Ca. 200 Programmorganisationseinheiten (POUs)
- Beispiel B (SoMachine: Funktionsbausteinsprache (FBD) + Strukturierter Text (ST))

- Pflasterverpackungsmaschine mit 2 Teilen: 1. Teil kontinuierliche + 2. Teil diskrete Prozesse
- Ca. 450 POUs

– Beispiel C (TwinCat 2 & 3 : ST)

- Kleine Sortieranlage: diskrete Prozesse
- Ca. 20 POUs

– Beispiel D (TwinCat 3: Ablaufsprache (AS) + ST)

- O-Ring-Montage: diskrete Prozesse
- Ca. 100 POUs

Bei der Untersuchung konnten die folgenden Erkenntnisse gewonnen werden. Der Aufbau des Fehlermanagements folgt, ebenso wie der Aufbau der restlichen Software, einer hierarchischen Struktur. Die Fehlererkennung erfolgt in den meisten Fällen auf unterster Bausteinebene von Bausteinen, die die Funktion einer Komponente steuern (siehe Abbildung 14). Auf dieser Ebene wird entschieden, ob der Fehler auf die nächste Hierarchieebene (Steuerung mehrerer Komponenten – Station) weitergeleitet oder lokal behandelt wird. In den meisten Fällen wird der Fehler weitergegeben und es wird eine entsprechende Fehlermeldung gesetzt. Auch auf Stationsebene kann der Fehler lokal behandelt werden, wird jedoch in den untersuchten Beispielen ebenfalls meist weitergeleitet und in einem separaten Baustein, der für das Fehlermanagement zuständig ist, ausgewertet. Die Fehlermanagement-Bausteine sind in der Regel für die Erkennung von Sammelfehlern und die Entscheidung, welche Fehlerbehandlung eingeleitet wird, zuständig, welche dann an alle Bausteine durch das Setzen einer Variablen getriggert wird (z.B. Heimlaufstopp).

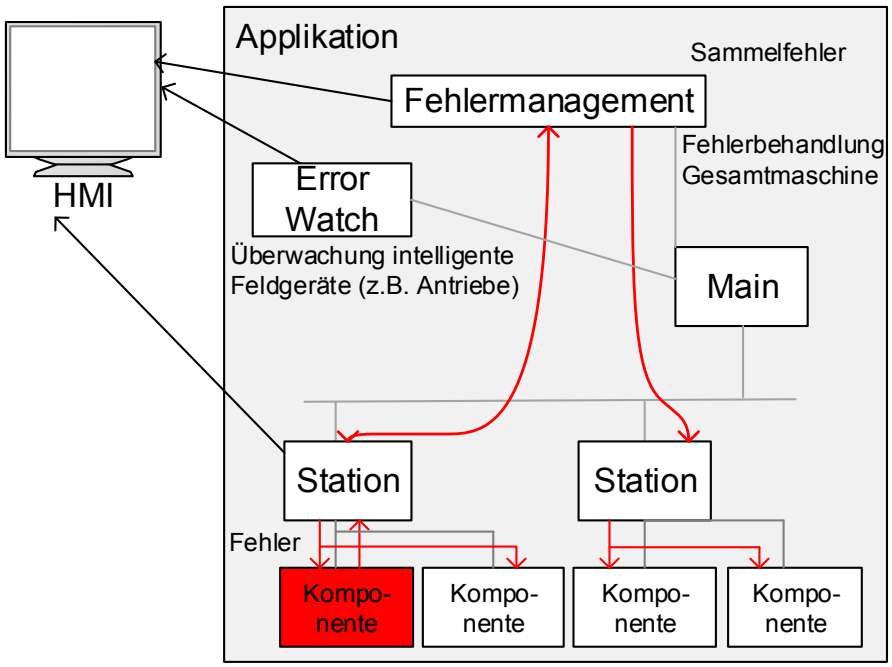


Abbildung 14: Fehlerbehandlung von Steuerungssoftware

Da bei dem Ansatz spezifische Szenarien getestet werden sollen, muss die Maschine hierfür in eine bestimmte Ausgangssituation gebracht werden. Alle Anwendungsbeispiele haben eine solche definierte Grundstellung, von der aus der Automatikmodus gestartet werden kann. Darüber hinaus wurde eine wichtige Erkenntnis bei der Untersuchung dieser Stellung aus den Anwendungsbeispielen gewonnen.

RB 1: Alle Anwendungsbeispiele enthalten Routinen für eine Grundstellungsfahrt/ Referenzpunktfahrt/ Reset.

Es können von den Testfällen also bereits definierte Routinen genutzt werden, um nach einer Fehlerinjektion wieder in einen definierten Zustand zu gelangen. Eine separate Implementierung für die Testfälle ist nicht notwendig.

Eine weitere Untersuchung der Routinen zu Grundstellungsfahrten ergab, dass hierfür bei einigen Anwendungsbeispielen manuelle Operatoreneingriffe notwendig sind. Daher wurde folgende Randbedingung aufgenommen:

RB 2: manuelle Eingriffe durch den Operator während der Testausführung müssen spezifizierbar sein.

Von den Ursachen, die zum Versagen eines Automatisierungssystems führen können, werden die in Abbildung 15 blau markierten von der Steuerungssoftware erkannt und behandelt.

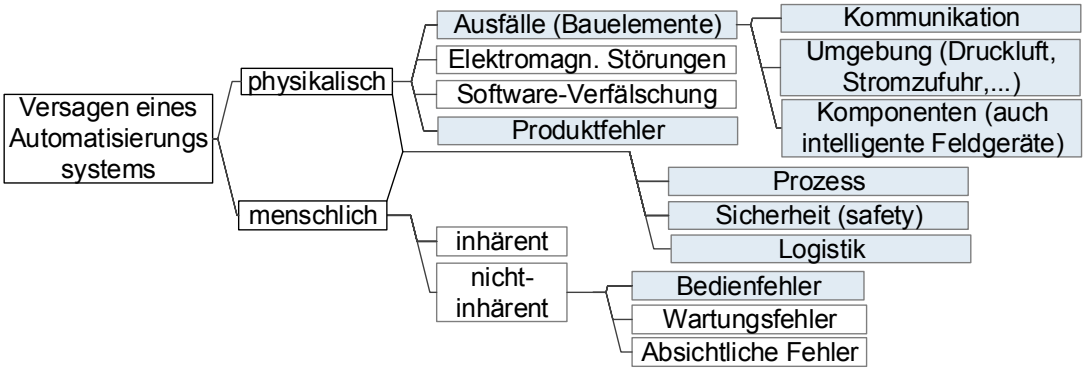


Abbildung 15: Fehler die von der Steuerungssoftware behandelt werden

Die Fehler werden dabei durch die in Abbildung 16 genannten Mechanismen erkannt. Die Überprüfung von Parametrierungen kommt insbesondere bei der Erkennung von Bedienerfehler zur Anwendung, also bei Schreib- und Lesevorgängen der Bedienerschnittstelle. Mit der Überprüfung von Sensoren werden die meisten Fehlerursachen überwacht. Bei der Überprüfung von Verschränkungen wird beispielsweise überprüft, ob ein Zustand gültig sein kann (es können z.B. nicht beide Endlagen eines Pneumatikzylinders gleichzeitig eingenommen werden). Die Laufzeitüberwachung überprüft, ob ein Sensorsignal in einer bestimmten, durch Anforderungen spezifizierte Zeitbeschränkungen, erreicht/ inaktiv wird. Gültige Intervalle werden insbesondere bei Temperatur- oder Druckluftüberwachungen überprüft. Ein fehlendes Signal ist für die Überprüfung von Safety-Funktionen wichtig, wie z.B. der Überwachung, ob die Schutztüren geschlossen sind. Sammelfehler werden bei der Erkennung von zwei oder mehr Fehlern zur gleichen Zeit gesetzt und geben meist detaillierteren Rückschluss auf eine mögliche Fehlerursache. Als eine Besonderheit bei der den Fehlererkennungsmechanismen wurden die Verriegelungsbedingungen identifiziert. Diese können nur begrenzt zu den Fehlerbehandlungsroutinen gezählt werden, da sie zur Fehlervermeidung eingesetzt werden. Dies funktioniert, indem bei jeder möglichen Ausführung, also in jedem möglichen Kontrollflusspfad, bestimmte Variablen überprüft werden. Als Beispiel ist die Überprüfung von Betriebsarten zu nennen. Es wird in jedem Zyklus überprüft, welche Betriebsart aktiv ist, ansonsten sind verschiedene Aktionen nicht erlaubt und werden nicht aufgerufen.

Die Fehlerkennungsmechanismen sind insbesondere für die Anforderung nach Tests relevanter Fehlerszenarien zu beleuchten, da die Fehlerinjektion genau diese Fehlererkennungsmechanismen triggern muss.

RB 3: Für den Test relevanter Fehlerszenarien müssen Fehler entsprechend der untersuchten Fehlererkennungsmechanismen injiziert werden.

Da Verriegelungen in jedem Pfad überprüft werden, macht eine Fehlerinjektion in diesem Falle jedoch wenig Sinn. An dieser Stelle ist vielmehr eine statische Codeanalyse angebracht, mit welcher überprüft werden kann, ob die Verriegelungsbedingung tatsächlich in jedem Pfad eingehalten wird.

Im Gegensatz dazu spiegeln die anderen Fehlererkennungsmechanismen bestimmte Szenarien wieder. Auf Basis von Code- bzw. Kontrollflussanalyse generierte Testfälle würden jedoch jedes mögliche Szenario, welches zu einer Fehlererkennung führt, abbilden. Dies bestätigen auch erste, in einem interdisziplinären Praktikum durchgeführte, Versuche. Die Komplexität der Pfadanalyse ist sehr hoch und führte zu einer Generierung einer Unmenge von Testfällen, die zum einen an einer Maschine kaum durchführbar sind und zum anderen keine realistischen Szenarien wiederspiegeln. Eine anforderungsbasierte Testfallgenerierung auf Basis der Modellierung ist hier vom Kosten-Nutzen-Faktor deutlich vorzuziehen. Es wurden daher folgende Randbedingungen formuliert.

RB 3.1: Zur Überprüfung von Verriegelungsbedingungen muss der Kontrollfluss untersucht werden.

RB 3.2: Zur Überprüfung von Fehlerbehandlungen von bestimmten Szenarien (Prüfung Parametrierung, Prüfung (Sensoren), Überprüfung komplexes Signal und Sammelfehler) soll eine anforderungs- bzw. modellbasierte Testfallgenerierung durchgeführt werden.

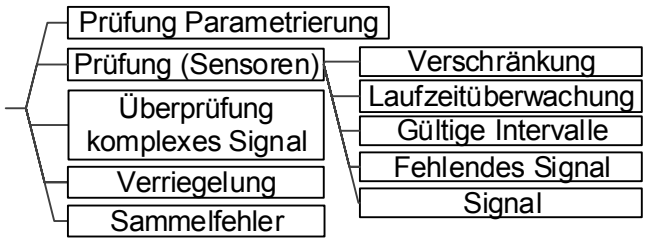


Abbildung 16: Fehlererkennungsmechanismen

Die Untersuchung von Fehlerbehandlungen ergab weitere wichtige Rückschlüsse. In allen Beispielen gibt es für die Behandlung von erkannten Fehlern keine lokale, individuelle Fehlerbehandlung. Die Fehler werden stattdessen nach Kritikalität eingestuft und entsprechend behandelt. Klassische Beispiele für Fehlerbehandlungsklassen sind Heimlaufstopp, Nothalt, etc. (Abbildung 17). Neben der allgemeinen Fehlerbehandlung gibt es jedoch noch einen individuellen Alarm und eine Markierung der betreffenden Produkte als Schlechtteil.

Als Randbedingungen wurden hier daher festgehalten:

RB 4: Tests zur Prüfung von Fehlerbehandlungen können aufgeteilt werden in:

- *Fehlerindividueller Teil: Test der richtigen Fehlererkennung und Meldung*
- *Test der richtigen Fehlerbehandlung für verschiedene Fehlerbehandlungsklassen*

Der Vorteil der aus dieser Randbedingung gezogen werden kann ist, dass der Aufwand zur Testspezifikation deutlich reduziert werden kann. Die verschiedenen Fehlerbehandlungen (Heimlaufstopp, Nothalt, etc.) können für die Tests einmal spezifiziert und für weitere Tests stets wiederverwendet werden.

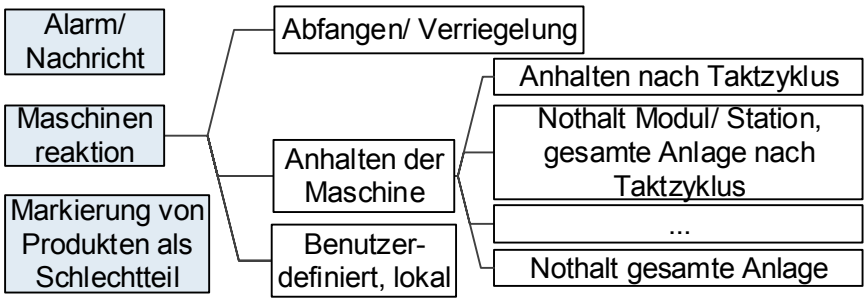


Abbildung 17: Fehlerbehandlung

Auf Wunsch des Projektausschusses und bei genauer Analyse der Anwendungsbeispiele kann es bei einer Fehlerinjektion von einigen Fehlern zu unsicheren bzw. undefinierten Zuständen der Maschine kommen. Folglich wurde zusätzlich die Randbedingung 5 formuliert:

RB 5: eine Abbruchroutine muss vorgesehen werden, um undefinierte Zustände zu vermeiden.

Zusammenfassend konnten aus der Analyse der Anwendungsbeispiele folgende Randbedingungen ermittelt werden:

RB 1: Alle Anwendungsbeispiele enthalten Routinen für eine Grundstellungsfahrt/ Referenzpunktfahrt/ Reset.

RB 2: manuelle Eingriffe durch den Operator während der Testausführung müssen spezifizierbar sein.

RB 3: Für den Test relevanter Fehlerszenarien müssen Fehler entsprechend der untersuchten Fehlererkennungsmechanismen injiziert werden.

RB 3.1: Zur Überprüfung von Verriegelungsbedingungen muss der Kontrollfluss untersucht werden.

RB 3.2: Zur Überprüfung von Fehlerbehandlungen von bestimmten Szenarien (Prüfung Parametrierung, Prüfung (Sensoren), Überprüfung komplexes Signal und Sammelfehler) soll eine anforderungs- bzw. modellbasierte Testfallgenerierung durchgeführt werden.

RB 4: Tests zur Prüfung von Fehlerbehandlungen können aufgeteilt werden in:

- *Fehlerindividueller Teil: Test der richtigen Fehlererkennung und Meldung*
- *Test der richtigen Fehlerbehandlung für verschiedene Fehlerbehandlungsklassen*

RB 5: eine Abbruchroutine muss vorgesehen werden, um undefinierte Zustände zu vermeiden

2.1.3 Untersuchung von Codier- bzw. Ausführungsrichtlinien

Eine Diskussion mit dem Projektausschuss ergab, dass Verriegelungsbedingungen unternehmensspezifisch in Codier- bzw. Ausführungsrichtlinien festgehalten werden. Es wurden Dokumente entsprechender Art von zwei Unternehmen untersucht. Da die Richtlinien weitere Regeln enthalten, für die sich eine Überprüfung an Kontrollflüssen anbietet, wurden auch diese extrahiert.

Abstrahiert wurden folgende Ausführungsrichtlinien, die über die klassische Codeanalyse hinausgehen extrahiert:

1. Regel: Bei jeder Schrittweiterschaltung müssen entsprechende Statusvariablen (return values) aktualisiert werden (sonst Abbruch)
2. Regel: Nach jedem Start-Schritt muss es einen Warteschritt geben
3. Regel: Innerhalb von FBs dürfen keine globalen Variablen verwendet werden
4. Regel: Es darf nicht auf das Eingangsabbild zurückgeschrieben werden

Regel 1 und Regel 2 sind dadurch gekennzeichnet, dass die Schritte bestimmte Lese- oder Schreibzugriffe auf bestimmte Variablen haben. Daher ist es möglich Kontrollflusspfade auf bestimmte Abfolgen von Lese- und Schreibvorgängen von Variablen zu überprüfen.

Es wurde daher folgende zusätzliche Anforderung formuliert:

A9: Die Codeanalyseregeln sollen entsprechend der aus den Unternehmen analysierten Regeln und den Verriegelungsbedingungen möglich sein

2.2 Vorrecherche Stand der Forschung und Entwicklung

Im Folgenden werden zu dem Forschungsthema relevante aktuelle Veröffentlichungen bzw. Ansätze vorgestellt und anhand von zentralen, aus dem Kontext des Antrages abgeleitete Kriterien miteinander verglichen. Als wesentliche Aspekte dieses Forschungsvorhabens können die Fokussierung auf Steuerungssoftware, die statische Analyse von Steuerungscode, das automatisierte Erstellen und Ausführen von Testfällen, sowie die besondere Berücksichtigung von Fehlerbehandlungsroutinen angesehen werden. Tabelle 1 fasst diesen Vergleich abschließend zusammen.

2.2.1 Statische Codeanalyse

Das Projekt [LD09] beschäftigt sich mit der testgetriebenen Automatisierung. Dabei wird versucht, potentielle Fehlerquellen in IEC 61131-3 Code mit Hilfe einer statischen Codeanalyse aufzudecken. Zu einem ausgiebigen Softwaretest werden zusätzlich Funktionstests benötigt, um das dynamische Verhalten zu überprüfen. Zur Formulierung der Testfälle wird ein schlüsselwortbasiertes Vorgehen vorgeschlagen. Die Testfallgenerierung erfolgt gemäß einer vorherigen Spezifikation in Excel-Tabellen. In diesem Ansatz werden keine Techniken der Fehlerinjektion angewendet.

In [FB05] wird ein Re-Engineering Ansatz für SPS Steuerungscode (IEC 61131-3) in einem zweistufigen Prozess eingeführt. Dabei wird die Programmstruktur zuerst in ein UML-Modell transformiert. Anschließend wird das Verhalten der Software (die verwendeten Algorithmen) in einen endlichen Automaten umgewandelt. Dieser Formalisierungsschritt ermöglicht die Anwendung weiterer Analyseverfahren, Verifikations- und/oder Simulationsläufe. Die Verwendung im Kontext einer Testfallgenerierung wird jedoch nicht angestrebt.

Zur Analyse der Struktur, Qualität und Fehlerfreiheit von Software haben sich Ansätze zur statischen Codeanalyse etabliert [EmNi08]. Im Gegensatz zu dynamischer Codeanalyse, die eine Ausführung der Software erfordert, bestimmt statische Codeanalyse die Eigenschaften, z.B. Softwarestrukturen oder mögliche Programmzustände [ArBi05], ohne Ausführung der Software [EmNi08]. Mittels statischer Analyse können zudem Abweichungen von der gewünschten Komplexität der Software oder auffällige Codefragmente identifiziert werden. Obwohl bereits einige Werkzeuge zur statischen Codeanalyse existieren, z.B. Lint für C [Jo78] und FindBugs für Java [AHM+08], wird IEC 61131-3 bisher nur von wenigen Anbietern unterstützt [APR+13]. CODESYS Static Analysis analysiert den Code anhand vordefinierter Regeln, beispielsweise zur Überprüfung der Einhaltung von Namenskonventionen oder Identifikation unerreichbarer Codebestandteile. Mittels itris PLC Checker können darüber hinaus Programmablaufpläne dargestellt, Komplexitätsmetriken analysiert und durch „Copy and Paste“ wiederverwendete Softwareeinheiten identifiziert werden. Es existiert jedoch bisher noch kein Ansatz, der eine Spezifikation unternehmensspezifischer Kriterien für Ausführungsrichtlinien erlaubt.

2.2.2 Fehlerinjektion

Um die Zuverlässigkeit von Systemen zu validieren, hat sich die Fehlerinjektion (FI) als Methode etabliert. Mit der Fehlerinjektion können die Mechanismen der Fehlerbehandlung und Fehlererkennung überprüft werden. FI-Ansätze können in hardwareimplementierte FI (HWIFI), softwareimplementierte FI (SWIFI) und modell- bzw. simulationsbasierte FI (MIFI) aufgeteilt werden [SVE+10]. Während HWIFI und SWIFI meist bei Prototypen oder für Systemtests verwendet werden, wird MIFI tendenziell eher in den frühen Phasen des Entwicklungsprozesses angewendet, um frühzeitig Feedback, bezüglich der Funktion eines Systems, zu bekommen [HTI97].

Die Testmethoden können auch durch die Arten von Fehlern, die injiziert werden, unterschieden werden. In einige Ansätzen werden bestimmte Klassen von Fehlern und die Reaktion eines Systems auf diese Klassen überprüft. Bei diesen Ansätzen werden explizit Fehlermodelle, d.h. mögliche Störungen, die in manchen Ansätzen auch als Mutanten oder Saboteure bezeichnet werden, spezifiziert. Das Fehlermodell definiert die Arten der möglichen Fehler eines Systems in Bezug auf verschiedene Kriterien wie die Phase der Erstellung (Design, Umsetzung, etc.), die Dimension (Hardware-Fehler, Software-Fehler), die Systemgrenze (von innerhalb oder außerhalb des Systems injizierte Fehler, etc.) oder die Persistenz (vorübergehender oder permanenter Fehler) [ALR + 04]. Bei vielen Ansätzen wird außerdem der Aspekt des Zeitverhaltens des Fehlers fokussiert, also das sporadische oder zufällige Auftreten eines Fehlers im Gegensatz zu Fehlern zu definierten Zeitpunkten.

Die drei verschiedenen Ansätze - HWIFI, MIFI und SWIFI werden in den folgenden Abschnitten näher betrachtet.

2.2.2.1 Hardwareimplementierte Fehlerinjektion (HWIFI)

Es existieren bereits zahlreiche Werkzeuge um integrierte Schaltkreise und insbesondere Mikroprozessoren mit HWIFI zu testen. Die Methoden sind daher insbesondere auf diese Art von Systemen und dementsprechend auf Arten von Fehlern, die hier auftreten können, ausgerichtet. Beispiele sind elektromagnetische Störungen [ZAV04] und Störungen auf Pin-Level Störungen [HTI97]. Ein Überblick über verschiedene Tools und Methoden kann in [HTI97] und [ZAV04] gefunden werden.

Für SPSen existieren zwar bereits Ansätze für Hardware-in-the-Loop-Prüfstände [SKV00], es wurde jedoch noch keine besondere Aufmerksamkeit auf FI-Techniken in diesem Bereich gelegt. Geeignete Fehlermodelle und Studien in diesem Bereich sind daher nicht verfügbar.

2.2.2.2 Modellbasierte Fehlerinjektion (MIFI)

Bei MIFI kann unterschieden werden, ob die Fehler in Hardware- oder in die Softwaremodelle injiziert werden. Hardwaremodelle existieren beispielsweise in der Elektrotechnik in Form der „Very High Speed Integrated Circuit Hardware Description Language“ (VHDL) und wurde insbesondere für integrierte Schaltkreise entworfen. Dementsprechend werden diese Modelle für die Fehlerinjektion und eine Simulation der Fehlerreaktion auf diese Fehler genutzt [BGG+05].

Im Automobilbereich gibt es weiterhin einige Ansätze für die modellbasierte Fehlerinjektion [SVE+10]. In der Regel werden bei diesen Ansätzen vor allem bereits vorhandene MATLAB/ Simulink-Modelle für die Fehlerinjektion genutzt, welche ohnehin im Entwicklungsprozess verwendet werden. In [SVE+10] wird darüber hinaus das Ergebnis der Testläufe für die Testfallgenerierung für Systemtests genutzt.

In der Produktionsautomatisierung wird in [KoVo11] ein Ansatz für die Fehlerinjektion in ausführbare UML-Zustandsdiagramm-Modelle vorgeschlagen. In dem Ansatz wird weiterhin vorgeschlagen aus dem Programmcode alle möglichen Pfade, die zu einem Komponentenausfall durch sogenanntes „program slicing“ führen, zu extrahieren, um volle Pfadabdeckung zu gewährleisten. Der Ansatz wurde jedoch nur auf konzeptioneller Ebene umgesetzt. Weiterhin werden keine Fehlermodelle verwendet, sondern fehlerhafte Komponenten müssen manuell im Modell ausgewählt werden.

2.2.2.3 Softwareimplementierte Fehlerinjektion (SWIFI)

Ebenso wie für MIFI und für HWIFI sind bereits Werkzeuge für die SWIFI in integrierten Schaltkreise vorhanden [CMS98]. Der Vorteil der SWIFI ist, dass sie garantiert zerstörungsfrei und reproduzierbar ist. In [PAC+12] wird ein Ansatz für die Validierung spezifizierter Safety-Funktionen vorgeschlagen. Die Vorgehensweise unterstützt die Überprüfung nach der Einhaltung dieser Safety-Funktionen unter jeder Bedingung. Fehlermodelle oder die Schaffung einer anwenderfreundlichen Notation sind nicht im Fokus der Arbeit.

Neben der FI bei der Co-Simulation wird in [SlVu05] auch die Einbringung von definierten Fehlern bei der Codegenerierung vorgeschlagen. Dieser Fehler tritt während der Ausführung auf. Ein ähnlicher Ansatz wird in [VBR+07] unter der Nutzung von SCADE-Modellen vorgeschlagen.

2.2.3 Testen in der Automatisierungstechnik

Im Maschinen- und Anlagenbau bzw. in der Produktionsautomatisierung ist das manuelle Testen immer noch dominierend. Seit einigen Jahren wird der Bedarf nach automatisierten Tests in dieser Domäne jedoch wahrgenommen und erste Ansätze bzw. Werkzeuge für die automatisierte Durchführung von Tests erscheinen auf dem Markt [Testmanager].

Auch in der Forschung existieren bereits einige Ansätze für die automatisierte Testdurchführung.

In [EKF+09] wird auf der Grundlage von IEC 61499 ein Grey-Box-Testverfahren erläutert. Bei dieser vorgeschlagenen testgetriebenen Entwicklung werden die Testdaten, bestehend aus Eingangs- und Ausgangsdaten, dem auslösenden Ereignis und dem erwarteten Ergebnis vom Steuerungsprogrammierer spezifiziert. Dieser Ansatz auf Unit-Testebene bedarf der manuellen Beschreibung des Testfalls durch den Entwickler und verwendet keine Automatisierung bei der Erstellung. Die Regressionstests werden jedoch automatisiert. Durch die fest vorgegebenen Testfälle erfolgt keine Fehlerinjektion zur Laufzeit, die ein mögliches Maschinenfehlverhalten darstellen könnte.

In [SEK+09] wird die automatische Ausführung von Tests für Steuerungsprogramme untersucht. Eine Funktionsspezifikation dient zur Erzeugung der Testabläufe auf einem Prüfstand, der hardwaregebunden mit der SPS gekoppelt ist. Dabei werden keine Fehler direkt injiziert, sondern der spezifizierte Funktionsumfang (also die Gutfälle) abgeprüft.

In dem Ansatz nach [StEr08] wird die Übertragbarkeit von Testkonzepten aus der Anwendungsentwicklung auf SPS-Steuerungssoftware überprüft. Dabei steht die automatische Ausführung von Modultests in Form von daily builds im Vordergrund. Eine wesentliche Kernaussage ist: „Leider unterstützen die meisten Toolhersteller im SPS-Umfeld die wesentlichen Aspekte einer modernen Softwareentwicklung nur unzureichend“ [StEr08].

In [OM09] sollen möglichst viele verschiedene Testfälle zur Überprüfung des korrekten Verhaltens einer SPS-Steuerungssoftware generiert und ausgeführt werden. Außerdem soll ein Verfahren für die intuitive Testfallspezifikation entwickelt werden. Es werden jedoch keine Mechanismen zur Fehlerinjektion in der Skizze genannt.

Das Forschungsprojekt „Virtueller Funktionstest für eingebettete Systeme (ViFES)“ fokussiert auf die auch in diesem Projekt angestrebte Vorverlagerung der Abnahmetestfälle in frühere Entwicklungsphasen und deren Wiederverwendung durch den Einsatz von Testautomaten für Regressionstests [Ru07].

Diese Lösungsansätze sind ein erster Schritt in die richtige Richtung. Der hohe Aufwand der Testerstellung ist damit jedoch noch nicht gelöst, um Tester adäquat zu unterstützen. In der Forschung wird daher intensiv an der Fragestellung der Generierung von Testfällen aus semiformalen Modellen durch die Weiterentwicklung und Formalisierung verschiedener Notationen gearbeitet.

In [Ot08] wird ein Testverfahren für Funktionsbausteine für funktional sichere Anwendungen vorgestellt. Dabei werden die auszuführenden Testfälle aus einem Status-Diagramm erzeugt, das als Funktionsbaustein-Spezifikation vorausgesetzt wird. Das Ergebnis des ausgeführten Testfalls wird anhand eines Ein-/Ausgabevergleichs in das Test-Logbuch eingetragen. Es werden Parameter aufgrund einer Risikoanalyse für Testfälle ausgewählt, jedoch werden keine Fehler direkt injiziert. Der dafür notwendige White-Box Test wird lediglich zur statischen Codeanalyse verwendet.

Die Unified Modeling Language (UML) ist eine der weitverbreitetsten Notation um die Struktur und das Verhalten von Software zu modellieren, daher ist es nicht verwunderlich, dass einige Forschungsansätze diese Sprache als Grundlage zur Testfallgenerierung nutzen.

In [KHC+99] werden Unit-Tests aus UML-Zustandsdiagrammen erzeugt. Die Transformation aus diesen Diagrammen heraus in erweiterte, endliche Zustandsautomaten kann sowohl kontrollfluss- als auch datenflussorientiert sein. Bei der Testdurchführung wird somit überprüft, ob sich das System-Under-Test (SUT) gemäß der Spezifikation (UML Zustandsdiagramm) verhält. Eine Berücksichtigung von Fehlersituationen ist bei diesem Ansatz nicht gegeben.

In [HWÖ+10] und [HuFr06] werden geeignete Diagramme der UML für die Testfallgenerierung untersucht, insbesondere mit dem Ziel Testfälle für die nach IEC 61499 implementierten Steuerungssoftware zu generieren. Dabei werden Interaktionsdiagramme für die Extraktion von Testsequenzen vorgeschlagen. [HKV+11] setzt eine solche Testfallgenerierung aus Zustandsdiagrammen mit Anwendung eines Algorithmus einer Extraktion aller Pfade um.

[KHD08] schlägt ebenso einen Ansatz zur automatischen Testfallgenerierung aus UML-Zustandsdiagrammen vor, indem diese zunächst in ein formales Petrinetzmodell übersetzt werden. Daraufhin können die Testfälle durch das Auffalten der Petrinetze generiert werden.

In [KCB11] wird die Testfallgenerierung aus UML-Zustandsdiagrammen über den Standard der Testing and Control Notation (TTCN-3) umgesetzt. Die Evaluation wird an einem Kommunikationsprotokoll gezeigt.

Die Ausführbarkeit durch Anpassung und Formalisierung von Sequenzdiagrammen wird in [KTV12] fokussiert. Dabei entstehen Sequenzdiagramm, die direkt als Testfälle in der CODESYS Programmierumgebung ausführbar sind.

Tabelle 1: Vergleich und Bewertung der existierenden Ansätze

	Referenz	Statische Codeanalyse	Fokussierung auf Fehlerbehandlung	Automatische Generierung von Testfällen	Automatisierte Ausführung von Testfällen	Fokussierung auf Steuerungssoftware
	[LD09]	+	-	-	+	+
	[FB05]	+	-	-	-	+
	[Jo78]	+	-	-	-	-
	[AHM+08]	+	-	-	-	-
	[CODESYS]	+	-	-	-	+
	[itris PLC Checker]	+	-	-	-	+
	[ZAV04]	-	+	-	+	-
	[HTI97]	-	+	-	+	-
	[SKV00]	-	-	-	+	+
	[BGG+05]	-	+	-	+	-
	[SVE+10]	-	+	+	+	-

Veröffentlichungen, Werkzeuge und Projekte	[KoVo11]	-	+	-	+	+
	[CMS98]	-	+	-	+	-
	[PAC+12]	-	+	-	+	-
	[SlVu05]	-	+	+	+	-
	[VBR+07]	-	+	+	+	-
	[Testmanager]	-	-	-	+	+
	[EKF+09]	o	-	-	o	+
	[SEK+09]	-	-	o	o	+
	[StEr08]	-	-	-	o	+
	[OM09]	-	-	+	o	+
	[Ru07]	o	-	o	+	o
	[Ot08]	o	-	+	+	o
	[KHC+99]	-	-	+	-	o
	[HWO+10][HuFr06]	-	-	+	-	+
	[HKV+11]	-	-	+	-	+
	[KHD08]					
	[KCB11]	-	-	+	-	+
	[KTV12]	-	-	-	+	+
	Legende: + Merkmal erfüllt, - Merkmal nicht erfüllt, o Merkmal teilweise erfüllt					

Wie in Tabelle 1 gezeigt, erfüllt keiner der existierenden Ansätze und Werkzeuge die Anforderungen an eine automatisierte Lösung für den Test von Fehlerbehandlungsroutinen im Bereich der Steuerungssoftware. Auf den Mangel solcher Lösungen wird zum Teil auch in [StEr08] eingegangen. Es wird insbesondere deutlich, dass es nur vereinzelt (im Automotive-Bereich) zum Test von Fehlerbehandlungsroutinen kommt, obwohl diese den Großteil der Gesamtsoftware ausmachen. Der Stand der Forschung und Entwicklung zeigt klar den Handlungsbedarf und somit die Notwendigkeit einer Lösung.

2.3 Verfahren zur statischen Codeanalyse

Neben dem Test von Fehlerbehandlungsroutinen wurde aus der Zielstellung und den im Projekt ermittelten Randbedingungen die Notwendigkeit abgeleitet, dass die Steuerungs-Software auf die Einhaltung von Ausführungsrichtlinien überprüft werden muss. Als umzusetzende Ausführungsrichtlinien wurden anhand der in der Ist-Analyse gegeben *Anforderung A9* gemeinsam mit den Industriepartnern des Projekts ein grundsätzliches Regelwerk entworfen, dass sich prinzipiell auf die IEC 61131-3 Programmiersprachen anwenden lässt:

1. Regel: Bei jeder Schrittweiterschaltung müssen entsprechende Statusvariablen (return values) aktualisiert werden (sonst Abbruch)
2. Regel: Nach jedem Start-Schritt muss es einen Warteschritt geben
3. Regel: Innerhalb von FBs dürfen keine globalen Variablen verwendet werden
4. Regel: Es darf nicht auf das Eingangsabbild zurückgeschrieben werden

Die Regeln müssen wie auch die Verriegelungsbedingungen bei jeder Ausführung, also in jedem möglichen Kontrollflusspfad, eingehalten werden. Um die Regeln an einem Code-Modell automatisch überprüfen zu können, müssen sie in einer formalisierten Art vorliegen.

Da ST die in den Anwendungsbeispielen am häufigsten verwendete Sprache ist wurde zunächst ein Konzept für die Extraktion eines Kontrollflussgraphen und einer Analyse desselben entwickelt. Das Vorgehen für die Codeanalyse ist in Abbildung 18 dargestellt. Zunächst wird der Code aus einem PLCopen XML Dokument extrahiert und über einen abstrakten Syntaxbaum (AST) in einen Kontrollflussgraphen (CFG) extrahiert. Nachdem eine Regel entsprechend der oben genannten Beispiele in formalisierter Form angegeben wurde, kann der Kontrollflussgraph schließlich auf die Einhaltung der Regel überprüft werden.

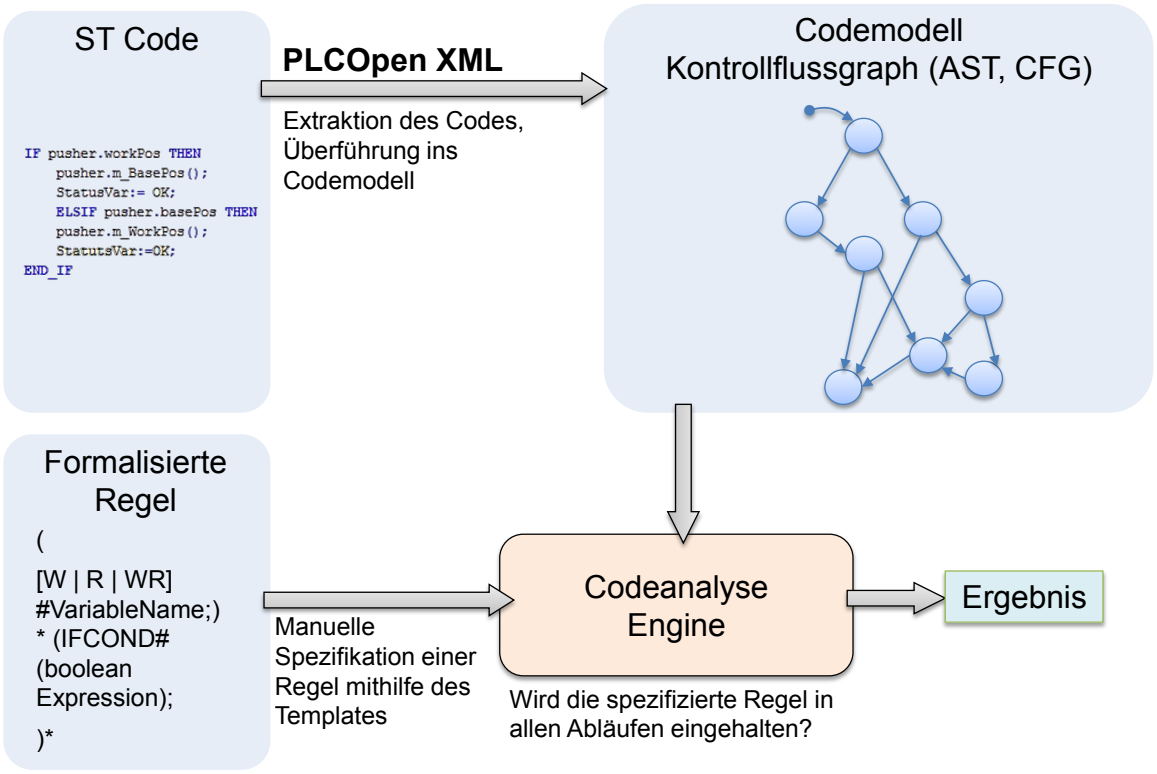


Abbildung 18: Grundsätzliche Vorgehensweise bei der statischen Codeanalyse von ST-Code

Abbildung 18 zeigt die verschiedenen Darstellungsmöglichkeiten für den Code. Neben der Darstellung als Kontrollfluss, unterstützt die ZuMaTra-Codeanalyse auch die Darstellung von sogenannten intraprozedualen Kontrollflüssen, bei denen Aufrufe anderer Funktionen und Funktionsbausteine „aufgeklappt“ werden können. Die Aufrufhierarchie kann dementsprechend ebenfalls auf die Einhaltung der Richtlinien überprüft werden.

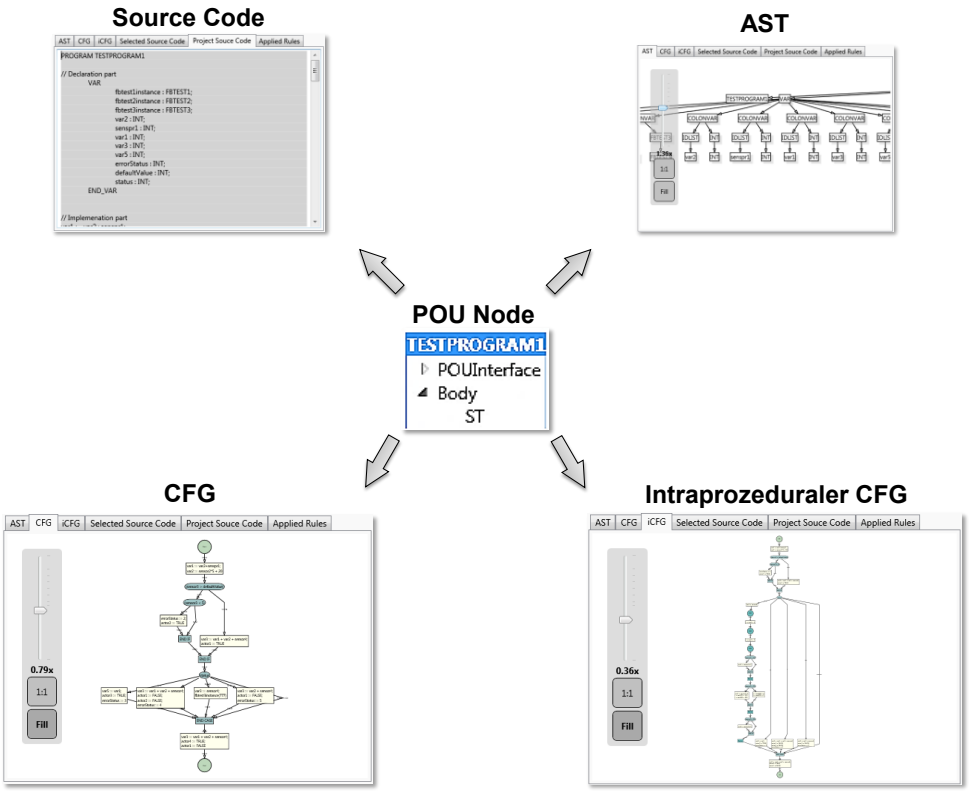


Abbildung 19: Verschiedene Darstellungsweisen des Codes

Regel 1 und Regel 2 sind dadurch gekennzeichnet, dass die Schritte bestimmte Lese- oder Schreibzugriffe auf bestimmte Variablen haben. Daher ist es möglich Kontrollflusspfade auf bestimmte Abfolgen von Lese- und Schreibvorgängen von Variablen zu überprüfen.

Um die Regeln formalisiert spezifizieren zu können, wurde ein einfaches Template, angelehnt an ST, entwickelt. Dies ermöglicht den Anwendern, die mit der IEC 61131-3 vertraut sind, einen einfachen Einstieg und schnelles Verständnis für die Spezifikation.

- **Target:** Modell, das untersucht werden soll (CFG oder iCFG)
- **Paths:** Bedingungen, nach denen die zu testenden Pfade ausgewählt werden:
  - ALL; – es werden alle möglichen Pfade betrachtet.
  - [W | R | WR]#VariableName; – Auswahl Pfade, entlang denen die bei der Variable „VariableName“ ein [schreibender | lesender | schreibender oder lesender] Zugriff erfolgt.
  - IFCOND#(booleanExpression); – Pfade, welche über einen Verzweigungsknoten (IF) verlaufen, der die Bedingung „booleanExpression“ hat. „booleanExpression“ ist ein boolescher Ausdruck nach IEC 61131-3 Syntax.
- **PathCondition:** Bedingung die überprüft werden soll (gleiche Spezifikation wie bei Paths)
- **Beispielregel:** „In allen Pfaden eines Kontrollflusses in denen ein Schreibzugriff auf VariableX stattfindet soll überprüft werden ob VariableX<5 ist.“

- Target: CFG;
- Path: W#VariableX;
- PathCondition: IFCOND#(VaribaleX<5);

Alle Pfade des Kontrollflusses werden entsprechend den spezifizierten Regeln untersucht.

2.4 Vorgehensweise für den Test von Fehlerbehandlungsroutinen

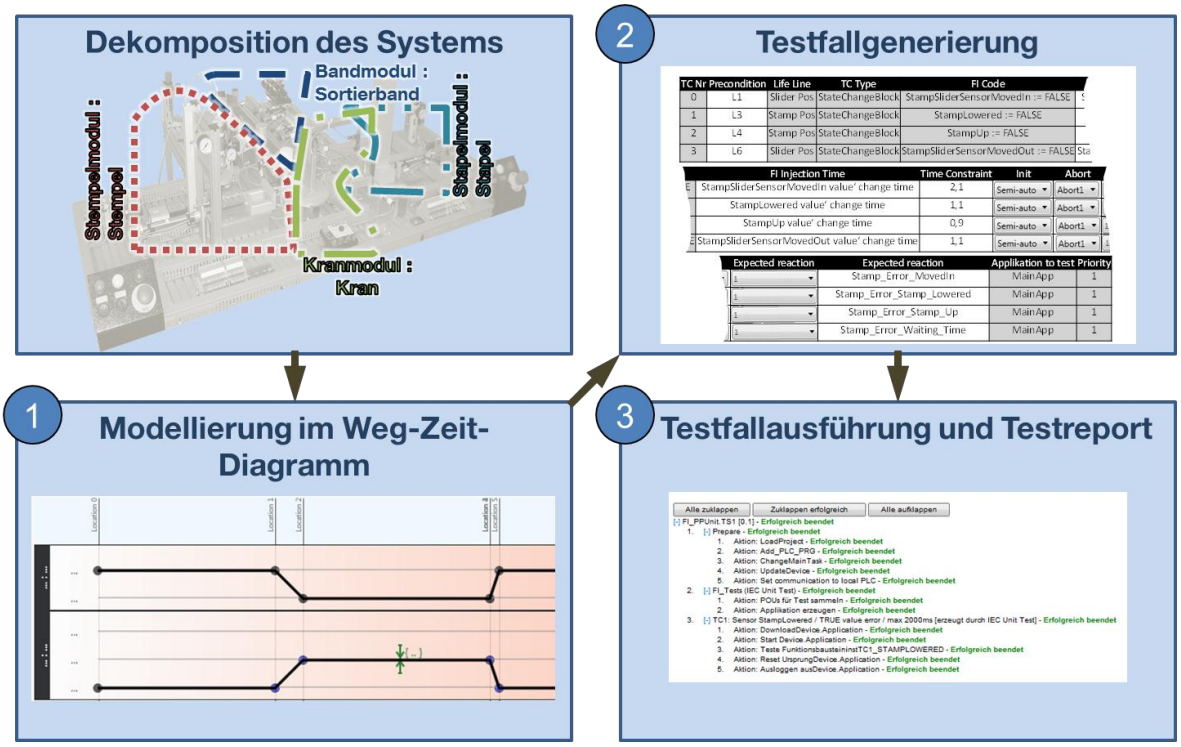


Abbildung 20: ZuMaTra-Vorgehensweise

Die Vorgehensweise zum Test von Fehlerbehandlungsroutinen von Maschinen und Anlagen umfasst 4 Schritte. Zunächst wird die Maschine in Funktionseinheiten aufgeteilt (Experten-Know-How, keine zusätzliche Unterstützung durch ZuMaTra). Für jede Funktionseinheit wird anschließend das Verhalten modelliert, aus welchem die Testfälle generiert werden. Damit die Testfälle möglichst automatisiert durchgeführt werden können, muss anschließend eine Tabelle vervollständigt werden, in welcher die Angaben zur Ausführung der Testfälle ergänzt werden müssen. Darüber hinaus können die Testfälle nach Kritikalität priorisiert werden. Auf Basis der Tabelle werden die Testfälle generiert und können mit PLCopen XML in die Programmierumgebung zur Ausführung importiert werden.

Die Testfälle werden auf Basis des modellierten Verhaltens generiert und berücksichtigen Zusammenhänge zwischen den verschiedenen modellierten Komponenten in einem Diagramm. Die Aufteilung der Einheiten sollte sich daher möglichst an dem Verhalten der Maschine orientieren. Für die Übersichtlichkeit sollte darauf geachtet werden, dass die Einheiten nicht zu groß gewählt werden, da sonst die Übersichtlichkeit bei der Modellierung verloren geht.

Für die Anwendung der Vorgehensweise sei insbesondere auf den Leitfaden verwiesen (siehe Anhang CD: FMEA Leitfaden).

2.4.1 Modellierungsverfahren für Komponenten (Abbildung 20, Nr. 1)

2.4.1.1 Modellierung im Weg-Zeit-Diagramm

Das Verhalten der Maschine wird bei der ZuMaTra-Vorgehensweise mit Weg-Zeit-Diagrammen modelliert. Dabei fokussiert werden die Sensorvariablen, welche den Zustand der Maschine abbilden.

Das Weg-Zeit-Diagramm ist nach der UML 2.0 ein Interaktionsdiagramm. Das Diagramm wird durch „Lifelines“, welche bestimmte Objekte repräsentieren und deren Zustandsinvarianten („State Invariants“), welche einen bestimmten Zustand der „Lifeline“ repräsentieren, strukturiert. Das Verhalten wird durch die Änderung der Zustände über die horizontale Zeit-Achse modelliert. Weitere Elemente wie Nachrichten („Messages“) und Zeitintervalle („Duration Interval“) bieten die Möglichkeit einer Modellierung von Interaktionen zwischen den „Lifelines“ und von Zeitabschnitten.

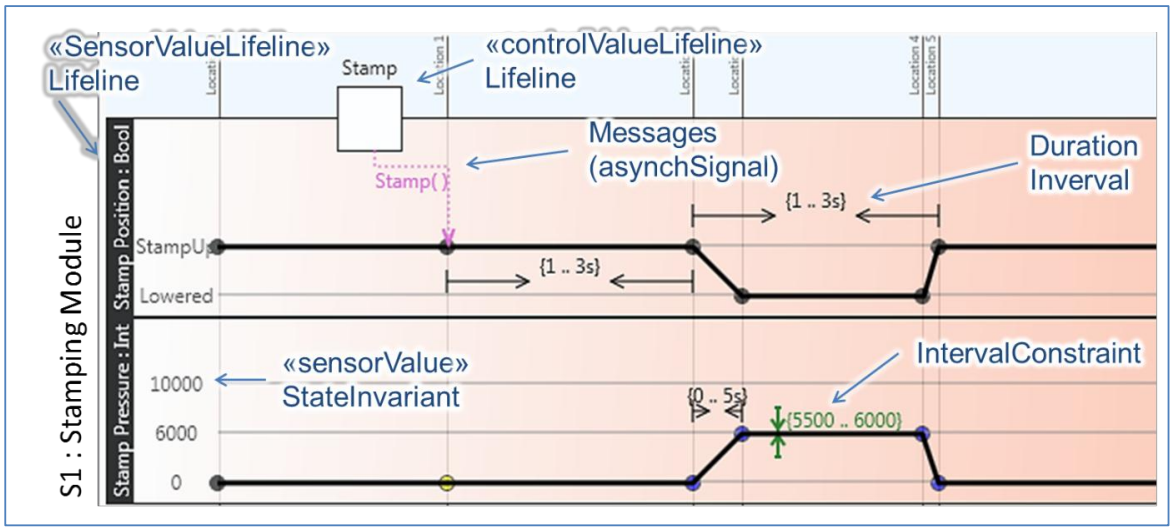
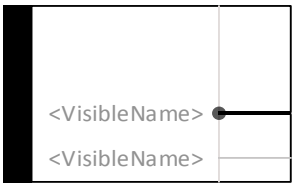
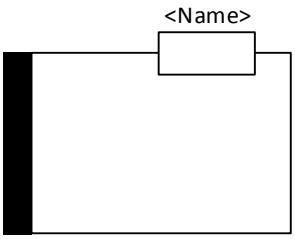
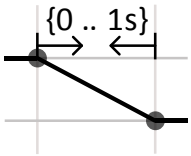
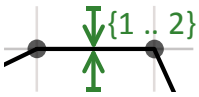
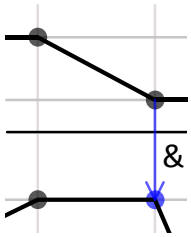


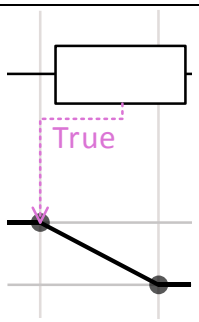
Abbildung 21: Das Weg-Zeit-Diagramm mit den Modellierungselementen

Das Weg-Zeit-Diagramm wurde im Rahmen des Projekt durch Profilierung angepasst (siehe Anhang Abbildung 38). Bei ZuMaTra ist das Weg-Zeit-Diagramm die Abbildung des Gutverhaltens einer Maschine und bildet die Grundlage für die Testfallgenerierung. Ein Beispiel ist in Abbildung 21 dargestellt. Die einzelnen Elemente werden in Tabelle 2 näher beschrieben. Bei der Definition der „StateInvariants“ (siehe Abbildung 21 und Tabelle 2) wurde auf die Einhaltung der Anforderung A3 geachtet. Durch die Definition einer „StateInvariant“ nach Wert, können beliebig viele bzw. wenige Invarianten für eine Sensorvariable definiert werden.

Tabelle 2: Elemente des Weg-Zeit-Diagrammes zur Beschreibung des Gut-Verhaltens.

Element	Notation	Beschreibung
«SensorValue Lifeline» Lifeline	<Name> : <Typ>	**<Name>:** Komponente, die abgebildet wird. **Beschreibung:** Kann beliebig gewählt werden und dient der Aufteilung in sinnvolle Einheiten. Eine *Lifeline* kann einen oder mehrere Sensoren abbilden.

«sensorValue» State Invariant		**<Variable>:** Bezeichner der Sensorvariable **<VisibleName>:** Sichtbarer Name für ein besseres Verständnis und größere Übersichtlichkeit **<Value>:** Konkreter Wert, den der Sensor auf diesem State Invariant annimmt. Für einen don't care Zustand kann „ ***** “ eingefügt werden. **Timeline-Points** „●“ unterstützen die Modellierung des Verhaltens und markieren einen Zustandswechsel bzw. Beginn und Ende wichtiger Abschnitte.
«ControlValue Lifeline» Lifeline		**<Name>:** Konkreter Bezeichner **Beschreibung:** Modellierung von weiteren beliebigen Variablen, die nicht das konkrete Verhalten der Funktionseinheit abbilden, sondern auf das Verhalten Einfluss nehmen. Z.B. Betriebsart oder andere HMI-Variablen.
Duration Interval		**Beschreibung:** Zeitintervall zwischen zwei verschiedenen Punkten bzw. Zuständen und Zustandswechseln.
Interval Constraint		**Beschreibung:** Intervall bzw. Toleranz für einen bestimmten Sensorwert (State Invariant) für den modellierten Zeitraum zwischen zwei Punkten (einem Zustand). Es können auch Variablen verwendet werden, auf welche von dem Testfall zugegriffen wird. *Bsp.: {Stamp.MinValue .. Stamp.MaxValue}*
«And» Message		**Beschreibung:** Message, um die Abhängigkeit zweier Zustandswechsel zu modellieren.

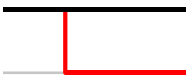
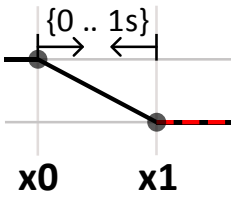
«IECMessage» Message (asynchSignal)		**<Value>:** Wert zum Zeitpunkt, an dem die Message gesendet wird. Diese Arten von Messages werden von einer *ControlValue* gesendet. Die Variable wird als Vorbedingung für die folgenden Zustandswechsel miteinbezogen.

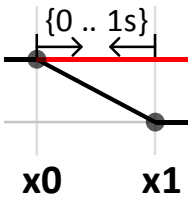
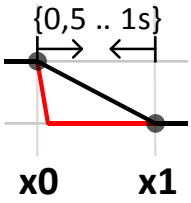
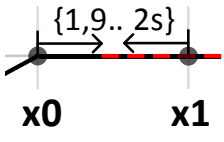
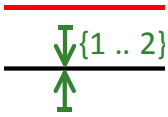
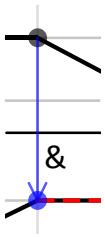
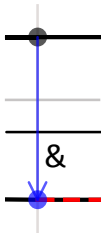
2.4.2 Automatische Testfallgenerierung (Abbildung 20, Nr. 2)

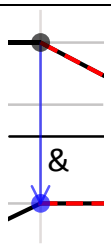
Die Testfälle zum Test von Fehlerbehandlungsroutinen simulieren stets eine Abweichung des Gutverhaltens. Eine Abweichung vom Gutverhalten kann mehrere Formen annehmen, welche im folgendem im Detail aufgelistet werden. Die Fehleroperatoren entsprechen dabei genau den nach Randbedingung RB 3.2 definierten Fehlererkennungsmechanismen und erfüllen damit Anforderung A6.

Die Fehleroperatoren werden automatisch auf das Weg-Zeit-Diagramm angewandt, wenn die Voraussetzungen für ihre Anwendung vom Modell gegeben sind, womit Anforderung A4 und A7 erfüllt werden. Durch die automatische Generierung wird auch der erste Teil der Anforderung A1 einer weitgehend automatischen Testfallerstellung erfüllt.

Tabelle 3: Fehleroperatoren für die Testfallgenerierung.

Element	Notation	Beschreibung
Zufälliger Zustandswechsel		Es wird keine Zustandsänderung erwartet, es taucht jedoch eine auf. **Regeln für die Modellierung:** Ein konstanter Abschnitt auf der Lifeline ist gegeben. **Anpassungsmöglichkeiten:** In der Testtabelle kann der Zeitpunkt der Fehlerinjektion durch Veränderung der Precondition (Location) konkret bestimmt werden.
Fehlender Zustandswechsel 1 (StateChange Block)		Nach Zustand **x0** wird Zustand **x1** innerhalb des spezifizierten Zeitintervalls erwartet. Das Signal des Sensors kommt jedoch nicht. Der Testfall überschreibt das Sensorsignal. **Regeln für die Modellierung:** Definierter Zustand (mit Zeit) **x0**, von dem aus das Zeitintervall startet und beim zweiten Zustand **x1** mit Angabe einer maximalen Zeit endet.

Fehlender Zustandswechsel 2 (StateChange BlockConstant)		Nach Zustand **x0** wird Zustand **x1** innerhalb des spezifizierten Zeitintervalls erwartet. Der Zustand hält jedoch länger als erwartet. **Regeln für die Modellierung:** Definierter Zustand (mit Angabe der maximalen Zeit), von dem aus das Zeitintervall startet und bei zweitem Zustand **x1** nach einer maximalen Dauer endet.
Zu früher Zustandswechsel 1 (StateChange Force)		Für einen Zustandswechsel von **x0** nach Zustand **x1** wird eine bestimmte Mindestzeit erwartet, der Wechsel geht aber schneller als gedacht. **Regeln für die Modellierung:** Definierter Zustand (Zeit), von dem aus das Zeitintervall mit der Angabe einer Mindestzeit > 0 startet und bei zweitem Zustand **x1** endet.
Zu früher Zustandswechsel 2 (StateChange ForceConst)		Die Dauer des Zustands **x0** wird für eine bestimmte Zeit erwartet, bricht jedoch früher ab als erwartet. **Regeln für die Modellierung:** Definierter Zustand **x0** (Zeit), von dem aus das Zeitintervall startet und eine minimale Dauer angegeben ist.
Verletzung eines gültigen Intervalls		Für einen Sensorzustand wird ein gültiges Intervall angegeben, dieses wird jedoch über- oder unterschritten. **Regeln für die Modellierung:** Angabe eines IntervalConstraint.
Bei Zustandswechsel erfolgt kein synchroner Zustandswechsel auf spezifizierter Lifeline.		Bei einem Sensorsignal wird zwingend ein anderes Sensorsignal erwartet. **Regeln für die Modellierung:** Verknüpfung des abhängigen Signals mit eine Message mit „&“.
Ein Zustand auf einer Lifeline zu einem bestimmten Zeitpunkt entspricht nicht dem erwarteten Zustand.		Bei einem Zustand wird in jedem Fall ein anderer Zustand erwartet. **Regeln für die Modellierung:** Modellierung einer „&“-Message.

Sammelfehler: Kein Zustands-wechsel nach einem Zeitin-tervall auf zwei Lifelines		Nach einem Zeitintervall werden synchron zwei Zu-standswechsel erwartet, diese kommen aber nicht. **Regeln für die Modellierung:** Modellierung eines Zeitintervalls mit einem Zustands-wechsel und einer „&“-Message.
Legende: Gutverhalten, Injizierter Fehler, Geblocktes Signal		

Der Anwender hat die Möglichkeit die Fehleroperatoren, welche auf das Weg-Zeit-Diagramm angewandt werden sollen, auszuwählen. Nach der Auswahl wird eine Testtabelle, bei der jede Zeile für einen Testfall steht, generiert. Der Aufbau der Tabelle des automatisch generierten Teils (Tabelle 3) ist wie folgt:

- Die Vorbedingung (*Precondition*) gibt den Zustand an, bei der die Vorbedingung für den Testfall als erfüllt gilt. Die Locations lassen sich direkt im Weg-Zeit-Diagramm zurückverfolgen und sind dort dargestellt.
- Die Spalte *Lifeline* gibt an, welche „Lifeline“ im Zentrum des Tests steht.
- Die Spalte *FI Code* zeigt an, welche Sensorvariable wie überschrieben wird.
- In der Spalte *FI Injection Time* wird angegeben, wann die Fehlerinjektion stattfindet. Bei „StateChangeBlock“ ist dies beispielsweise standardmäßig der Zeitpunkt, bei dem das Gutverhalten des Sensors detektiert wird (Bsp. „Stamp Filled“ wird erwartet, kommt im Ablauf auch entsprechend, wird aber geblockt). Bei Angabe einer Zahl ist die Zeit ab Erreichen der Vorbedingung gemeint.
- Der *Time Constraint* gibt an, wie lange die Reaktion dauern darf (Differenz zur „FI Injection Time“).

TC Nr	Precondition	Life Line	TC Type	FI Code	
0	L1	Slider Pos	StateChangeBlock	StampSliderSensorMovedIn := FALSE	S
1	L3	Stamp Pos	StateChangeBlock	StampLowered := FALSE	
2	L4	Stamp Pos	StateChangeBlock	StampUp := FALSE	
3	L6	Slider Pos	StateChangeBlock	StampSliderSensorMovedOut := FALSE	Sta

FI Injection Time	Time Constraint	Init	Abort
StampSliderSensorMovedIn value change time	2,1		
StampLowered value change time	1,1		
StampUp value change time	0,9		
StampSliderSensorMovedOut value change time	1,1		

Legende
- Automatische Generierung aus Weg-Zeit-Diagramm
- Manuelle Ergänzung für Testfallgenerierung

rt Expected reaction	Expected reaction	Applikation to test	Priority
	Fault reaction expression	MainApp	1
	Fault reaction expression	MainApp	1
	Fault reaction expression	MainApp	1
	Fault reaction expression	MainApp	1

Abbildung 22: Testtabelle zur Vervollständigung der Testfälle

In der Testtabelle müssen außerdem die Informationen, die nach den Randbedingungen RB 1, RB 2 und RB 4 definierten notwendigen Operationen für die Testdurchführung ergänzt werden. Dies sind die Init-, EvaluateReaction- und Abort-Funktionen. Dies erfolgt durch Codeabschnitte, die im Editor definiert und in der Testtabelle referenziert werden können.

Hierfür sind folgende Spalten in der Testtabelle vorgesehen:

- *Init*: Referenz auf eine vordefinierte Funktion. Falls eine mehrzyklische Ausführung notwendig ist, soll in dem Codeschnipsel die Variable OK für die Ausführungskontrolle verwendet werden. Solange „OK = FALSE“, wird das Codeschnipsel aufgerufen. Die Testfallausführung wird pausiert und erst dann fortgesetzt, wenn „OK = TRUE“ wird. Namens-Konflikte mit der Variable OK sind ausgeschlossen.
  Bsp.: TestInit(); OK:= TestInit.xDone;

- *Abort*: Referenz auf eine vordefinierte Funktion. Falls eine mehrzyklische Ausführung notwendig ist, soll die Variable OK für die Ausführungskontrolle in dem Codeschnipsel verwendet werden. Solange „OK = FALSE“, wird das Codeschnipsel aufgerufen. Die Testfallausführung wird pausiert und erst dann fortgesetzt, wenn „OK = TRUE“ wird. Namenskonflikte mit der Variable OK sind ausgeschlossen.
  Bsp.: TestResetSemiAuto(); OK:= TestResetSemiAuto.xDone;

- *Expected Reaction*: Referenz auf eine vordefinierte Funktion. In dem Codeschnipsel sollen die Variablen PASS und/oder FAIL verwendet werden. Das Codeschnipsel wird während der Fehlerinjektion ausgeführt. Das Ausführungsergebnis ist PASS/FAIL oder noch kein PASS oder FAIL (bei dieser Ausführung wurde nicht PASS oder FAIL gesetzt). *Bsp.: StopStamp(); PASS:= Stopped;*

  - PASS - beobachtete Reaktion entspricht der erwarteten (durch das Codeschnipsel spezifizierten) Reaktion.
  - FAIL - beobachtete Reaktion entspricht der erwarteten (durch das Codeschnipsel spezifizierten) Reaktion nicht.

- *Expected Alarm*: Hier wird eine Boolean Expression erwartet. Wird kein besonderer Alarm bei einem bestimmten Testfall erwartet, kann hier auch TRUE angegeben werden.
- *Application*: Dies ist das Programm, welches getestet werden soll. Das Programm muss in jedem Fall angegeben werden.
- *Priority*: Durch -1 werden Testfälle deaktiviert. Alle anderen Testfälle werden aufsteigend, beginnend mit der 0, abgearbeitet. Dies gibt dem Anwender die Möglichkeit, die Testfälle zu priorisieren.

2.4.2.1 Risikobasierte Testfallselektion

Da die Spezifikation nach der Anforderungsanalyse und Anforderung A8 als optionales Element in der ZuMaTra-Vorgehensweise enthalten ist, wurde ein Leitfaden für die Einstufung der Risikopriorität von Fehlern, welche von der Steuerungssoftware behandelt werden müssen, entworfen (siehe Anhang CD: FMEA Leitfaden). Die Einstufung erfolgt hierbei in Form einer Prozess-Failure Mode and Effects Analysis.

Der Aufbau orientiert sich hierbei an der VDA 4.2 „Sicherung der Qualität vor Serieneinsatz System FMEA“ nach Bertsche [VDA4.2]. Ein Beispiel für eine derartige FMEA ist in Abbildung 23 dargestellt und wird im Leitfaden ausführlicher erläutert.

Wurde eine FMEA durchgeführt, kann die Priorität der Testfälle systematisch festgelegt werden, indem in der Testtabelle das Feld Priority verändert wird. Bei einer hohen Risikoprioritätszahl (wie in der Abbildung 23 z.B. 90), wird eine hohe Priorität angegeben (z.B. 1). Die Testfälle werden bei der Abarbeitung nach Priorität geordnet (von 1 aufsteigend).

Type/Model/Fabrication/Load; System Structure; System Element; Item Code, Responsible; Company; Created; Modified								
Potential effect(s) of failure	**S**	**Potential failure mode(s)**	**Potential Causes**	**Preventive actions**	**O**	**Detection Actions**	**D**	**RPN**
System element: Sortierstrecke								
Function: sortieren								
Werkstück niO	8	Fehler Endlagensensor → **Fehler ausstoßen** → → **Keine korrekte Sortierung WS**	Falsche Auswahl Sensor, Betriebs-bedingungen, Abnutzung	Regel-mäßige Wartung	3	Endlagensensor-überwachung	3	72
		Sensorfehler optisch → **Keine Erkennung WS** → → **keine korrekte Sortierung WS**			3	Zeitmessung WS ab Pos1 und 2	3	72
Sicherheits-risiko	10	Kein Anhalten bei Nothalt gedrückt	Leiter defekt	Regel-mäßige Wartung	1	Wartung	9	90

Abbildung 23: Beispiel für eine FMEA (ausführlichere Beschreibung siehe Anhang CD: FMEA Leitfaden)

2.4.3 Automatisches Echtzeit-Testfallausführungsverfahren (Abbildung 20, Nr. 3)

Neben dem tatsächlichen Testobjekt, dem sogenannten „System Under Test“ (*SUT*), werden zwei weitere Testkomponenten (*TestComponent*) für die Testdurchführung benötigt. Dies ist zum einen das Testsystem und zum anderen die Komponente, gegen die getestet wird.

Die Testfälle für den Test von Fehlerbehandlungen durch Fehlerinjektionen haben den folgenden Aufbau:

- *TestComponent – Testsystem mit Testbausteinen*: Das Testsystem setzt sich durch den CODESYS Test Manager und den generierten Testcode aus dem AIS-Editor zusammen. Der Test Manager generiert zusätzlich Code für die Ausführung und das Logging der Testfälle, der Testcode wird anwendungsspezifisch erstellt, liegt als POU im Projekt vor und injiziert den tatsächlichen Fehler.
- *SUT – Steuerungscode*: Für den Test der Applikation werden einige Funktionen dieser genutzt und es werden vordefinierte Variablen erwartet, die beobachtet und geloggt werden. Zum einen sind dies Funktionen wie die Grundstellungsfahrt oder Abbruchroutinen und zum anderen die Beobachtung der Fehlerreaktionen.
- *TestComponent – Reale HW, Maschine*: Falls eine Simulation der Maschinen vorhanden ist, kann diese genutzt werden, ansonsten ist eine Ausführung der Testfälle an der realen Hardware mit dem Test Manager möglich.

In Abbildung 24 werden zunächst die Prepare-Funktionen durch den CODESYS Testmanager durchgeführt. Die Prepare-Funktionen werden im Testmanager konfiguriert und führen Aktionen wie die Generierung des Codes, der für die Ausführung der Testfälle und die Dokumentation dieser notwendig ist, durch.

Da für die Ausführung der Fehlerinjektion ein definierter Zustand gegeben sein soll, wird zu Beginn eine Initialisierung durchgeführt. Hier soll eine Unterscheidung zwischen automatischer und semi-automatischer Initialisierung gemacht werden. Bei der automatischen Initialisierung wird davon ausgegangen, dass das Programm bei Start einen Initialisierungsvorgang durchführt. Bei der semi-automatischen Initialisierung wird eine Freigabe durch den Operator erwartet. So hat dieser ggf. Zeit Maßnahmen für die korrekte Initialisierung bzw. Grundstellungsfahrt zu ergreifen. Wurde die Initialisierung abgeschlossen, wird das Programm durchlaufen, bis die Vorbedingung für die Fehlerinjektion gegeben ist. Sobald diese erkannt wird, wird der Fehler injiziert und der weitere Ablauf beobachtet. Wird die korrekte Fehlermeldung und die korrekte Fehlerbehandlung eingeleitet, wird der Testfall mit dem Ergebnis „erfolgreich“ beendet. Überschreitet die erwartete Zeit einen Wert, wird der Testfall abgebrochen. Hierfür wird eine Abbruchroutine benötigt, die von dem Anwender definiert wird. Auch hier soll die Möglichkeit gegeben werden, eine automatische oder eine semi-automatische Abbruchroutine zu starten. Bei der Automatischen wird lediglich eine Methode oder Variable aus dem Programm getriggert, bei der semi-automatischen Ausführung wird eine Bestätigung erwartet. In Abbildung 24 ist ein semi-automatischer Testfall dargestellt.

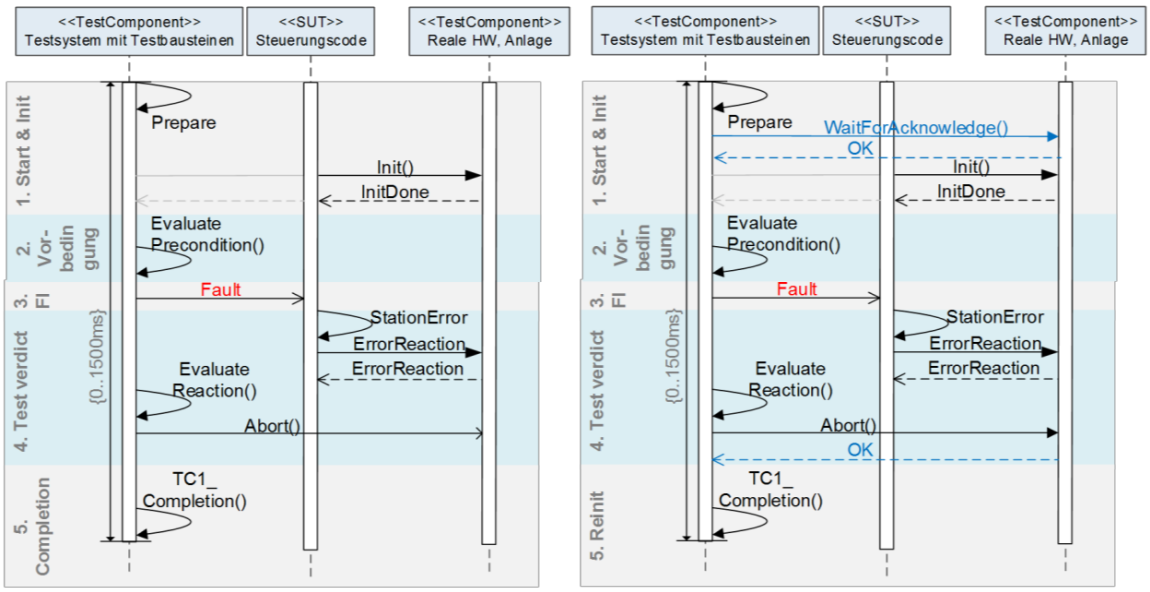


Abbildung 24: Aufbau der Testfälle (links) und semi-automatische Ausführung der Testfälle (rechts).

2.4.3.1 Dokumentation der Testfälle

Um die Anforderung A5 zu erfüllen, wurde über die Dokumentation, ob der Test erfolgreich war, noch eine spezifische Einordnung, warum ein Testfall ggf. nicht erfolgreich war, in der Dokumentation ergänzt. Die Testfälle geben nach Ausführung folgendem Schema Rückmeldung über deren Status:

Ausgabe	**Bedeutung**
Error detected: TRUE/ FALSE	Bei TRUE wurde der Alarm detektiert
Error-Code: 1	Die Vorbedingung Teil 1 wurde nicht erreicht
Error- Code: 2	Die Vorbedingung (FI injection Time) wurde nicht erreicht

Error- Code: 3	Die Fehlerreaktion wurde nicht beobachtet
Error- Code: 4	Zeitüberschreitung für Init
AbortErrorDetected (5)	Der Abort wurde nicht in der vorgegebenen Zeit ausgeführt
Faultreaction: PASS	EvaluateReaction war erfolgreich
Faultreaction: FAIL	EvaluateReaction war nicht erfolgreich
Faultreaction: TIMEOUT	EvaluateReaction war wegen der Zeitüberschreitung nicht erfolgreich

Abbildung 25: Dokumentation der Testfälle

2.5 Software-Funktionsmuster als Basis der Evaluierung

2.5.1 Konzeption und Implementierung des Funktionsmusters

Zur Anwendung der Vorgehensweise können unterstützende Werkzeuge eine erhebliche Effizienzsteigerung bewirken oder sogar erst ermöglichen. Vor allem im Bereich der Erstellung der Modelle, wie auch bei der Testfallgenerierung ist eine Unterstützung sinnvoll. Dazu müssen zunächst die Aktivitäten in Bezug auf mögliche Effizienzsteigerungen analysiert werden. Anschließend können diese bezüglich ihrer Machbarkeit und des Aufwands zur Umsetzung bewertet und ausgewählt werden.

Im Folgenden wird die – auf Basis dieser Anforderungen erstellte – Werkzeugunterstützung beschrieben. Zunächst werden in Kapitel 2.5.2 die einzelnen Werkzeugfunktionen identifiziert und beschrieben. Anschließend wird das Konzept der Werkzeugunterstützung – im Folgenden als „ZuMaTra-Plugin“ bezeichnet – in Kapitel 2.5.3 erläutert und die Umsetzung in Kapitel 2.5.6 beschrieben.

2.5.2 Identifikation und Beschreibung der Werkzeugfunktionen

Ziel des ZuMaTra-Plugins ist die Effizienzsteigerung im Bereich der Testfallerstellung bzw. -generierung. Fokus des Software-Funktionsmusters ist es, die ZuMaTra-Vorgehensweise effizient und intuitiv verständlich zu unterstützen.

Ein Überblick über die hierzu zu realisierenden Funktionalitäten ist in Abbildung 26 aufgezeigt. Die Hauptfunktionalitäten des ZuMaTra-Plugins sind folgende:

Realisierung der ZuMaTra-Vorgehensweise: Zur Unterstützung der ZuMaTra-Vorgehensweise müssen die unter Kapitel 2.4 beschriebenen Schritte unterstützt werden. Der Nutzer des Editors muss hierzu die Modellierungselemente des Weg-Zeit-Diagramms modellieren und die Testtabelle spezifizieren können. Eine anschließende Testfallgenerierung und ebenso die Ausführung der Testfälle muss adäquat unterstützt werden.

Realisierung von Editor-Funktionen: Zur Effizienzsteigerung bei der ZuMaTra-Vorgehensweise müssen Standard-Editor-Funktionalitäten bereitgestellt werden. Durch das ZuMaTra-Plugin muss die grafische Erstellung und Parametrierung der Elemente möglich sein. Um Modellierungsprojekte zu einem Zeitpunkt unterbrechen und später wieder fortsetzen zu können, müssen Modelle gespeichert und geladen werden können.

Einbindung in die Werkzeugkette: Um den Editor in die unternehmensspezifische Werkzeugkette integrieren zu können, sind entsprechende Austauschformate anzubieten. Hierzu wird für die Übergabe der Testfälle an eine Programmierumgebung für die Testausführung das PLCopen XML-Format gewählt. Des Weiteren soll zu Dokumentationszwecken der Testreport so ausführlich wie möglich erfolgen. Hierfür wurde eine *sInfo*-Variable angelegt, die durch den CODESYS-Testmanager automatisch ausgewertet, ggf. aber auch manuell abgefragt werden kann.

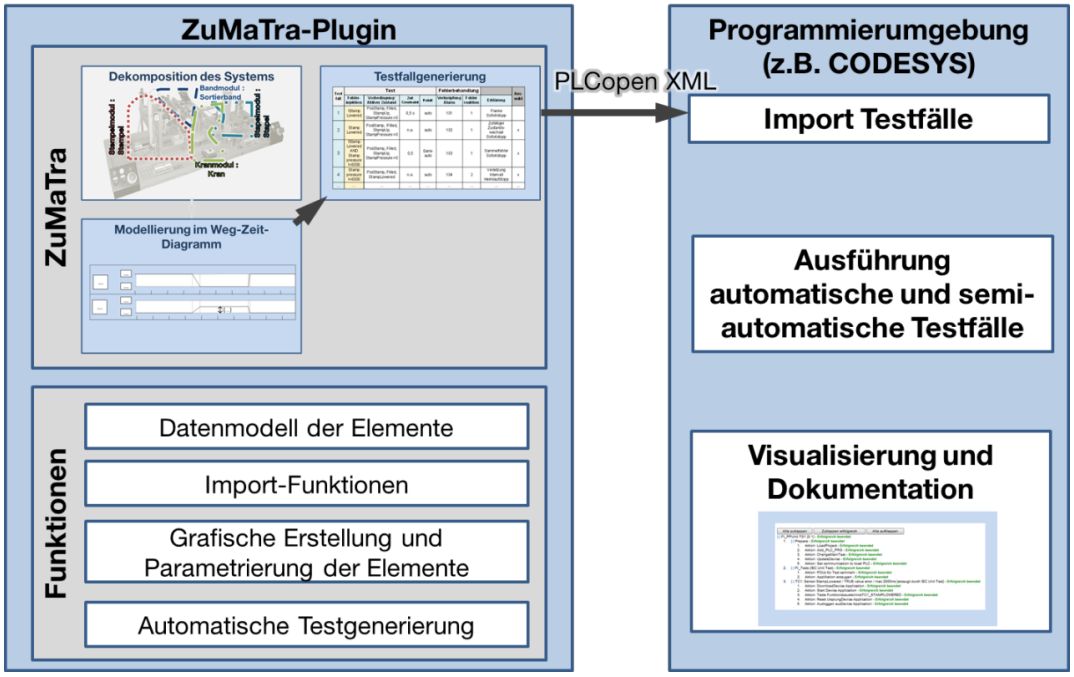


Abbildung 26: Überblick über das ZuMaTra-Plugin

2.5.3 Aufbau des Editors

Zur Realisierung der zuvor identifizierten Funktionalitäten wird im Folgenden das Konzept des ZuMaTra-Plugins erläutert.

Der Aufbau des ZuMaTra-Plugins ist in Abbildung 27 dargestellt. Neben einer Symbolleiste zur *Erstellung der ZuMaTra-Modellierungselemente*, in welcher der Nutzer des Editors die zu erstellenden Elemente auswählen kann, gibt eine Baumansicht einen *Überblick über die Elemente des Modells*. Innerhalb des Diagrammbereichs können somit Elemente im *Zeichenbereich* platziert und im *Eigenschaftsbereich* bearbeitet werden.

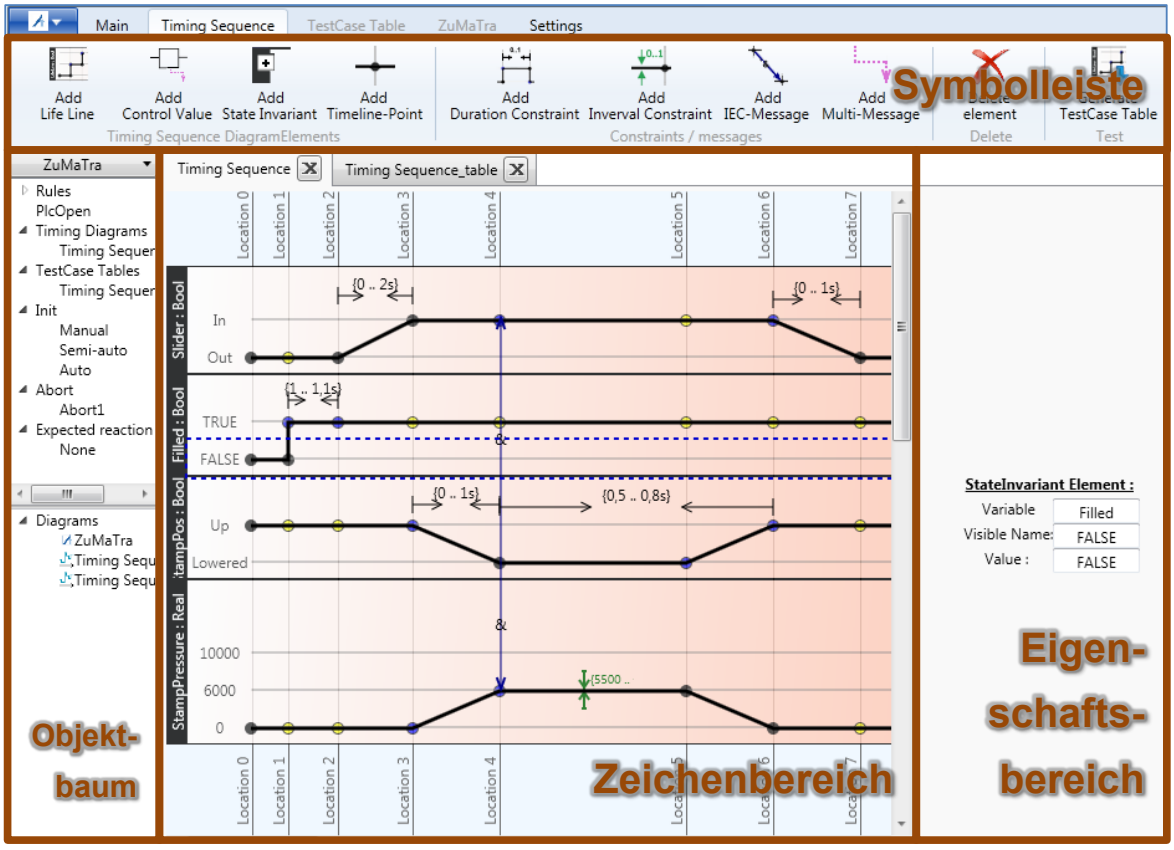


Abbildung 27: Übersicht ZuMaTra-Plugin

2.5.4 Codeanalyse im Editor

Für die Codeanalyse wird das ZuMaTra-Diagramm im Editor bereitgestellt. Eine Schnittstelle für den Import von PLCopen XML-Dateien ermöglicht das Importieren von Projekten beziehungsweise Funktionsbausteinen aus TwinCAT und CODESYS.

Die Funktionsbausteine können dann im Objektbaum ausgewählt und direkt als Text angezeigt werden. Zur Generierung des Kontrollflusses werden in der Menüleiste entsprechende Buttons dargestellt.

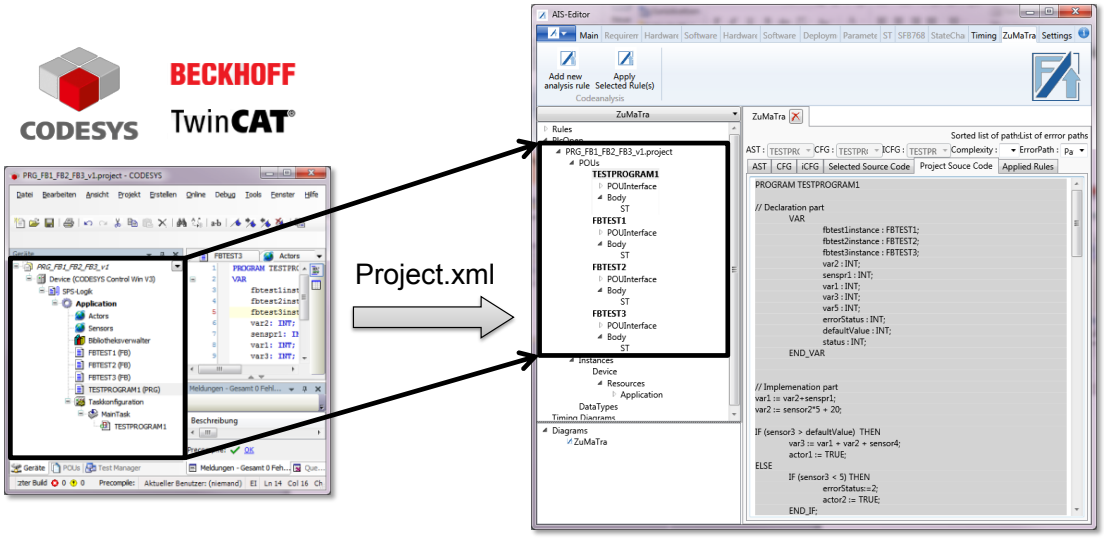


Abbildung 28: PLCopen XML-Import (CODESYS-/TwinCAT3-Format) in das ZuMaTra-PlugIn

Im Objektbaum können die Regeln zur Überprüfung des Kontrollflusses angegeben werden. Per Rechtsklick kann eine neue Regel eingegeben werden. Im Editor steht dann das im Konzept entworfene Template mit entsprechenden Feldern zur Verfügung:

- **Target:** Modell, das untersucht werden soll (CFG oder iCFG)
- **Paths:** Pfade die untersucht werden sollen
- **PathCondition:** Bedingung die überprüft werden soll

2.5.5 Das Weg-Zeit-Diagramm, die Testfallgenerierung und Testtabelle im Editor

Um die Modellierung im Editor möglichst effizient und verständlich zu gestalten, sollen die ZuMaTra-Modellierungselemente möglichst exakt im ZuMaTra-Plugin abgebildet werden. Dies beinhaltet zum einen die exakte grafische Darstellung der Modellierungselemente, zum anderen die Ermöglichung der Spezifikation der Inhalte der Modellierungselemente. Mehrere Beispiele für diese Umsetzung können in Abbildung 32 oder Abbildung 36 bzw. im Leitfaden nachvollzogen werden.

Die Testfallgenerierung wurde entsprechend der in Kapitel 2.4.2 definierten Fehleroperatoren umgesetzt. Die Vorbedingung wird aus den Locations – also dem modellierten Verlauf – des Modells extrahiert.

Die manuell ergänzten Informationen in der Testtabelle werden zusammen mit den aus dem Weg-Zeit-Diagramm gewonnenen Informationen in ein PLCopen XML übersetzt.

2.5.6 Umsetzung

Die Realisierung des ZuMaTra-Plugins erfolgte unter Verwendung der Programmiersprache C# .NET in der Entwicklungsumgebung Microsoft Visual Studio. Der Leitfaden sowie ein Installer für das ZuMaTra-Plugin können unter der URL http://zumatra.ais.mw.tum.de/ bezogen werden. Als Basis für die Implementierung des ZuMaTra-Plugins wurde das in Abbildung 38 und Tabelle 2 vorgestellte Metamodell hinzugezogen.

2.6 Evaluierung der Konzepte

Um einerseits die Machbarkeit und andererseits die Anwendbarkeit der ZuMaTra-Codeanalyse, Modellierung und Testgenerierung und -ausführung für Industriebeispiele zu überprüfen,

wurden ein Anwendungsbeispiel aus der Industrie im Rahmen einer Machbarkeitsstudie und ein Anwendungsbeispiel im Rahmen eines Workshops gemeinsam mit den Teilnehmern erarbeitet. Die Codeanalyse wurde anhand einiger Bausteine mit begrenzter Komplexität evaluiert.

2.6.1 Evaluierung der Codeanalyse

Die Regeln wurden entsprechend Anforderung A9 entworfen und an einigen Beispielen getestet, wie beispielsweise dem hier dargestellten:

```
PROGRAM testRule
VAR
        i1: INT;
        i2: INT;
        i3: INT;
 END_VAR
var1 := var2+senspr1;
var2 := sensor2*5 + 20;
IF (sensor3 > defaultValue)  THEN
        errorStatus:=1;
        var3 := var1 + var2 + sensor4;
        actor1 := TRUE;
ELSIF (sensor3 < 5) THEN
        errorStatus:=2;
        actor2 := TRUE;
END_IF;
```

```
CASE status OF
1, 5:
        var5 := var1;
        actor3 := TRUE;
        errorStatus:=3;
2:
        var3 := var1 + var2 + sensor4;
        actor1 := FALSE;
        actor2 := FALSE;
        errorStatus:=4;
10..20:
        var3 := sensor4;
        actor1 := FALSE;
0..11:
        var3 := var2 + sensor4;
        actor1 := FALSE;
        errorStatus:=5;
END_CASE;
        var3 := var1 + var2 + sensor4;
        actor4 := TRUE;
        actor1 := FALSE;
END_PROGRAM
```

Der Funktionsbaustein wurde per PLCopen XML aus CODESYS exportiert und in das ZuMaTra-Plugin importiert (Menü Load PLCopen XML). Durch Auswahl des Menüpunkts „Generate AST(s)/ CFG(s)“ werden der abstrakte Syntaxbaum und der Kontrollfluss des Funktionsbausteins generiert.

In Abbildung 29 ist der Kontrollfluss und die Regelüberprüfung der Regel „errorStatus muss in jedem möglichen Pfad geschrieben werden“ dargestellt. In dem einfachen dargestellten Kontrollfluss ist ein Pfad, bei dem diese Regel verletzt wird, rot markiert.

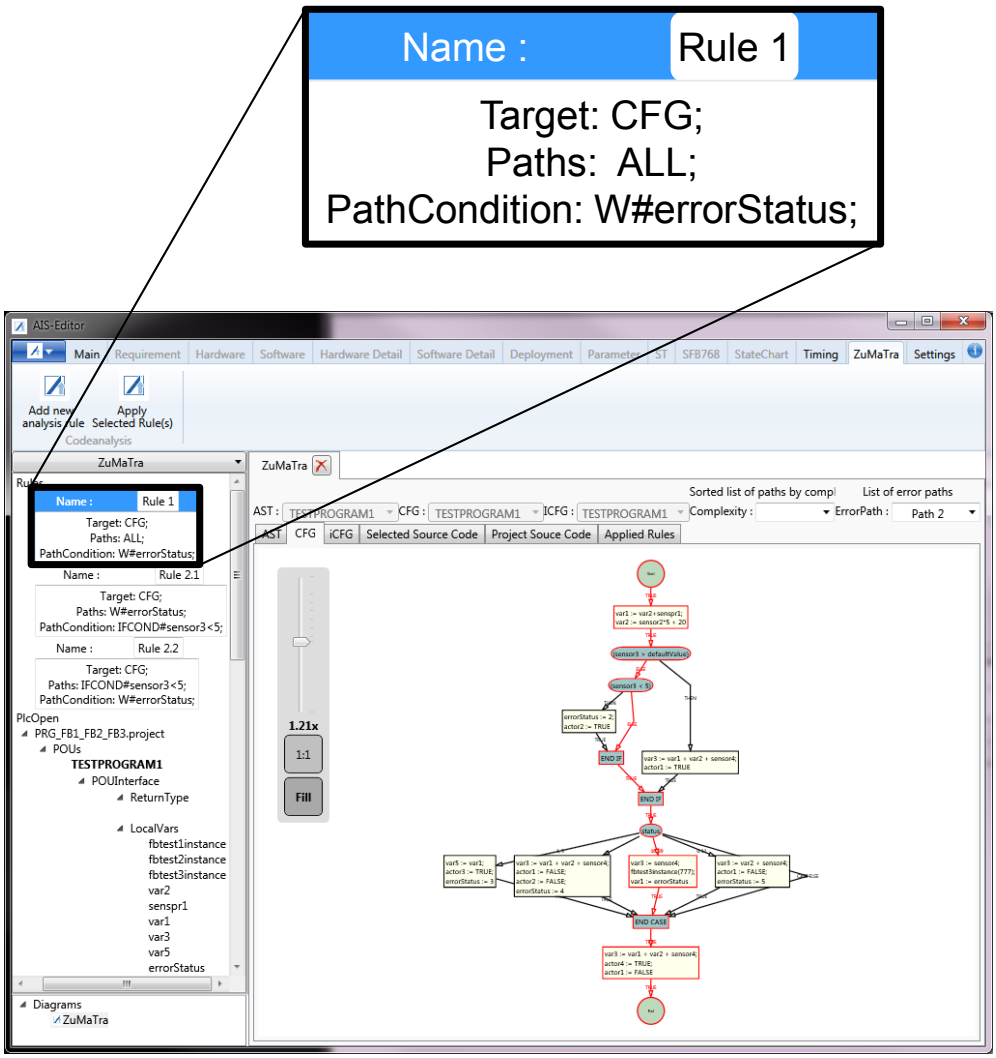


Abbildung 29: Regel 1 als Beispiel für die Regeln im ZuMaTra-PlugIn

Der grundsätzliche Nachweis der Machbarkeit konnte durch das Funktionsmuster erbracht werden. Die Skalierbarkeit der Pfadanalyse und der Umgang mit Komplexität ist jedoch noch eine offene Frage, da für den Funktionsnachweis die Anzahl der Pfade auf 100.000 beschränkt wurde, was für aktuelle Anwendungsbeispiele aus der Industrie nicht immer ausreichend ist. Die Optimierung der Algorithmen zur Pfadanalyse müssen daher ggf. verbessert werden oder sehr hohe Rechenzeiten in Kauf genommen werden.

Neben dem Nachweis für strukturierten Text konnte in einer Bachelorarbeit der Nachweis der Machbarkeit für Funktionsblockdiagramme erbracht werden [Ta14]. Hier ist hervorzuheben, dass der Ansatz für komplexe Funktionsblockdiagramme erbracht werden konnte, da Funktionsblockdiagramme in der Regel deutlich weniger Verzweigungen im Kontrollfluss aufweisen.

2.6.2 Evaluierung der Vorgehensweise für den Test von Fehlerbehandlungsroutinen an einem Anwendungsbeispiel mit kontinuierlichen Prozessen

2.6.2.1 Das Anwendungsbeispiel Pflasterverarbeitung

Die von Harro Höfliger zur Verfügung gestellt Maschine für die Machbarkeitsstudie ist in Abbildung 30 dargestellt. Zusätzlich wurde ein SoMachinMotion-Programm für den Betrieb der

Maschine zur Verfügung gestellt. Fehler die an der Maschine behandelt werden müssen, sind zum einen Fehler die von den Antrieben kommen, zum anderen die Regelung bzw. die Abweichung von der Tänzerposition bei der Abwicklung und bei der Aufwicklung wie z.B. eine zu starke Abweichung von der Mittelposition bei verschiedenen Zuständen – Anlauf, Betrieb, Stopp, Kalibrierung.

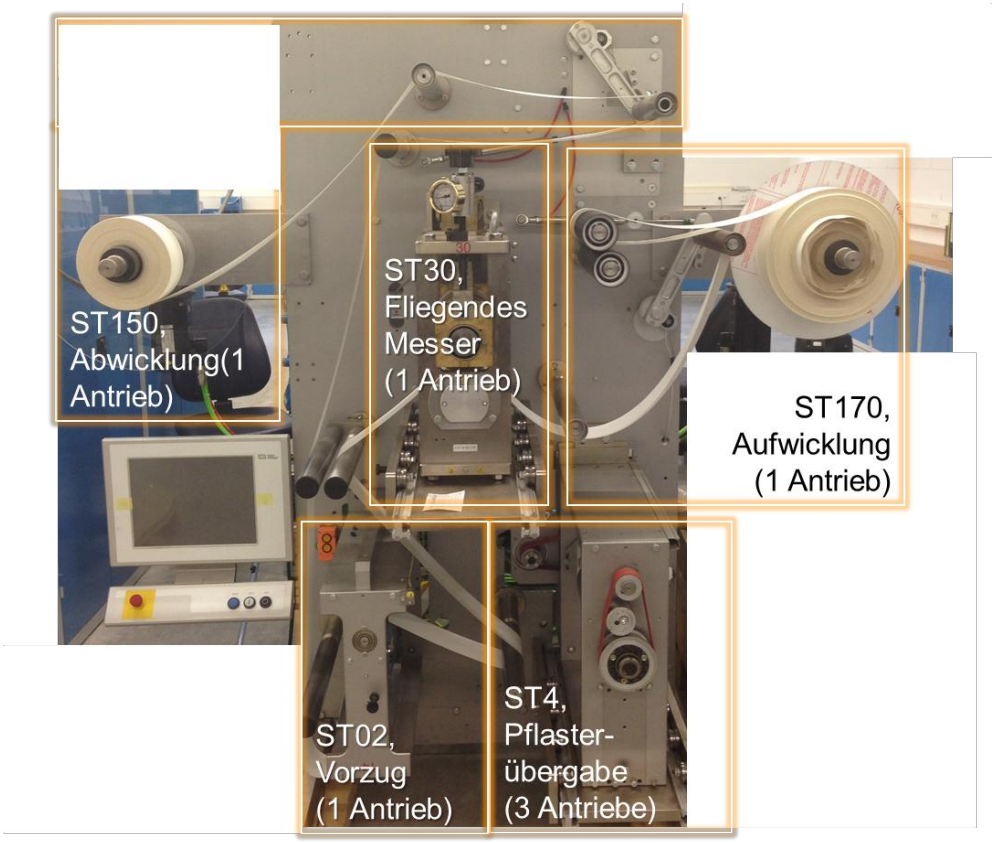


Abbildung 30: Anwendungsbeispiel eines kontinuierlichen Prozesses

2.6.2.2 Vorgehen bei der Machbarkeitsstudie

Nach einer intensiven Analyse des Anwendungsbeispiels wurden diverse Testfälle entworfen, bzw. die Methodik noch einmal iterativ nach diversen Fehlversuchen angepasst.

Test der Reaktion der Maschine auf Antriebsfehler

Es wurde schnell klar, dass für den Test der Antriebe ein Entwurf im Weg-Zeit-Diagramm keinen Sinn macht, da auf Applikationsebene lediglich die richtige Konfiguration getestet werden muss.

Die genaue Fehlermeldung liefert die Diagnose des Antriebsherstellers. Dementsprechend wurde eine Fehlerursache, die sich für die Fehlerinjektion bei Antrieben eignet – der Schleppfehler – identifiziert. Ziel ist es, dass der physikalische Zustand der Maschine mit dem in der Steuerung identifizierten Zustand nicht abweicht, da es sonst zu gefährliche Zuständen kommen kann.

In Abbildung 31 sind die für die Antriebe entworfenen Testfälle dargestellt. Einzige Voraussetzung für einen sinnvollen Test ist, dass die Antriebe laufen. Demzufolge wird dies als Vorbedingung für die Testfälle angegeben. Die Anforderung, dass der physikalische Zustand dem erkannten Zustand in der Steuerung entspricht kann durch die Injektion eines Schleppfehlers,

durch Herabsetzung des Nutzstroms erreicht werden. Dementsprechend ist die Fehlerinjektion die Herabsetzung des Stroms eines Antriebs (vgl. FI Code Abbildung 31).

Die dargestellten Testfälle wurden erfolgreich an der Maschine erprobt. In einem Testlauf wurden 5 von 7 Testfällen erfolgreich durchgeführt. Danach konnte der Lauf nicht automatisiert fortgeführt werden, da die Maschinenparametrierung durch dem 5. Testlauf nicht mehr korrekt war und eine Neuparametrierung erforderlich wurde.

TC Nr	Precondition	Life Li	TC Type	FI Code	FI InjectiTime C	Init	Abort	Expected reaction	Expected alarm	Application	tPriority
0	MC_1002T4_1_A.RefVelocity > 5 OR MC_1002T4_1_A.RefVelocity < 5	-	FaultInjectionAxis	MC_1002T4_1_A. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_002_T4_1_A_PULLROLLS_AXIDX]	OB1_Task1	1
1	MC_1004T2_1_A.RefVelocity > 5 OR MC_1004T2_1_A.RefVelocity < 5	-	FaultInjectionAxis	MC_1004T2_1_A. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_004_T2_1_A_3Zug_Transfersegment1_AXIDX]	OB1_Task1	1
2	MC_1004T2_1_B.RefVelocity > 5 OR MC_1004T2_1_B.RefVelocity < 5	-	FaultInjectionAxis	MC_1004T2_1_B. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_004_T2_1_B_3Zug_Transfersegment2_AXIDX]	OB1_Task1	1
3	MC_1004T3_1_A.RefVelocity > 5 OR MC_1004T3_1_A.RefVelocity < 5	-	FaultInjectionAxis	MC_1004T3_1_A. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_004_T3_1_A_3Zug_Transfersegment3_AXIDX]	OB1_Task1	1
4	MC_1030T1_1.RefVelocity > 5 OR MC_1030T1_1.RefVelocity < 5	-	FaultInjectionAxis	MC_1030T1_1.UserCurrentLimit:=1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_030_T1_1_DRIVECUTTINGCYLINDER_AXIDX]	OB1_Task1	1
5	MC_1150T4_1_B.RefVelocity > 5 OR MC_1150T4_1_B.RefVelocity < 5	-	FaultInjectionAxis	MC_1150T4_1_B. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_150_T4_1_B_UNWINDER_AXIDX]	OB1_Task1	1
6	MC_1170T5_1_A.RefVelocity > 5 OR MC_1170T5_1_A.RefVelocity < 5	-	FaultInjectionAxis	MC_1170T5_1_A. UserCurrentLimit: =1;	0	3	Semi-auto	Abort1	MasterQStop	AxisError [ST_170_T5_1_A_REWINDER_AXIDX]	OB1_Task1	1

Abbildung 31: Testfälle für den Test von Antriebsfehlern

Test der Reglerbausteine

Im Gegensatz zu diskreten Prozessen stellte die Modellierung kontinuierlicher Systeme eine erheblich höhere Anforderung an die Modellierung im Weg-Zeit-Diagramm. Für die Regelung ist lediglich ein Sensorwert – die Tänzerposition – relevant. Dennoch gibt es einige komplexe Abläufe zu beachten, welche durch weitere Steuervariablen gesteuert werden. Bei der Analyse der Bausteine wurden 12 verschiedene Zustände des Reglerbausteins identifiziert, inklusive Kalibrierung, Wartezuständen, Überprüfungen der korrekten Position, Start-Zustand, Automatikmodus, etc.

Um verschieden Szenarien, die auch durch Fehlerbehandlungen überprüft werden müssen, abzubilden, kam daher insbesondere das Modellierungselement „ControlValueLifeline“ zum Einsatz (vgl. Abbildung 32). Weiterhin musste das Modellierungselement „IntervalConstraint“ angepasst werden, da Anforderungen wie „es darf bei dem Automatikbetrieb zu einer maximalen Abweichung von 5 % zur Mittelposition kommen“ nur durch die Angabe einer Variablen und nicht durch die Angabe einer festen Zahl ausdrückbar sind. Bei der Durchführung wurde festgestellt, dass die Vorbedingung teilweise noch zu eng gefasst sind, da die in Abbildung 32 dargestellte Vorbedingung, dass das gültige Intervall innerhalb von 6 Sekunden verlassen wird, nicht eingehalten werden kann.

Mit einer manuellen Anpassung der Vorbedingung konnten erfolgreich Testfälle durchgeführt werden. Eine weitere Untersuchung mit dem neu definierte „don’t care“ Element (siehe folgender Abschnitt) könnte das Problem ggf. auch beheben, eine Untersuchung steht hier jedoch noch aus.

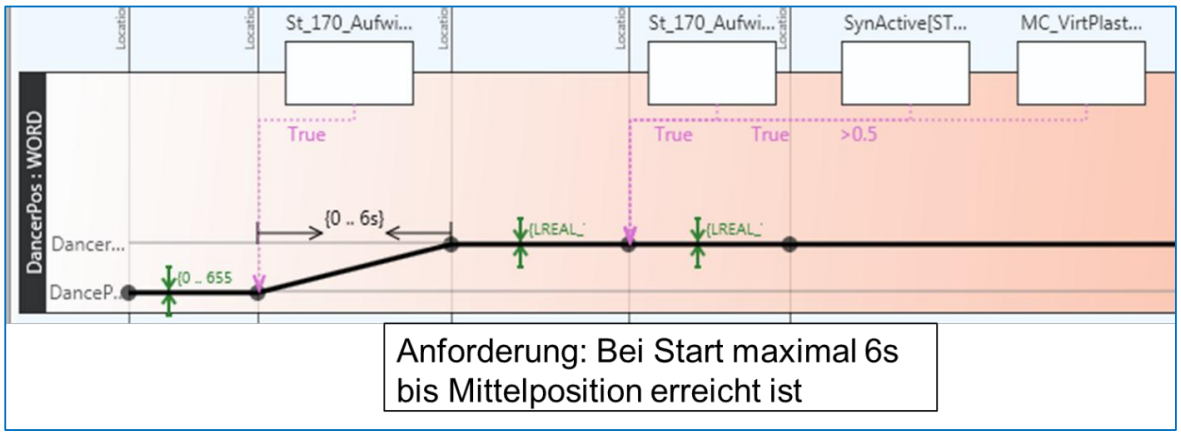


Abbildung 32: Modellierung der Soll-Position des Tänzers

Festhalten lässt sich, dass die grundsätzliche Machbarkeit gezeigt werden konnte. Insbesondere war erkenntlich, dass für die Modellierung kontinuierlicher Systeme andere Modellierungselemente wichtig sind, als bei der Modellierung diskreter Prozesse.

Der Nutzen-Faktor eines modellbasierten Ansatzes konnte ebenfalls noch nicht nachgewiesen werden, da hier vermutlich noch weitere Anpassungen und Nachweise notwendig sind.

2.6.3 Evaluierung der Vorgehensweise für den Test von Fehlerbehandlungsroutinen in einem Workshop an einem Anwendungsbeispiel mit diskreten Prozessen

2.6.3.1 Das Anwendungsbeispiel Trainingsstation

Als Anwendungsbeispiel wurde für den Workshop von Bosch die in Abbildung 33 dargestellte Maschine zur Verfügung gestellt. Weiterhin stand für das Evaluationsbeispiel ein fertig implementiertes TwinCat 3-Programm zur Verfügung.

Da entsprechend ohne den CODESYS Test Manager gearbeitet werden musste, wurde weiterhin ein Programm für die Ausführung, Visualisierung und Dokumentation erarbeitet, was einen zusätzlichen Aufwand von ca. einem Tag bedeutete.

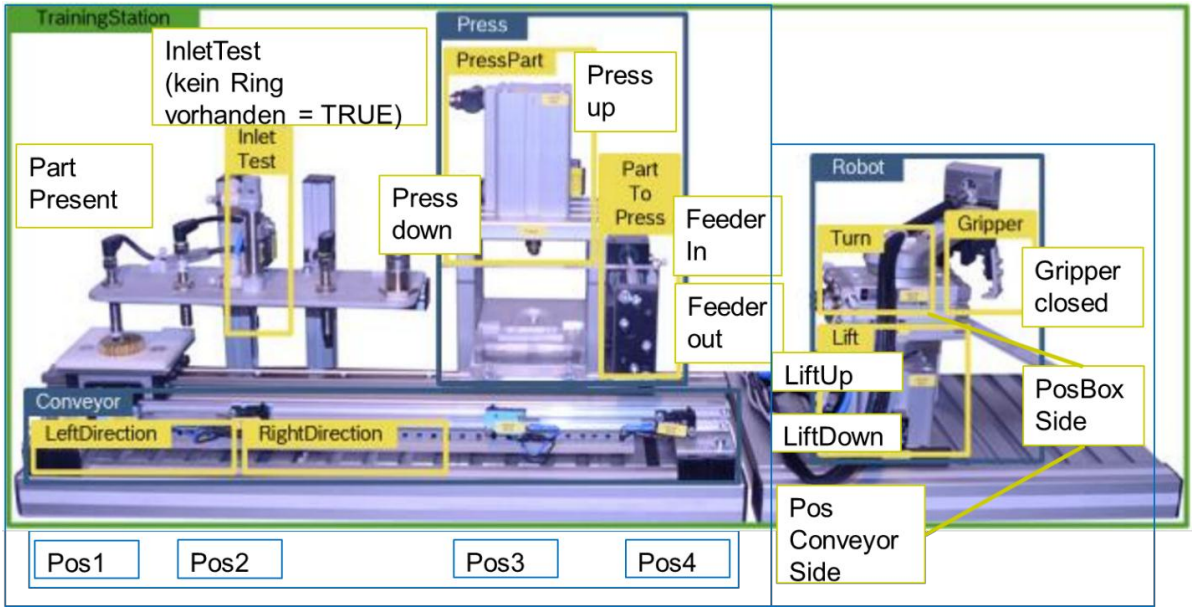


Abbildung 33: Trainingsstation Bosch

2.6.3.2 Workshop-Durchführung und Ergebnisse

Der Workshop wurde anhand folgender Vorgehensweise durchgeführt:

Vorstellung und Diskussion des Leitfadens: Anhand eines Leitfadens wurde die ZuMaTra-Modellierungsmethodik und Vorgehensweise vorgestellt und diskutiert. Als Beispiel wurde die Sortier- und Stempelanlage mit Fokus auf den Stempel gewählt. Der Leitfaden kann im Anhang (Anhang CD: Leitfaden ZuMaTra) entnommen werden.

Gemeinschaftliche Erarbeitung der Anwendungsbeispiele: Das zuvor definierte Anwendungsbeispiel wurde anhand von definierten Aufgaben einzeln von den Workshop-Teilnehmern erarbeitet. Neben der Aufgabenstellung wurde den Teilnehmern eine Spezifikation der Code-Schnipsel zur Verfügung gestellt (Anhang CD: Zusatzmaterial Workshop Bosch).

Durchführung der Testfälle an der Trainingsstation: Die Testfälle wurden an der Trainingsstation erprobt.

Diskussion der Ergebnisse: Die Ergebnisse des Workshops wurden in einer Fokusgruppe diskutiert. Hierbei wurde zum einen auf die Anwendbarkeit der Vorgehensweise und Modellierung allgemein, zum anderen auf die Verbesserung der Anwendbarkeit durch eine Verbesserung der Werkzeugunterstützung eingegangen.

Allgemeine Ergebnisse:

Insgesamt nahmen an dem Workshop 7 Experten teil. Davon waren 5 Anwender, 1 Komponentenhersteller und 1 Plattformentwickler. Die Schulung nahm ca. 1,5 Stunden in Anspruch, die Erarbeitung und Durchführung des Beispiels ca. 4 Stunden und die Expertendiskussion ca. 45 min. Von den 7 Teilnehmern gab es eine 2er-Gruppe, es wurden also 6 Beispiele erarbeitet. Von 6 Beispielen konnten 2 erfolgreiche Testläufe durchführen. Bei 2 weiteren war die Zeit, auch aufgrund von Problemen mit dem Editor, zu knapp und bei den anderen war ein Fehler an der Modellierung für den Fehlschlag der Testfälle verantwortlich. Die Ursachen hierfür wurden in der Expertendiskussion vertieft.

2.6.3.3 Ergebnisse der Expertendiskussion

Allgemein wurde festgehalten, dass das Weg-Zeit-Diagramm intuitiv und einfach erlernbar ist und eine gute Überschaubarkeit der Darstellung erlaubt. Auch die Ausdrucksmächtigkeit des Weg-Zeit-Diagramms und die wählbare Granularität/Abstraktion wurden positiv aufgenommen (Erfüllung der Anforderung A3).

Anforderungsbewertung durch die Experten:

Der Aufbau der Testfälle wurde als passend bewertet mit der Angabe der Init-, Abbruch- und Überprüfungsroutinen als Code-Schnipsel (Randbedingung RB 1, RB 2 und RB 4).

Die generierten Testfälle – also die definierten Fehleroperatoren – wurden als sinnvoll und den Anforderungen entsprechend eingestuft (A6 und A7). Damit lässt sich die Abdeckung von modellierten Anforderungen und die Überprüfung dieser dokumentieren und nachweisen (A5).

Die automatische Generierung und automatische bzw. semi-automatische Durchführung wurde als Effizienzsteigerung im Gegensatz zu aktuellen Vorgehensweisen bewertet.

Anforderungen für eine Weiterentwicklung der Notation, der Vorgehensweise und des Editors

Um den Anwender besser bei der Erstellung zu unterstützen gib es einige allgemeine Anforderungen, die durch eine adäquate Werkzeugunterstützung erfüllt werden sollten.

- Speichern unter; projektbezogener, standardmäßiger Dateiname beim Speichern; Button für das Speichern
- Drucken und Screen-Shots
- Bei der Erstellung eines Elements, sollte dieses gleich angewählt bleiben, damit dessen Eigenschaften im Eigenschafts-Editor bearbeitet werden können
- Das Löschen von Elementen war nicht immer fehlerfrei möglich
- „Copy & Paste“-Funktionen für die Modellierungselemente
- Sortierkriterien für Aktionen und Objekte im Objektbaum

Neben der allgemeinen Kritik zum Editor wurden folgende Verbesserungspotentiale bei der Modellierung mit dem Weg-Zeit-Diagramm festgehalten (siehe auch Abbildung 34):

- Bezüglich des Editors sind folgende Punkte zu verbessern
  - Fangbereich für „TimeLinePoints“ vergrößern
  - Gruppierung und neue Anordnung von „Lifelines“ ermöglichen
  - Verschieben und Einfügen von Locations besser unterstützen, Schriftgröße
  - Messages verschieben ermöglichen
  - Automatische Skalierung der Zeit (nach „Duration Intervals“)
- Bezüglich der Modellierung/Notation wurden folgende Punkte aufgenommen
  - „Don‘t care“-Zustände („StateInvariants“ ermöglichen)
  - Kommentar als zusätzliches Modellierungselement
  - Benennung von Locations ermöglichen
  - Modularisierung und Kapselung besser unterstützen/ermöglichen
    - Wiederverwendung einzelner „Lifelines“/ einer gruppierten Menge von „Lifelines“ und Ermöglichen einer reinen Anzeige von „Lifelines“ ohne Testfallgenerierung
  - Standardmäßige Vorgabe einer Zeitachse

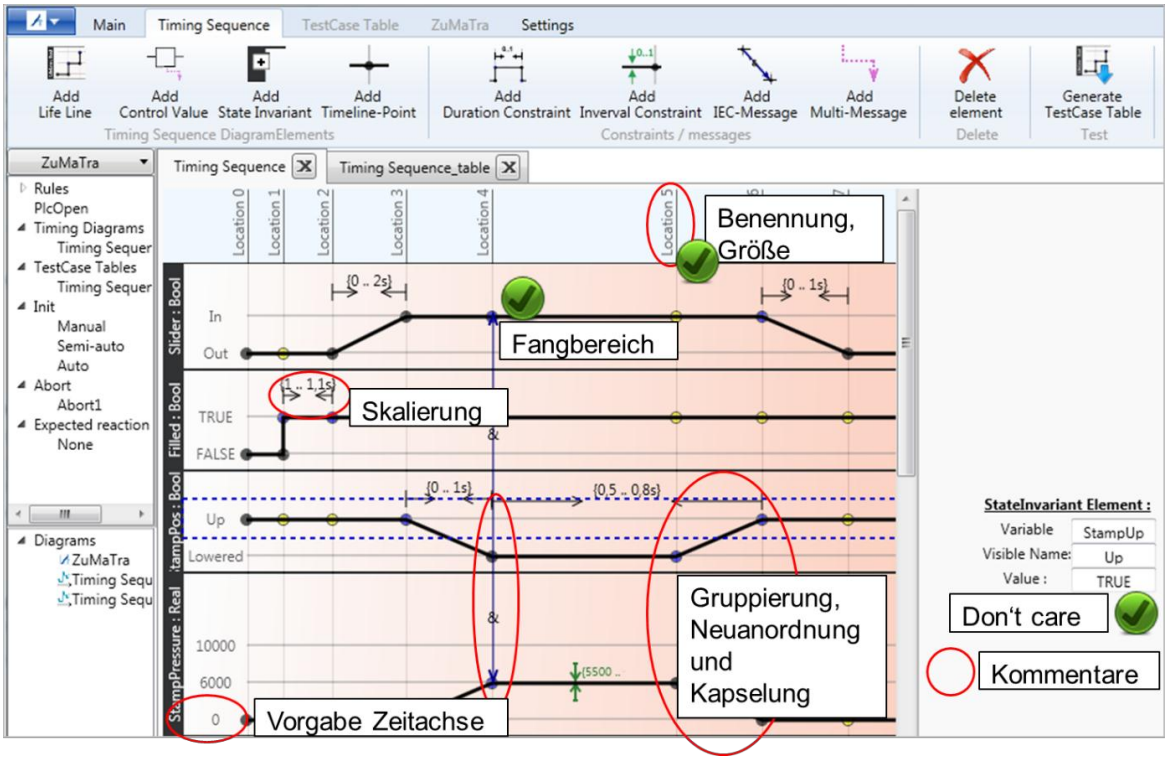


Abbildung 34: Verbesserung der Modellierung im Weg-Zeit-Diagramm

Von den Anforderungen wurden im Nachgang noch einige Punkte aufgenommen (grüne Häkchen in Abbildung 34), wie die Einführung von der Angabe von „StateInvariant“-Werten mit „don’t care“. Dies hat den Vorteil, dass der Anwender spezifizieren kann, wann bestimmte Zustände ggf. nicht relevant für eine Testfallgenerierung sind.

Bezüglich der Testfallgenerierung wurde folgendes Feedback aufgenommen:

- Einführung eines zusätzlichen Fehleroperators „&“ auf konstante „StateInvariant“-Abschnitte.
- Einführung von „Locked“-Variablen, für die eine Fehlerinjektion explizit freigegeben werden muss (sicherheitskritische Fehler bzw. Fehlerbehandlungen).

Durch den Fehleroperator werden Testfälle generiert, bei denen Verschränkungsbedingungen überprüft werden können und wurde bereits in dem Prototyp umgesetzt. Die „Locked“-Variablen sollen für sicherheitskritische Fehlerinjektionen gesetzt werden können, um beispielsweise bei kritischen Tests eine explizite Freigabe durch berechtigte Personen zu verlangen.

Zur Testtabelle wurde Folgendes festgehalten:

- Nachvollziehbarkeit Weg-Zeit-Diagramm – Testfälle/Testtabelle schwierig
  - Symbolische Bezeichnungen in Testtabelle anzeigen/verwenden
  - Doppelte Ansicht Weg-Zeit-Diagramm – Testtabelle: Optimale Lösung – bei Anwählen eines Testfalls wird dieser im Weg-Zeit-Diagramm visualisiert (wie in Abbildung 35 dargestellt)
- Testfälle die abgewählt wurden ausgrauen, nicht komplett löschen oder -1
- Kommentieren der Testfälle ermöglichen

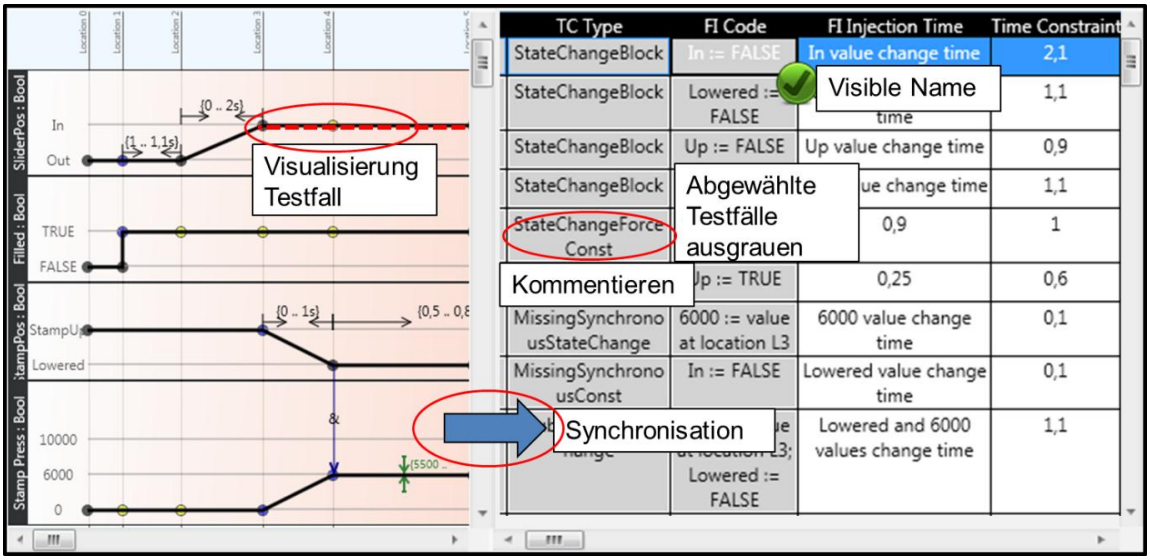


Abbildung 35: Parallele Ansicht Tabelle und Modellierung

Der Punkt der symbolischen Bezeichnung wurde in Folge noch innerhalb des Projekts umgesetzt. Eine der wichtigsten Anforderungen für eine Entwicklung zum Produkt muss mit der Synchronisation des Weg-Zeit-Diagramms mit der Testtabelle hervorgehoben werden. 2 der Testläufe scheiterten an dieser fehlenden Funktion, da die Testtabelle bei einer Neu-Generierung aus dem Weg-Zeit-Diagramm komplett neu generiert wird. Dadurch gehen bereits ergänzte Informationen in der Testtabelle vollständig verloren, was einen erheblichen Mehraufwand bedeutet.

Eine weitere erhebliche Verbesserung der Anwendbarkeit wurde einer engeren Integration mit Programmierumgebungen prognostiziert. Der Import von Variablen bzw. das Einlesen von Variablenlisten und Auswahl mit Drop-Down-Menüs in den Eigenschaftsbereichen würde die Datendurchgängigkeit erhöhen und dementsprechend Fehler bei der Spezifikation vermindern.

Zusammenfassend konnte durch den Workshop das Potential, die Machbarkeit des Ansatzes und die Lücke zu einer Einführung in die Industrie eindeutig aufgezeigt werden.

2.6.4 Beantwortung eines Fragebogens zur Bewertung des Gesamtkonzepts

Beantwortung eines Fragebogens: Mittels eines Fragebogens wurden der Projektausschuss befragt, inwiefern die ZuMaTra-Codeanalyse, Modellierungsmethodik und Vorgehensweise die Testfallerstellung und den Testprozess allgemein verbessern kann. Der Fragebogen kann

dem Anhang (Anhang CD: Fragebogen ZuMaTra) entnommen werden. Vom Projektausschuss wurden 9 Fragebögen mit Teilnehmern aus 7 Unternehmen beantwortet.

Die allgemeine Auswertung des ZuMaTra-Ansatzes im Vergleich zu aktuellen Vorgehensweisen in den Unternehmen zeigt eine Verbesserung des Testprozesses auf allen Ebenen. Insbesondere die Erhöhung des Automatisierungsgrades bei der Durchführung und Erstellung von Testfällen zeigt die Erfüllung der Anforderungen an den Ansatz.

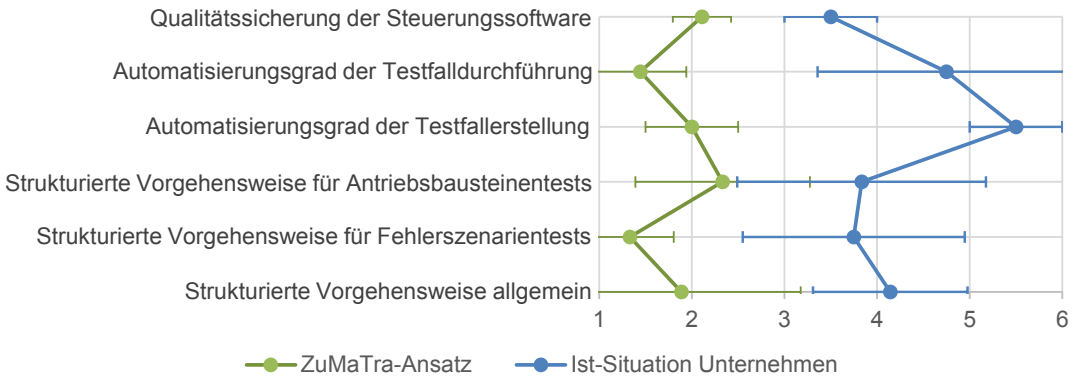


Abbildung 36: Bewertung des ZuMaTra-Ansatzes im Vergleich zur aktuellen Vorgehensweisen - 1: voll erfüllt, 6: überhaupt nicht erfüllt

Das Meinungsbild über die Notation des Weg-Zeit-Diagramms spiegelt die Erfahrungen während der Machbarkeitsstudie und des Workshops wieder. Man sieht, dass die Fokusgruppe die Abbildung diskreter Prozesse deutlich besser, als die Modellierung kontinuierlicher Prozesse einschätzt. Die gute Bewertung der Verständlichkeit und Erlernbarkeit der Notation kann vermutlich auf die Verwendung und Anpassung bereits vorhandener Notationen in den Unternehmen zurückgeführt werden.

Bei der Codeanalyse sind die Ergebnisse ebenso tendenziell positiv. Die Anforderung der Nachvollziehbarkeit von fehlerhaften Kontrollflusspfaden wird durch die einfache Visualisierung von fehlerhaften Pfaden gut erfüllt. Die Einschätzung, dass die Qualitätssicherung und Überprüfung von Ausführungsrichtlinien noch nicht ausgereizt sind, zeigt jedoch einen weiteren Handlungsbedarf in diesem Bereich.

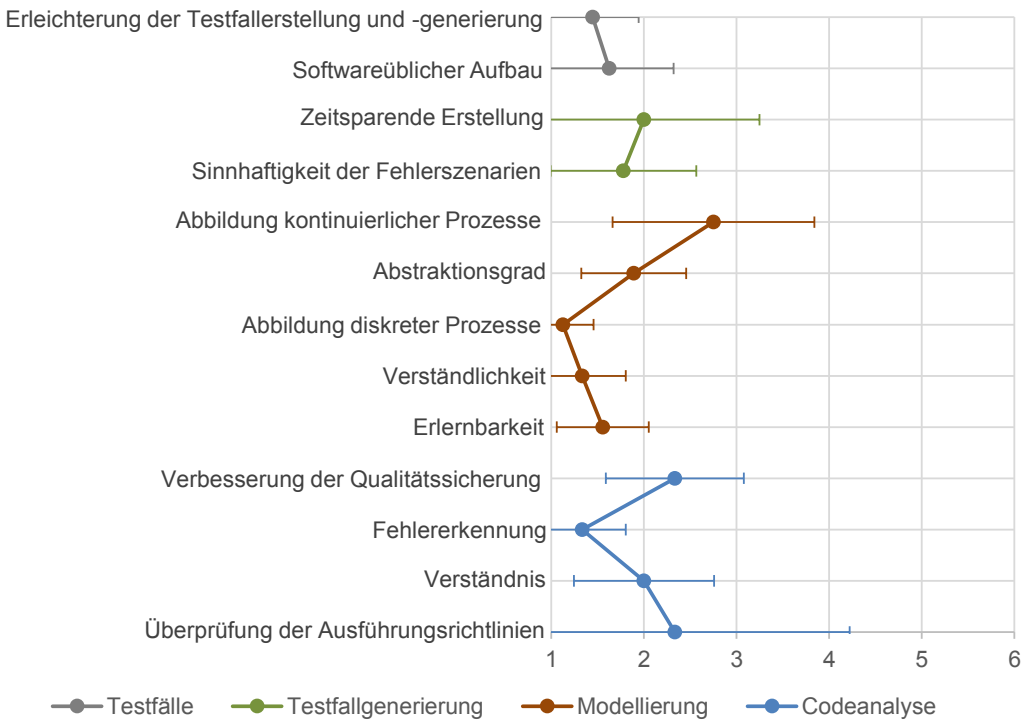


Abbildung 37: Evaluierung des ZuMaTra-Ansatzes - 1: voll erfüllt, 6: überhaupt nicht erfüllt

Neben den vorgegebenen Fragen konnten die Teilnehmer der Umfrage auch weiteres Feedback zu den Ansätzen und Teilansätzen geben.

2.6.4.1 Feedback und weitere Potentiale zur Codeanalyse

Zur Codeanalyse wurde hervorgehoben, dass ggf. nicht nur bestimmte Ausführungsrichtlinien, sondern auch Programmiermuster eingehalten werden müssen. Dies stellt eine weitere Herausforderung an Folgeprojekte. Weiterhin wurde hervorgehoben, dass es tendenziell immer Pfade gibt, in denen bestimmte Regeln nicht gelten. Hier müssen Mechanismen zur Definition von Ausnahmen und insbesondere eine Dokumentation dieser geschaffen werden. Die Visualisierung des Codes als Kontrollfluss wurde außerdem für Bausteine mit begrenzter Komplexität als hilfreich eingestuft. Der Umgang mit komplexeren Bausteinen birgt ein weiteres Untersuchungspotential.

2.6.4.2 Feedback zum Test von Fehlerbehandlungsroutinen

Für die bessere Einsetzbarkeit der Methodik wurde die Notwendigkeit der Datendurchgängigkeit hervorgehoben. Es müssen Schnittstellen und Austauschformate zu anderen Engineering-Werkzeugen geschaffen werden, um Arbeiten wie die manuelle Übertragung von Daten zu vermeiden. Ein Beispiel für ein solches Modellierungswerkzeug ist das EPLAN Engineering Center.

Ein weiterer Hinweis zur Anwendbarkeit war, dass die Lösung auch für Siemens umgesetzt werden müsste, um ein weitere Anwender zu erreichen, da hier noch einige Unterschiede zur CODESYS-Welt bestehen.

Für den Test kontinuierlicher Systeme sind insbesondere Regler, wie auch in dem Beispiel gezeigt, wichtig. Hier müsste die Modellierung von Reglerverhalten genauer untersucht und in

den Ansatz integriert werden. Ein weiteres Potential wird insbesondere bei der Simulation von kontinuierlichen Systemen gesehen.

2.6.5 Zusammenfassung der Evaluierung

Durch die entwickelten Methoden und die mehrstufige Evaluierung durch Machbarkeitsnachweis, Workshop und Umfrage konnte gezeigt werden, dass die zu Beginn gesetzten Anforderungen erfüllt und die Randbedingungen eingehalten werden konnten.

Die Anforderungen und Randbedingungen sind im Folgenden noch einmal zusammengefasst.

A1: der Automatisierungsgrad der Testerstellung und –ausführung muss möglichst hoch sein.

A2: Durchführbarkeit sowohl gegen eine Simulation als auch gegen die reale Maschine

A3: freie Wahl des Abstraktionsgrades bei der Modellierung

A4: die Neugenerierung von Testfällen und Anpassung von Modellen als Grundlage für die Generierung muss möglich sein

A5: Nachweis des Abdeckungsgrades der Anforderungen sowie der Dokumentation für den Kunden

A6: Test von relevanten Fehlerszenarien

A7: Testfallgenerierung aus Weg-Zeit-Diagrammen

A8: optionale Verwendung der FMEA für die Priorisierung der Testfälle

A9: Die Codeanalyseregeln sollen entsprechend der aus den Unternehmen analysierten Regeln und den Verriegelungsbedingungen möglich sein

RB 1: Alle Anwendungsbeispiele enthalten Routinen für eine Grundstellungsfahrt/ Referenzpunktfahrt/ Reset.

RB 2: manuelle Eingriffe durch den Operator während der Testausführung müssen spezifizierbar sein.

RB 3: Für den Test relevanter Fehlerszenarien müssen Fehler entsprechend der untersuchten Fehlererkennungsmechanismen injiziert werden

RB 3.1: Zur Überprüfung von Verriegelungsbedingungen muss der Kontrollfluss untersucht werden

RB 3.2: Zur Überprüfung von Fehlerbehandlungen von bestimmten Szenarien (Prüfung Parametrierung, Prüfung (Sensoren), Überprüfung komplexes Signal und Sammelfehler) soll eine anforderungs- bzw. modellbasierte Testfallgenerierung durchgeführt werden

RB 4: Tests zur Prüfung von Fehlerbehandlungen können aufgeteilt werden in:

- *Fehlerindividueller Teil: Test der richtigen Fehlererkennung und Meldung*
- *Test der richtigen Fehlerbehandlung für verschiedene Fehlerbehandlungsklassen*

RB 5: eine Abbruchroutine muss vorgesehen werden um undefinierte Zustände zu vermeiden

Anforderung 1 wurde erfüllt, indem insbesondere der Testerstellungsprozess so weit wie möglich automatisiert wurde. Die Modellierung des Gutverhaltens im Weg-Zeit-Diagramm ermöglicht die automatische Generierung zahlreicher Testfälle für die Fehlerinjektion. Die Generie-

rung von Test-Funktionsbausteinen aus dem Weg-Zeit-Diagramm und der Testtabelle ermöglicht eine nahtlose Überführung ausführbarer Testfälle in die Programmierumgebungen. Die Auswahl einer automatischen oder semi-automatischen Ausführung ermöglicht den Testern volle Flexibilität. Der Automatisierungsgrad der Testerstellung und –ausführung konnte auch laut Ergebnis der Umfrage (Abbildung 36: Bewertung des ZuMaTra-Ansatzes im Vergleich zur aktuellen Vorgehensweisen - 1: voll erfüllt, 6: überhaupt nicht erfülltAbbildung 36) erheblich erhöht werden und in dem Workshop mit den Unternehmen konnte die Durchführbarkeit der Tests gegen die reale Maschine gezeigt werden (A2).

Auf die Einhaltung der Anforderungen A3, A4, A7 und A8 wurde bereits inhärent während der Entwicklung der Konzepte geachtet. Die Generierung von Testfällen aus Weg-Zeit-Diagrammen, in denen das Gutverhalten modelliert ist, wurde konzipiert und evaluiert. A4 wird soweit abgedeckt, dass das Weg-Zeit-Diagramm stets geändert und neugeneriert werden kann. Verbesserungspotential bietet noch die Beibehaltung der bereits spezifizierten Daten in der Testtabelle um diese Anforderung vollständig zu erfüllen. Auch die Wahl des Abstraktionsgrades wurde von den Teilnehmern der Evaluation als gut bewertet.

Die Erstellung der FMEA wurde als optionaler Teil in den Ansatz integriert (A8). Ein kurzer Leitfaden zeigt, wie Fehler und Fehlerbehandlungen nach Risiko priorisiert werden können. Eine hohe Priorisierung kann in die konzipierte Testtabelle durch die manuelle Priorisierung übertragen werden. Die manuelle Priorisierung erlaubt es dem Tester alternativ auch nach Erfahrungswerten zu priorisieren.

Weiterhin wurde bei der Generierung der Testfällen die nach den Randbedingungen RB 1, RB 2, RB 4 und RB 4 vorgegebene Struktur eingehalten, welche nach der Fragebogenevaluation auch nochmals als sinnvoll bestätigt wurde eingehalten. Die Machbarkeit gegen eine Simulation wurde nicht gezeigt, wäre aber grundsätzlich durch einen Austausch der Maschine gegen eine Simulation denkbar. Der Nachweis des Abdeckungsgrades der Anforderungen sowie der Dokumentation (A5) konnte anhand der Aufgabenstellung und Abdeckung der Testfälle aller Anforderungen in dem Workshop illustriert werden. Die nach RB 3.2 relevanten Fehlerszenarien (A6) wurden alle im Ansatz berücksichtigt und in der Umfrage noch einmal bestätigt (Abbildung 37). Die Regeln welche sich zur Codeanalyse eignen (RB3.1) wurden weiterhin implementiert und überprüft.

2.7 Ergebnistransfer in die Wirtschaft

Ein zentrales Ziel des Forschungsvorhabens war die wissenschaftlich fundierten Resultate anwenderbezogen aufzubereiten. Für einen reibungslosen und stetigen Informationsfluss bedurfte es verschiedener Maßnahmen bereits während der Projektlaufzeit. In regelmäßigen Projekttreffen werden die Projektfortschritte den Mitgliedern des Projektbeirats und weiteren interessierten Unternehmen vorgestellt und die Ergebnisse vor dem Hintergrund der Praxiserfahrung der Industrie reflektiert. Die Veröffentlichungen von Ergebnissen auf einschlägigen Konferenzen (IFAC World Congress) waren, ebenso wie die öffentliche Präsentation der Zwischen- und Endergebnisse, obligatorisch. Darüber hinaus wurden die Projektergebnisse und Konzepte auf einer Messe (SPS/IPC/Drives Kongress) vorgestellt. Auf dem regelmäßig stattfindenden Automation Symposium der Forschungsstelle, an dem ca. 60 Teilnehmer größtenteils aus der industriellen Praxis teilnehmen, dem auch Workshops vorausgehen, wurden zudem die Ergebnisse und Konzepte als Präsentation bzw. Demonstration mit den Laboranlagen des Lehrstuhls vorgestellt. Zusätzlich fand ein Workshop im Rahmen des Projekts mit den Mitgliedern des Projektausschusses statt, der die Ergebnisse des Forschungsvorhabens an Interessensvertreter der Industrie transportiert. Die Anwendung des Software-Funktionsmusters wurde in einem Leitfaden festgehalten und ist unter http://zumatra.ais.mw.tum.de/ veröffentlicht.

Die Forschungsstelle ist darauf bedacht die neuesten wissenschaftlichen Erkenntnisse und deren praktischen Nutzen auch über das Projektende hinaus weiter zu vermitteln. Neben dem starken Industriebezug der Forschungsstelle wurde und wird weiterhin ein Wissenstransfer an die angehenden Ingenieure über die universitären Lehrveranstaltungen Softwareentwicklung

für Ingenieure 1 & 2 erfolgen. Dadurch werden „Berufseinsteiger“ frühzeitig mit den neuesten Erkenntnissen ausgebildet und können somit neue Impulse setzen. Die wissenschaftlich erarbeiteten Methoden fließen nach Projektende in Dissertationen und dokumentieren die Relevanz des Forschungsvorhabens.

Eine Übersicht über die realisierten und weiterhin geplanten Maßnahmen zum Ergebnistransfer ist in Tabelle 4 dargestellt. Die Workshops stellten sich als besonders erfolgreich für den Ergebnistransfer heraus weshalb das Schulungskonzept, welches am 06.02.2015 erstmalig umgesetzt wird, als besonders erfolgsversprechend erachtet wird.

Tabelle 4: Maßnahmen zum Ergebnistransfer

	Maßnahme	Ziel	Rahmen	Datum
Während der Projektlaufzeit (01.01.2012 – 30.09.2014)	Diskussionsforum zusammen mit PA	Fortschrittsberichte, Diskussionen und Koordination weiterer Schritte, Informationsverbreitung	PA-Sitzungen	10.05.2012, 23.10.2012, 16.07.2013, 03.12.2013, 12.03.2014, 22.05.2014, 23.07.2014,
	Vergabe studentischer Arbeiten	Studierende frühzeitig an neue Erkenntnisse heranführen, Umsetzung und Evaluierung von Teilaspekten	Semester- und Abschlussarbeiten	Wintersemester 2013/2014, Wintersemester 2014/2015
	Übernahme der Ergebnisse in universitäre Lehrveranstaltungen	Die Wissensweitergabe als Multiplikator, Basis für neue Mitarbeiter und Arbeiten	Aktualität der Lehrveranstaltungen	Sommersemester 2012
	Vortrag / Demonstration	Verbreitung der Zwischenergebnisse, Evaluation des Konzepts	Automation Symposium 2012	21.02.2012
	Vortrag / Demonstration	Verbreitung der Zwischenergebnisse bei ZVEI-Mitgliedern	ZVEI Herbstsitzung	03.09.2013
	Vortrag / Demonstration/ Diskussion	Einzelworkshops mit Unternehmen	Unternehmen Projektausschuss	01.08.2012, 10.10.2012, 03.09.2013, 01.10.2013, 08.10.2013, 10.10.2013, 24.01.2014, 05.02.2014, 30.09.2014
	Vortrag / Demonstration/ Veröffentlichung	Verbreitung der Zwischenergebnisse, Evaluation des Konzepts	IFAC World	29.08.2014
Nach Abschluss	Abschlussveranstaltung	Anfertigen eines schriftlichen Berichts mit den Forschungsergebnissen für PA und AiF, Präsentation der Ergebnisse gegenüber PA	AiF, PA-Sitzung	11.11.2014

der Projekt-laufzeit	Vortrag/ Demonstration	Verbreitung der Ergebnisse	SPS IPC Drives	25.11.-27.11.2014
	Leitfaden und Editor	Verbreitung der Ergebnisse	Website	12.2014
	Vortrag/ Schulung	Verbreitung der Ergebnisse	Automation Symposium und Schulung	05.02.2015/ 06.02.2015
	Veröffentlichung	Verbreitung der Ergebnisse	Verteiler des ZVEI	02.2015
	Workshop mit 10 Vertretern von KMU des Maschinen-/ Anlagenbaus	Verbreitung der Projekt-ergebnisse, Evaluation des Konzepts	Unternehmen Projektausschuss	13.10.2014
	Dissertation	Dokumentation der wissenschaftlichen Aspekte des Forschungsvorhabens		2015

3 Nutzen für KMUs

Die Zuverlässigkeit einer Maschine bzw. Anlage ist in Zeiten von weltweit vernetzen Produktionssystemen, die nach just-in-time oder build-to-order Konzepten zeitlich voneinander abhängig sind und keine Pufferzeiten für Maschinenausfälle mehr einplanen, ein sehr wichtiges Verkaufskriterium. Daher bietet die nachweisbare Zuverlässigkeit von Maschinen und Anlagen bei gleichzeitig nur sehr geringen Mehrkosten für einen automatisierten Testprozess ein klares Differenzierungsmerkmal und sichert besonders die Wettbewerbsfähigkeit kleiner und mittelständischer Maschinen- und Anlagenbauer.

Steuerungssoftware wird entweder direkt vom Maschinen- und Anlagenbauer entwickelt und getestet oder von einem Dienstleister. Der Maschinen- und Anlagenbau ist in Deutschland traditionell mittelständisch geprägt (wie auch im projektbegleiteten Ausschuss vertreten). Dienstleister in diesem Bereich sind in der Regel kleine Unternehmen. Sowohl Maschinen- und Anlagenbauer als auch Dienstleister sind einem hohen Kosten- und Zeitdruck ausgesetzt bei gleichzeitig hohen Qualitätsanforderungen. Durch die bessere Unterstützung systematischer Software-Tests wird eine kontinuierliche Steigerung der Softwarequalität erreicht. Typische Fehler können projektübergreifend analysiert und systematisch vermieden werden. Langfristig wird ein insgesamt steigendes Qualitätsniveau bei der Softwareentwicklung erreicht. Im Vergleich zum Nutzen und zur Steigerung der Wettbewerbsfähigkeit ist der aufzubringende personelle Aufwand zur Anwendung der angestrebten Forschungsergebnisse als gering zu betrachten. Durch die angestrebte erhebliche, quantifizierbare Qualitätssteigerung der Steuerungssoftware und damit auch der gesamten Maschine bzw. Anlage bei einem gleichzeitig nur geringen Mehraufwand für die Anwendung der Testautomatisierung wird die Wettbewerbsfähigkeit deutscher Maschinen- und Anlagenbauer erheblich gesteigert und langfristig gesichert.

Neben der Verbesserung der Steuerungssoftware können Unternehmen die angestrebten Forschungsergebnisse direkt in Form eines Produkts umsetzen. Für den Bereich der Automatisierungstechnik lassen sich kaum Ansätze für den automatisierten Test von Steuerungssoftware finden. Daher würde ein solches Produkt einen hohes Innovationspotential bieten. Neben dem eigentlichen Produkt kann eine Vielzahl von abgestimmten Dienstleistungen angeboten werden. Die Entwicklung eines marktfähigen Produkts kann ab sofort erfolgen. Idealerweise durch oder in Kooperation mit Herstellern einer Programmierumgebung für Steuerungen und/oder Komponentenherstellern. Durch die Zusammenarbeit der KMUs von Maschinen- und Anlagenherstellern als auch Anbietern von Automatisierungslösungen innerhalb des Projekts, konnten die Anforderungen an ein solches Produkt insbesondere während der Evaluationsworkshops ermittelt werden.

4 Zusammenfassung und Ausblick

In dem Projekt ZuMaTra konnte eine umfassende Vorgehensweise von der Modellierung über die Testfallgenerierung bis hin zur automatisierten Durchführung von Testfällen entwickelt werden. Durch eine Analyse im Industrieumfeld im Bereich der Fertigungstechnik konnten die wesentlichen Anforderungen und Randbedingungen für einen automatisierten Test von Fehlerbehandlungsroutinen ermittelt werden. Aufbauend auf den ermittelten Anforderungen wurde ein Ansatz für die modellbasierte Generierung von Testfällen zur Überprüfung von Ausnahmesituationen entwickelt.

Durch den modellbasierten Ansatz können aus angepassten Weg-Zeit-Diagrammen, in welchen das Gutverhalten modelliert wird, mit Hilfe von in dem Projekt definierten Fehleroperatoren, welche realistische Fehler der Fertigungstechnik abbilden, Testfälle generiert werden. Testtabellen, welche die einzelnen Testfälle auflisten dienen zur Ergänzung von Informationen welche den Testfall ausführbar machen. Ein Testfall testet die Reaktion der Maschine oder Anlage auf ein bestimmtes Fehlerszenario wie beispielsweise den Ausfall eines Endlagensensors. Eine Import in Programmierumgebungen per PLCopen XML sichert die Datendurchgängigkeit und ermöglicht die direkt Ausführung der generierten Testfälle.

Die Evaluation mit zwei Beispielen aus der Industrie von Robert Bosch und Harro Höfliger ergab, dass die Vorgehensweise die Anforderungen aus der Industrie für eine effiziente Testfallgenerierung und automatisierte Testdurchführung für diskrete Prozesse voll erfüllt. Die Nutzung des Weg-Zeit-Diagramms als Grundlage für die Testfallgenerierung erfüllt die wesentlichen Kriterien der schnellen Erlernbarkeit und guten Anwendbarkeit bei niedrigem Aufwand durch eine adäquate Abstraktion des Maschinen-Modells. Diese Faktoren führen maßgeblich dazu, dass alle Voraussetzungen für eine niedrige Einführungsbarriere des Ansatzes in die Industrie gegeben sind. Der Ansatz kann wesentlich zur Erhöhung der Effizienz bei der Qualitätssicherung bzw. zur Erhöhung der Testabdeckung beitragen. Für kontinuierliche Prozesse konnte der grundsätzliche Nachweis der Machbarkeit erbracht werden, ein Nachweis bzw. eine Anpassung des Ansatzes auf die Anforderungen, die solche Prozesse mit sich bringen, steht jedoch noch aus. Insbesondere was den Test von Reglerbausteinen angeht, gibt es noch Forschungsfragen, wie die Eignung von Modellen für einen Testansatz und die Umsetzung einer Testfallgenerierung auf Basis dieser Modelle.

Auch bei der Codeanalyse konnte gezeigt werden, dass die statische Analyse wesentlich zur Qualitätssicherung beitragen kann. Insbesondere wurden jedoch noch weitere Handlungsbedarfe was den Umgang mit der Komplexität beispielsweise in Form geeigneter Visualisierungen, der Untersuchung von Programmiermustern betrifft aufgedeckt, die in einem Folgeprojekt des Lehrstuhls bearbeitet werden sollen.

Eine weitere wichtige Fragestellung ergab sich aus der Untersuchung zur Testfallselektion. Da der Projektausschuss den grundsätzlichen Einsatz der FMEA zur Testfallselektion als zu aufwändig eingestuft hat, wäre zu untersuchen welche Methoden und Kriterien bei einer automatisierten Selektion geeignet und einsetzbar wären.

Das Software-Funktionsmuster der ZuMaTra-Vorgehensweise, umgesetzt durch einen Editor, und die Veröffentlichung unter davon http://zumatra.ais.mw.tum.de/ ermöglicht Unternehmen die Vorgehensweise direkt in Ihrem Unternehmen einzusetzen und zu erproben. Durch den Workshop, bei dem die Vorgehensweise erprobt wurde, konnte außerdem gezeigt werden, welche Anforderungen für die Unterstützung der Vorgehensweise durch Software-Engineering Werkzeuge noch erfüllt werden müssen. Insbesondere sind hier die Datendurchgängigkeit zu anderen Engineering-Werkzeugen und die engere Integration und Synchronisation zwischen Testtabelle und Weg-Zeit-Diagramm hervorzuheben.

5 Anhang

5.1 Literaturverzeichnis

[AAA+90] Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E., Powell, D.: Fault injection for dependability validation: a methodology and some applications, *IEEE Trans. Softw. Eng.*, Vol. 16, Nr. 2, S. 166-182, 1990

[ArCr10] Arlat, J., Crouzet, Y.: Physical Fault Models and Fault Tolerance, *Models in Hardware Testing*, Frontiers in Electronic Testing, Springer Netherlands, Vol. 43, S. 217-255, 2010

[AHM+08] Ayewah, N., Hovemeyer, D., Morgenthaler, J.D., Penix, J., Pugh W.: Using Static Analysis to Find Bugs, *IEEE Softw.*, Vol. 25, Nr. 5, S. 22–29, 2008

[ALR+04] Avizienis, A., Laprie, J.-C., Randell, B., Landwehr C.: Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing*, Vol. 1, Nr. 1, S. 11-33, 2004

[APR+13] Angerer, F., Prähofer, H., Ramler, R., Grillenberger, F.: Points-to analysis of IEC 61131-3 programs: Implementation and application, in *IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, S. 1–8, 2013

[ArBi07] Artho, C., Biere, A.: Combined Static and Dynamic Analysis, *Electron. Notes Theor. Comput. Sci.,* Vol. 131, S. 3-14, 2005

[AVF+01] Aidemark, J., Vinter, J., Folkesson, P., Karlsson, J.: GOOFI: generic object-oriented fault injection tool, *Proc. Int. Conf. on Dependable Syst. and Networks*, IEEE Comput. Soc., S. 83-88, 2001

[BGG+05] Baraza, J.C., Gracia, J., Gil, D., Gil, P.J.: Improvement of fault injection techniques based on VHDL code modification, *10th IEEE Int. High-Level Design Validation and Test Workshop*, IEEE, S. 19-26, 2005

[CuRo05] Cunning, S., Rozenblit, J.: Automating test generation for discrete event oriented embedded systems, Springer, 2005

[CMS98] Carreira, J., Madeira, H., Silva, J.G.: Xception: Software Fault Injection and Monitoring in Processor Functional Units, *Dependable Computing and Fault Tolerant Systems 10*, S. 245-266, 1998

[EKF+09] Ebenhofer, G., Kerbleder, G., Fritsche, J., Strasser, T.: Fokus auf Online-Monitoring und Debugging, *Computer & Automation*, 12 Nov. 2009

[EmNi08] Emanuelsson, P., Nilsson, U.: A Comparative Study of Industrial Static Analysis Tools, *Electron. Notes Theor. Comput. Sci.*, Vol. 217, S. 5–21, 2008.

[FB05] Frey, G., Baniy, M.: Systematisches Re-Engineering bestehender Steuerungsprogramme auf der Basis formaler Beschreibung. SPS/IPC/Drives, Nürnberg, Nov. 2005

[HKV+11] Hametner, R., Kormann, B., Vogel-Heuser, B., Winkler, D., Zoitl, A.: Test case generation approach for industrial automation systems, *The 5th International Conference on Automation, Robotics and Applications*, S. 57-62, Dec. 2011

[HTI97] Hsueh, M.-C., Tsai, T.K., Iyer, R.K.: Fault Injection Techniques and Tool, Computer, Vol. 30, Nr. 4, S. 75-82, Apr. 1997

[HuFr06] Hussain, T., Frey, G.: UML-based Development Process for IEC 61499 with Automatic Test-case Generation, *IEEE Conference on Emerging Technologies and Factory Automation, 2006. ETFA'06*, S. 1277-1284, 2006

[HWÖ+10] Hametner, R., Winkler, D., Östreicher, T., Biffl, S., Zoitl, A.: The Adaptation of Test-Driven Software Processes to Industrial Automation Engineering, *IEEE International Conference on Industrial Informatics (INDIN)*, S. 921-927, 2010

[IEC03] IEC: International Electrical Commission, IEC 61131 Programmable Controllers - Part 3: Programming Languages, 2003.

[Jo78] Johnson, S.C.: Lint, a C Program Checker, Bell Laboratories, 1978

[KCJ11] Kumar, B., Czybik, B., Jasperneite, J.: Model based TTCN-3 testing of industrial automation systems — First results, *IEEE Conference on Emerging Technologies and Factory Automation*, IEEE, S.1-4, 2011

[KHC+99] Kim, Y., Hong, H., Cho, S., Bae, D. Cha S.: Test Cases Generation from UML State Diagrams, *IEE Proceedings - Software*, Vol. 146, No. 4, S. 187-192, Aug. 1999

[KHD08] Krause, J., Herrmann, A., Diedrich, C.: Test case generation from formal system specifications based on UML State Machines, atp international, 2008

[KoVo11] Kormann, B., Vogel-Heuser, B.: Automated Test Case Generation Approach for PLC Control Software Exception Handling using Fault Injection, *IECON 2011 - 37th Annual Conf. of the IEEE Ind. Electronics Soc.*, IEEE, S. 365-372, 2011

[KTV12] Kormann, B., Tikhonov, D., Vogel-Heuser, B.: Automated PLC Software Testing using adapted UML Sequence Diagrams, *14th IFAC Symposium of Information Control Problems in Manufacturing*, Bucharest, Romania, S. 1615-1621, 2012

[LD09] logi.DIAC: Test Driven Automation und Condition Monitoring in der Systemumgebung logi.cals, 2009

[Mo00] Montenegro, S.: Fehlertoleranz und Sicherheit, *Workshop "Software-Entwurf für Kfz-Steuergeräte und komplexe eingebettete Systeme"*, Erfurt, 24. / 25., Okt., 2000

[OM09] On-the-fly-Migration und Sofort-Inbetriebnahme von automatisierten Systemen (OMSIS), http://www.vdivde-it.de, abgerufen 2009

[Ot08] Otto, A.: Der Weg zum sicheren Funktionsbaustein, *SPS-MAGAZIN SPSS*, 2008

[PAC+12] Powell, D., Arlat, J., Chu, H.N., Ingrand, F., Killijian, M.: Testing the Input Timing Robustness of Real-Time Control Software for Autonomous Systems, *9th European Dependable Computing Conf.*, IEEE, S. 73-83, May 2012

[Ru07] Russ, M.: Virtueller Funktions-Prüfstand für softwareintensive mechatronische Produkte, Dissertation Technische Universität München, Sierke Verlag, 2007,

[SCW+11] Sung, A., Choi, B., Wong, W.E., Debroy, V.: Mutant generation for embedded systems using kernel-based software and hardware fault simulation, *Inform. and Software Technology*, Elsevier B.V., Vol. 53, Nr. 10, S. 1153-1164, Oct. 2011

[StEr08] Stetter, R., Erben, M.: Automatisches Testen bei SPS-Steuerungssoftware, atp-Special Steuerungstechnik aktuell, 2008

[SEK+09] Seitz, M., Ehret, V., Kiefer, M., Ziegler, A., Kruschitz, E., Usselmann, E.: Automatisches Testen von Automatisierungssystemen, http://www.automatisierungs-region.de, 2009

[SKV00] Schludermann, H., Kirchmair, T., Vorderwinkler, M.: SOFT-COMMISSIONING: HARDWARE-IN-THE-LOOP-BASED VERIFICATION OF CONTROLLER SOFTWARE, *Proc. of the 2000 Winter Simulation Conf.*, S. 893-899, 2000

[SlVu05] Schlingloff, B.-H., Vulinovic, S.: Zuverlässigkeitsprüfung eingebetteter Steuergeräte mit modellgetriebener Fehlerinjektion. *Simulations- und Testmethoden für Software in Fahr*, Berlin, 2005

[SVE+10] Svenningsson, R., Vinter, J., Eriksson, H., Törngren, M.: MODIFI: a MODel-implemented fault injection tool, *Computer Safety, Reliability, and Security, Lecture Notes in Computer Science,* Vol. 6351, S.210 – 222, 2010

[Ta14]* Tanz, S.: Konzeption und Implementierung eines Ansatzes für die Codeanalyse von IEC 61131-3 Funktionsblockdiagrammen. Bachelorarbeit am Lehrstuhl für Automatisierung und Informationssysteme, Technische Universität München, 2014.

[TeCom06]* Testfallcompiler für den Funktionstest eingebetteter Software. Gefördert aus Haushaltsmitteln des BMWi über die AiF (AiF-FV-Nr. 13660 N/1), 2004-2006, geleitet von AIS

[TTCN3] TTCN3: ETSI ES 201 873-6 V3.2.1, Feb. 2007

[VBR+07] Vinterl, J., Bromander, L., Raistrick, P., Edlerl, H.: FISCADE - A Fault Injection Tool for SCADE Models, *3rd Inst. of Eng. and Technology Conf. on IET*, S. 1-9, 2007

[YRL+03] Yuste, P., Ruiz, J.C., Lemus, L., Gil, P.: Non-intrusive Software-Implemented Fault Injection, *Dependable Computing*, Springer Berlin Heidelberg, S. 23-38, 2003

[ZAV04] Ziade, H., Ayoubi, R., Velazco, R.: A Survey on Fault Injection Techniques, *The Int. Arab J. of Inform. Technology*, Vol. 1, Nr. 2, S. 171-186

* Veröffentlichungen von Mitarbeitern der Forschungsstelle

5.2 Abbildungsverzeichnis

5.3 Tabellenverzeichnis

5.4 Anhang A

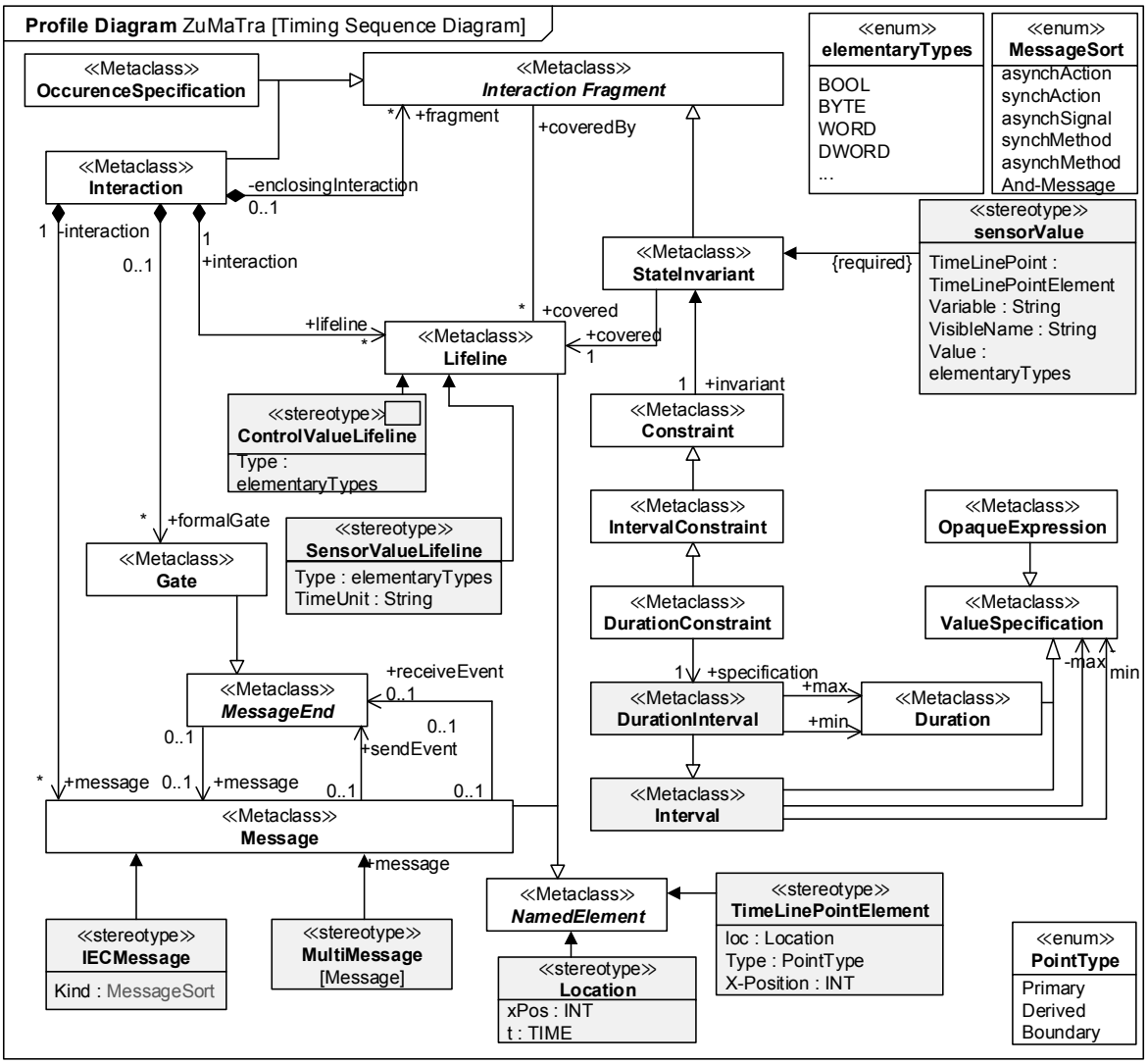


Abbildung 38: Metamodell des Weg-Zeit-Diagramms

5.5 Verzeichnis Anhang CD

Auf der CD befinden sich folgende Folien und Materialien

Material und Leitfäden

- ZuMaTra Leitfaden Gesamt
- ZuMaTra Leitfaden Folien
- Aufgabenstellung Workshop Bosch
- Zusatzmaterial Workshop Bosch

Evaluation

- Fragebogen ZuMaTra

Veröffentlichungen

- Automation Symposium 2012: Testen in der Automatisierungstechnik
- SPS IPC Drives: Modellbasierter Fehlerinjektions-Applikationstest für SPS-Programme basierend auf dem CODESYS Test Manager
- Poster SPS IPC Drives

Danksagung

Diese Veröffentlichung entstand im Rahmen des IGF-Projekts Steigerung der **Zu**verlässigkeit von **M**aschinen und Anlagen durch **a**utomatisiertes **T**esten von Fehlerbehandlungs**r**outinen in der Steuerungssoftw**a**re (ZuMaTra). Das IGF-Vorhaben 16906 N der Forschungsvereinigung Elektrotechnik beim ZVEI e.V. wurde über die AiF im Rahmen des Programms zur Förderung der Industriellen Gemeinschaftsforschung (IGF) vom Bundesministerium für Wirtschaft und Technologie aufgrund eines Beschlusses des Deutschen Bundestages gefördert.