Masterarbeit am Institut für Informatik der Freien Universität Berlin, Arbeitsgruppe Software Engineering, FG Modellbasierte Entwicklung und Qualitätssicherung in Zusammenarbeit mit dem Bereich Satellites

der EADS Astrium GmbH, Standort Ottobrunn

Automatisierte Generierung von Testfällen basierend auf Fehlerauswirkungsanalyse (FMEA) und modellbasiertem Testansatz (MBT) am Beispiel des Satellitennavigationssystems Galileo

Michelle Zeuner
Matrikelnummer: 4490477
Michelle.Zeuner@gmx.de

1. Betreuer:	Dr. Helmut Gogl
2. Betreuer:	Dipl.-Inform. M.Sc. Marc-Florian Wendland
Eingereicht bei:	Prof. Dr. Ina Schieferdecker

Berlin, 4. Dezember 2012

Zusammenfassung

IT-Systeme kommen heutzutage zunehmend in sicherheitskritischen Bereichen zum Einsatz, in welchen von diesen Systemen eine besonders hohe Zuverlässigkeit verlangt wird. Zur Gewährleistung der geforderten Zuverlässigkeit sind in der Folge aufwendige und mit hohen Kosten verbundene Systemtests notwendig. Um den dafür zu betreibenden Aufwand und die damit verbundenen Kosten niedrig zu halten sind deshalb Testansätze gesucht, die es erlauben die nötigen Systemtests möglichst effizient zu erstellen, durchzuführen, auszuwerten und zu warten.

Zu diesem Zweck wird in dieser Arbeit untersucht, ob ausgehend von Ergebnissen aus der Fehlerauswirkungsanalyse (FMEA) und unter Nutzung eines modellbasierten Testansatzes (MBT) eine automatisierte Generierung von Testfällen bzw. Testcode auf effiziente Weise möglich und sinnvoll ist.
Dazu wird zunächst vorbereitend ein Überblick über die gängigsten Testansätze aus der Literatur gegeben und dabei insbesondere auf den modellbasierten Testansatz sowie dazu mögliche Modellnotationen eingegangen. Danach werden verschiedene Arten von Fehleranalysen betrachtet und speziell die hier zu verwendende FMEA genauer erläutert. Auf Basis dieser beiden Komponenten, also des modellbasierten Testansatzes und der Fehlerauswirkungsanalyse, wird die FMEA/MBT-Testmethode schrittweise entwickelt und allgemein eingeführt.

Im praktischen Teil dieser Arbeit wird die entwickelte FMEA/MBT-Methode anhand eines zuvor ausgewählten MBT-Werkzeugs auf Teilsysteme der Galileo Satelliten angewendet. Dazu wird diese neue Testmethode zunächst in das Projekt Galileo integriert. Dabei wird kurz auf die relevanten Strukturen und Komponenten des Satellitensystems eingegangen und dargestellt wie sich die FMEA/MBT-Methode in die vorhandene Galileo Testumgebung einbinden lässt. Nach Darstellung der Realisierung des gesamten Testsystems, erfolgt die exemplarische Anwendung der FMEA/MBT-Methode auf zwei Teilsysteme (Erdsensor und Stabilisierungsschwungräder) der Galileo Satelliten.

Anhand der gewonnenen Erfahrungen und Erkenntnisse aus der Entwicklung und Anwendung der FMEA/MBT-Methode wird abschließend bewertet, ob sich dieser Testansatz in der Praxis bewährt hat und ob dieser die Erwartungen im Hinblick auf eine effizientere Durchführung von Systemtests erfüllen kann.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

4. Dezember 2012

(Michelle Zeuner)

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Herrn Dr. Helmut Gogl der Astrium GmbH für seine ausgezeichnete Betreuung.

Des Weiteren bedanke ich mich für die hilfreichen Unterstützungen von Herrn Dipl.-Inform. M.Sc. Marc-Florian Wendland und Prof. Dr. Ina Schieferdecker. Außerdem danke ich Herrn Dr. Hans-Jürgen Herpel der Astrium GmbH für die Ermöglichung dieser Arbeit.

Ebenfalls möchte ich meiner Familie und meinen Freunden meinen herzlichen Dank ausdrücken, die mir während dieser Arbeit und der Studienzeit zur Seite standen.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Das Projekt Galileo . . . 2
1.3 Besondere Anforderungen in der Raumfahrt . . . 3
1.4 Aufgabenstellung und Ziel . . . 4

2 Ansätze zum Testen von IT-Systemen **7**
2.1 Aufgaben, Ziele und Herausforderungen . . . 7
2.2 Manuelles Testen . . . 9
2.3 Automatisiertes Testen . . . 11
2.3.1 Capture & Replay Testen . . . 12
2.3.2 Skriptbasiertes Testen . . . 13
2.3.3 Schlüsselwortgetriebenes Testen . . . 14
2.3.4 Modellbasiertes Testen . . . 16
2.4 Vergleich der Testmethoden . . . 19
2.5 Methoden zur Beschreibung von Modellen . . . 20
2.5.1 Zustandsbasierte Notation . . . 20
2.5.2 Transitionsbasierte Notation . . . 21
2.5.3 Weitere Notationen . . . 21

3 FMEA/MBT Methode zur automatisierten Testfallgenerierung **23**
3.1 Allgemeines zur FMEA/FMECA . . . 23
3.1.1 Definition . . . 23
3.1.2 Vorgehensweise . . . 24
3.1.3 Vor- und Nachteile . . . 25
3.2 Inhalt und Zweck der FTA . . . 26
3.3 Vorgehensweise zur Testerstellung im Projekt Galileo . . . 27
3.4 Neue Vorgehensweise basierend auf FMEA und MBT . . . 28

4 Werkzeugunterstützung für MBT **31**
4.1 Klassifizierung von MBT-Werkzeugen . . . 31
4.2 Anforderungen an MBT-Werkzeuge bzgl. Testdokumentation 31
4.3 Übersicht einer Auswahl von MBT-Werkzeugen . . . 32
4.3.1 SpecExplorer . . . 32
4.3.2 NModel . . . 33
4.3.3 Conformiq Tool Suite . . . 33
4.3.4 Fokus!MBT . . . 34
4.4 Testsprachen . . . 34
4.4.1 UTP . . . 34
4.4.2 TTCN-3 . . . 35
4.5 Auswahl eines MBT-Werkzeugs . . . 35

5 Die FMEA/MBT-Methode im Kontext von Galileo **37**
5.1 Die HW/SW Architektur der Galileo Satelliten 37
5.2 Das FDIR Konzept im Galileo Projekt 40
5.3 Die Betriebsmodi der Galileo Satelliten 41
5.4 Beschreibung der Galileo spezifischen FMEA 43
5.4.1 Strukturierung der FMEA im Galileo Projekt 43
5.4.2 Inhalt der FMEA im Galileo Projekt 44
5.4.3 Beziehung zwischen FMEA, FDIR und Tests 47
5.5 Die Testumgebung von Galileo 47
5.6 Anbindung von SpecExplorer an die Testumgebung 48

6 Anwendung der FMEA/MBT-Methode im Projekt Galileo **51**
6.1 Beispiel 1.1: Erdsensor (SpecExplorer) 51
6.1.1 FMEA basierte Modellierung mit SpecExplorer 52
6.1.2 Erweiterung des FMEA-Modells mit SpecExplorer . . 69
6.2 Beispiel 1.2: Erdsensor (Fokus!MBT) 73
6.3 Beispiel 2: Stabilisierungsschwungräder 77
6.3.1 FMEA basierte Modellierung mit SpecExplorer 78
6.3.2 Erweiterung des FMEA-Modells mit SpecExplorer . . 88

7 Zusammenfassung **92**
7.1 Ergebnisse 94
7.1.1 Ergebnisse: Modellierung und Testfallgenerierung . . . 94
7.1.2 Das Gesamtergebnis 98
7.2 Kritische Bewertung 99
7.3 Ausblick 107

1 Einleitung

1.1 Motivation

In zunehmendem Maße kommen heutzutage IT-Systeme in sicherheitskritischen Bereichen unseres Lebens zum Einsatz. Als Folge wird von diesen Systemen eine besonders hohe Zuverlässigkeit verlangt. Entsprechende Systeme finden sich in unterschiedlichen Bereichen wieder, wie beispielsweise der Medizintechnik oder der Luft- und Raumfahrtindustrie. In diesen Anwendungsbereichen sind idealerweise fehlerfreie Systeme gefordert, da ein Fehlverhalten dort zu enormen Folgeschäden führen kann.

Zur Analyse der Zuverlässigkeit eines Systems existieren verschiedene Methoden. Unter diesen stellt die FMEA (engl. Failure Modes and Effects Analysis) eine der am häufigsten angewandten Methoden dar (vgl. Abschnitt 3.1). Anhand solcher Fehleranalysen können Fehlerfälle, die zur Betriebszeit eines Systems auftreten können, auf ihre Ursachen und Folgen untersucht und entsprechende Maßnahmen zu deren Verhinderung bzw. zur Schadensbegrenzung getroffen werden. Die in den Fehleranalysen betrachteten Fehlerfälle sollten dabei die Menge aller im System möglichen Fehlerzustände umfassen.

Um nachzuweisen, dass ein System die bezüglich Zuverlässigkeit und Funktionalität gestellten Anforderungen erfüllt, wird dieses einem aufwändigen Testprozess unterzogen. Die Erstellung der dafür notwendigen Einzeltests sollte dabei möglichst effizient erfolgen, um alle nötigen Testfälle innerhalb eines vorgegebenen Zeit- und Kostenrahmens abdecken zu können. Zu diesem Ziel kann der Einsatz von automatisierten Testverfahren, wie z.B. dem modellbasierten Testen beitragen. Mit deren Hilfe kann anhand der Testfallgenerierung der zu investierende Aufwand für die Testfallerstellung reduziert werden.

Zum Nachweis, dass beim Testen eines Systems alle sicherheitskritischen Fehlerfälle des Systems einbezogen wurden, ist von Interesse, ob das Heranziehen der Ergebnisse aus der FMEA als Anforderungsspezifikation für die Testerstellung sinnvoll ist. Die Vollständigkeit der Fehleranalyse vorausgesetzt könnte aus dieser das geforderte Systemverhalten für jeden möglichen Fehlerzustand entnommen werden.

Mittels einer Machbarkeitsstudie soll deshalb in dieser Arbeit überprüft werden, ob es möglich und sinnvoll ist, auf Basis der FMEA und unter Nutzung eines modellbasierten Testansatzes die Generierung von Testfällen bzw. Testcode zu automatisieren. Die in dieser Arbeit entwickelte Vorgehensweise soll dazu exemplarisch am Projekt Galileo demonstriert werden.

1.2 Das Projekt Galileo

Das Satelliten-Navigationssystem Galileo wurde von der Europäischen Union initiiert und stellt das europäische Pendant zum US-amerikanischen GPS (Global Positioning System) bzw. dem russischen Glonass (Global Navigation Satellite System) dar. Die Leitung des Galileo Projektes obliegt der europäischen Raumfahrtagentur ESA (European Space Agency). Mit diesem Projekt wird das erste europäische Navigationssystem realisiert (vgl. [ITW12]).

Das Galileo Satellitensystem wurde für die Bereitstellung folgender Navigationsanwendungen ausgelegt:

1. Basisdienst
2. Kommerzieller Dienst
3. Sicherheitskritischer Dienst
4. Öffentlich regulierter Dienst
5. Such- und Rettungsdienst

Am 21. Oktober 2011 und am 12. Oktober 2012 wurden jeweils zwei Galileo Satelliten für die IOV (engl.: In-Orbit Validation) Phase ins Weltall geschossen. Die Anforderung, dass sich zur Validierung der Navigationsfunktionalitäten mindestens vier Satelliten in der Umlaufbahn befinden müssen, ist somit erfüllt. Nach Abschluss dieser Phase wird in die IOC (engl.: Initial Operational Capability) Phase übergegangen, in der das Galileosystem durch die Inbetriebnahme weiterer Satelliten vervollständigt wird. Am Endes des Jahrzehnts sollen alle benötigten Galileo Satelliten in Betrieb sein, so dass die endgültige FOC (engl.: Full Operational Capability) Phase umgesetzt werden kann.

Insgesamt sollen 30 Satelliten in einer Höhe von 23.222 km auf drei kreisförmigen Satellitenbahnen um die Erde kreisen. Auf dieser Höhe befindet sich das Satellitensystem im MEO (engl.: Medium Earth Orbit). Pro Flugbahn steht jeweils ein redundanter Satellit zur Verfügung, um im Falle eines Satellitenausfalls das Navigationssystem aufrecht erhalten zu können (vgl. [ESA12]).

Für den Fall, dass die Galileo-Satelliten ungünstige Empfangsbedingungen aufweisen, kann auf Grund der Interoperabilität zu GPS und Glonass zusätzlich auch auf die Signale dieser Systeme zurückgegriffen werden, so dass immer ausreichend Navigationssignale empfangen werden können (vgl. [ITW12]).

Die Entwicklung der Galileo IOV-Satelliten erfolgte durch die Astrium GmbH, so dass für die Erstellung dieser Arbeit sowohl die Systemsoftware der Galileo Satelliten als auch die Ergebnisse der vollständigen FMEA zur Verfügung standen. Dadurch bietet das Projekt Galileo alle nötigen Voraussetzungen, um die in der Motivation erwähnte Machbarkeitsstudie durchführen zu können.

1.3 Besondere Anforderungen in der Raumfahrt

In der Raumfahrt herrschen besondere Anforderungen, die bei bemannten oder unbemannten Flügen im Weltraum zu bewältigen sind. Denn die unterschiedlichen Raumfahrtsysteme (wie Raketen, Satelliten, Weltraumfrachter, etc.) operieren dort in einer deutlich andersartigen Umgebung als auf der Erde.

Befindet sich ein Raumfahrtsystem im Weltall, so ist es starken Strahlungen ausgesetzt. Die solaren, UV oder kosmischen Strahlungen verursachen an den an Bord befindlichen Geräten einen erheblich schnelleren Alterungsprozess als auf der Erde. Diesen gilt es zum einen durch geeignete Maßnahmen so weit wie möglich zu verlangsamen. Zum anderen muss den erhöhten Ausfallraten von HW-Komponenten beim Systemdesign Rechnung getragen werden.

Zusätzlich zur Strahlung erfordert die Raumfahrt, dass extreme Temperaturschwankungen bewältigt werden, da sich in Abhängigkeit von der Position des Raumfahrzeuges (sonnenbeschienen, im Erdschatten, schwankende Entfernung zur Sonne, etc.) Materialtemperaturen von über 100°C bis unter -100°C ergeben können (vgl. [Lei12]).

Neben diesen direkten Umgebungsfaktoren des Weltalls gibt es weitere Anforderungen, welche die Wartung der Raumfahrtsysteme betreffen. Betrachtet man beispielsweise den Bereich der Satelliten, so besteht zwischen diesen und der Erde eine enorme Distanz. Diese ist auf Grund des dafür notwendigen Aufwands nur noch schwer bzw. unter extremen Kosten zu überbrücken.

Daher entfällt im Allgemeinen die Möglichkeit notwendige Fehlerbehebungen direkt vor Ort am Satelliten vorzunehmen. Die einzigen Kontaktmöglichkeiten zum Satelliten sind funktechnischer Natur und durch Befehle von der Bodenstation an den Satelliten (Telekommando) bzw. Nachrichten vom Satelliten zur Bodenstation (Telemetrie) gegeben.

Falls bei Satelliten Fehlerzustände eintreten, so können diese zwar häufig manuell über entsprechende Telekommandos der Bodenstation kompensiert werden. Allerdings besteht nicht immer Funkkontakt zum Satelliten. Daher

ist es nötig an Bord autonome Systeme einzusetzen, welche auf einen Fehlerfall selbständig reagieren und einen sicheren Betrieb zumindest bis zum nächsten Kontakt mit der Bodenstation sicherstellen.

Dieser Grad an Autonomie soll den Satelliten bzw. generell die Raumfahrtsysteme insbesondere vor Ausfällen schützen, die in der Folge zu einem Totalverlust des Systems führen würden. Dazu sollte das System eigenständig die möglichen Fehlerzustände erkennen und beheben können. Zur Durchführung der Fehlerbehebung werden in der Raumfahrt in erster Linie redundante Ausstattungen genutzt, so dass bei fehlerbehafteten Einheiten einfach auf die Ersatzeinheiten umgeschaltet werden kann.

Wird das Raumfahrtsystem wie beschrieben vor sicherheitskritischen Ausfällen geschützt, so kann die Ausfallwahrscheinlichkeit von für eine Raumfahrtmission erfolgsentscheidenden Komponenten auf ein Minimum reduziert werden. Durch das auf diese Weise realisierte Maß an Ausfallsicherheit bzw. Zuverlässigkeit kann das System den korrekten operationalen Betrieb über die geplante und geforderte Lebensdauer erreichen.

1.4 Aufgabenstellung und Ziel

Wie unter Abschnitt 1.1 bereits erwähnt, soll mittels dieser Abschlussarbeit überprüft werden, ob auf Basis der FMEA und unter Nutzung eines modellbasierten Testansatzes (MBT) eine automatisierte Generierung von Testfällen bzw. Testcode möglich und sinnvoll ist. Neben der Überprüfung der prinzipiellen Machbarkeit dieses Ansatzes besteht das Ziel insbesondere darin eine praxistaugliche Vorgehensweise zu entwickeln, welche exemplarisch am Projekt Galileo durchzuführen und zu bewerten ist.

Zur Erreichung obiger Ziele gliedert sich die Umsetzung der Aufgabenstellung in folgende Hauptschritte:

- Sichtung gängiger Testansätze
- Beschreibung der Fehleranalyse mittels FMEA/FMECA/FTA
- Entwicklung der FMEA/MBT-Methode
- Auswahl eines MBT-Werkzeugs
- Integration der FMEA/MBT-Methode in Galileo
- Beispielhafte Anwendung der FMEA/MBT-Methode in Galileo

Im Anschluss an diese Einleitung werden in Kapitel 2 zunächst die gängigsten Testansätze aus der Literatur dargestellt. Nach einer kurzen Erläuterung des manuellen Testens wird dabei vor allem auf verschiedene Ansätze

zum automatisierten Testen eingegangen. Anhand einer kurzen Gegenüberstellung von Vor- und Nachteilen dieser Ansätze (vgl. Abschnitt 2.4) wird abgesichert, dass sich der modellbasierte Testansatz MBT tatsächlich für die gegebene Aufgabenstellung eignet. Danach werden speziell für diesen Ansatz die unterschiedlichen für eine Modellierung verfügbaren Beschreibungsmethoden präsentiert.

Als nächster Schritt wird in Kapitel 3 auf die Durchführung von Fehleranalysen mittels der Methoden FMEA/FMECA und FTA eingegangen. Da die in dieser Studie zu entwickelnde Testmethode auf die Daten/Ergebnisse der Fehleranalyse FMEA zurückgreifen soll, wird speziell die FMEA eingehender besprochen.

Da die zu entwickelnde FMEA/MBT-Testmethode im späteren Verlauf der Arbeit auf das Projekt Galileo angewendet werden soll, wird zur Abgrenzung die bisher im Projekt verwendete Testvorgehensweise geschildert. Daraus wird abgeleitet, an welchen Stellen die neu zu entwickelnde Methode zu Optimierungen beitragen könnte. Im Anschluss daran wird die FMEA/MBT-Methode in Abschnitt 3.4 zunächst allgemein, d.h. ohne weiteren direkten Bezug zum Projekt Galileo erläutert. Anhand der entwickelten Vorgehensweise wird schrittweise gezeigt, wie ausgehend von der FMEA entsprechende Testmodelle erstellt, daraus Testfälle automatisiert generiert und schließlich ebenso automatisiert ausgeführt werden können.

Um den in der FMEA/MBT-Methode enthaltenen modellbasierten Testansatz verwenden zu können, ist dessen Unterstützung durch ein geeignetes MBT-Werkzeug nötig. In Kapitel 4 werden dazu einige Werkzeuge sowohl aus dem kommerziellen als auch dem nicht-kommerziellen Umfeld vorgestellt. In diesem Zusammenhang wird auch kurz auf den Bereich der Testsprachen eingegangen. Danach erfolgt in Abschnitt 4.5 die Wahl des in dieser Arbeit eingesetzten Werkzeugs, bei welcher insbesondere auf die Verfügbarkeit des Tools sowie die Abdeckung aller für die FMEA/MBT-Methode nötigen Funktionalitäten Wert gelegt wurde.

Nach Festlegung der FMEA/MBT-Methode und dem dafür zu verwendenden MBT-Werkzeug, wird in Kapitel 5 auf die Integration dieser Vorgehensweise in das Projekt Galileo eingegangen. Dazu werden zunächst grundlegende Informationen zur Architektur eines Galileo Satelliten, dessen Fehlerbehandlungskonzept und Betriebsmodi gegeben. Ebenso werden die Strukturierung und der Inhalt der Galileo spezifischen FMEA dargestellt und deren Bezug zum erwähnten Fehlerbehandlungskonzept erläutert. Anschließend wird die Funktionsweise des umzusetzenden Gesamttestsystems, bestehend aus dem zuvor gewählten Werkzeug und der ursprünglichen Galileo Testumgebung, näher beschrieben.

In Kapitel 6 erfolgt schließlich die praktische Anwendung der erarbeiteten FMEA/MBT-Methode auf zwei unterschiedliche Teilsysteme aus dem Fluglageregelungssystem der Galileo Satelliten. Dabei wird die Modellierung jedes Teilsystems zunächst jeweils strikt (der Aufgabenstellung folgend) auf die aus der FMEA ableitbaren Anforderungen begrenzt. Die dabei generierten Testfälle entsprechen somit direkt und nur den Einzelfehlerbetrachtungen aus der FMEA[1]. Die Kombinationen von Fehlern werden also in der Regel nicht in den entstehenden Testfällen berücksichtigt.

Aus diesem Grund wurde für jedes Teilsystem eine zweite, erweiterte Modellierung vorgenommen, in welcher obige Restriktion gelockert wurde und die Verwendung von zusätzlichem Systemwissen (welches ebenso wie die FMEA in frühen Projektphasen verfügbar wäre) zulässig war. Wie in Kapitel 6 gezeigt wird, erlaubt dies eine natürlichere Modellierung des Systemverhaltens und es entstehen bei moderatem Aufwand viele zusätzliche Testfälle. Weiterhin wird in diesem Kapitel an einem der Teilsysteme beleuchtet, wie sich die FMEA/MBT-Methode bei Wechsel des verwendeten MBT-Werkzeugs verhält.

In Kapitel 7 werden die Ergebnisse der Arbeit zusammengefasst und eine kritische Bewertung der Vor- und Nachteile des vorgestellten Testansatzes insbesondere im Hinblick auf Praxistauglichkeit des Verfahrens vorgenommen. Abschließend gibt dieses Kapitel einen Ausblick zu denkbaren Weiterentwicklungsmöglichkeiten der FMEA/MBT-Methode.

[1] Die Galileo FMEA betrachtet grundsätzlich nur Einzelfehler und deren direkte Auswirkungen. Die Kombination von Einzelfehlern wird nur betrachtet, wenn sich die Fehlerfolge ausschließlich aus der Kombination ergibt.

2 Ansätze zum Testen von IT-Systemen

2.1 Aufgaben, Ziele und Herausforderungen

In diesem Abschnitt werden die Aufgaben, Ziele und Herausforderungen beim Testen von IT-Systemen betrachtet. Die Betrachtung bezieht sich an dieser Stelle auf allgemeine Aspekte, ist also unabhängig von einer bestimmten Testmethode. Im Anschluss an diesen Abschnitt werden die in der Literatur gängigen Vorgehensweisen zum Testen von IT-Systemen näher erläutert.

Aufgaben und Ziele

Marc Utting und Bruno Legeard haben sich intensiv mit der Thematik des Softwaretestens beschäftigt (vgl. [UL07]). Die Autoren beschreiben das Testen als eine Aktivität, bei der die Produktqualität evaluiert wird. Zusätzlich können die Informationen der identifizierten Fehlerzustände und Probleme herangezogen werden, um die Produktqualität optimieren zu können (s. [UL07], S. 3).

In Bezug auf das Testen von IT-Systemen bedeutet dies, dass das globale Ziel das Messen der Qualität des zu testenden Systems (SUT, engl.: System under Test) ist. Die Anforderungsspezifikationen des SUT dienen dabei als Prüfreferenz, um die korrekte Umsetzung der Anforderungen durch das System bzw. Abweichungen zwischen spezifiziertem und tatsächlichem Verhalten des Systems identifizieren und später korrigieren zu können.

Nach der ISO/IEC 25000er-Reihe (früher ISO/IEC 9126) kann die Qualität einer Software in die beiden wesentlichen Bereiche *Gebrauchsqualität* und *äußere und innere Qualität* gegliedert und anhand deren Unterpunkten bewertet werden (vgl. [Sch11]):

1. Gebrauchsqualität
   (a) Effektivität
   (b) Produktivität
   (c) Sicherheit
   (d) Zufriedenheit
2. Äußere und innere Qualität
   (a) Funktionalität
   (b) Zuverlässigkeit
   (c) Benutzbarkeit
   (d) Effizienz
   (e) Änderbarkeit
   (f) Portierbarkeit

Die Gebrauchsqualität bezieht sich dabei auf die Verwendbarkeit des Systems aus Sicht des Kunden. Dazu wird in Betracht gezogen, ob das System den Anforderungsspezifikationen innerhalb der vorgegebenen Grenzen entspricht. Diese können u.a. die Genauigkeit und Vollständigkeit betreffen, oder aber auch den Aufwand des Nutzers in Form von z.B. Zeit oder Geld. Ebenfalls sollten die Risikogrenzen eingehalten werden, sowie die subjektive Zufriedenheit des Benutzers innerhalb des Nutzungskontextes.

Die äußere und innere Qualität bezieht sich wiederum direkt auf das System. Dabei wird evaluiert, ob die Funktionalitäten des Systems korrekt sind, inwiefern das System bei Problemen zuverlässig ist, wie gut es sich benutzen lässt (z.B. Verständlichkeit, Bedienbarkeit) oder wie sich das Zeit- und Verbrauchsverhalten äußert. Ebenfalls soll es durch eine ausgeprägte Änderbarkeit analysierbar, modifizierbar, stabil und testbar sein. Außerdem trägt die Portierbarkeit (u.a. Anpassbarkeit, Konformität, Installierbarkeit) zur Systemqualität bei.

Die in der Aufzählung genannten Merkmale bestimmen die Gesamtqualität des Systems. Da beim Testen eines IT-Systems eben diese Qualitätsmerkmale untersucht werden, müssen die entsprechenden Systemanforderungen diesbezüglich herangezogen werden. Anhand dieser Informationen kann dann abgeglichen werden, ob sich das SUT spezifikationskonform verhält.

Vor Beginn des eigentlichen Testprozesses zur Überprüfung des SUT gegen dessen Anforderungsspezifikationen muss entschieden werden, welche Testvorgehensweise angewendet werden soll. Die dazu möglichen Vorgehensweisen verfügen über jeweils verschiedene Vor- und Nachteile.

Da in dieser Studie das Hauptaugenmerk auf automatisierte Testverfahren gerichtet wird, werden deren Herangehensweisen in Abschnitt 2.3 näher erläutert.

Herausforderungen

Im Bereich des Testens von IT-Systemen bestehen Herausforderungen, die vom Testmanagement bzw. dem ausführenden Testingenieur bewältigt werden müssen. Anteile der unten genannten Herausforderungen wurden aus der veröffentlichten Übersicht des Fraunhofer-Instituts über Herausforderungen beim modellbasierten Testen (vgl. [Koc12]) entnommen.

Eine Herausforderung, welche wohl in den seltensten Fällen tatsächlich bewältigt werden kann, stellt die Realisierung aller theoretisch denkbaren Tests für das SUT dar. Da dies auf Grund der Größe der Testmenge oder wegen Beschränkungen (z.B. durch Abgabefristen) in der Regel nicht machbar ist,

hat das angewendete Testen oftmals einen Stichprobencharakter. Dabei wird die Auswahl der Stichproben so gestaltet, dass eine möglichst umfassende Abdeckung des Systemverhaltens ermöglicht wird. Der dazu investierende Aufwand in Zeit und Kosten sollte dabei so gering wie möglich gehalten werden.

Die verwendete Testmethode kann dabei entsprechend ihrer Vorgehensweise den Aufwand an diversen Stellen minimieren, wie z.B. bei der benötigen Einarbeitungszeit, bei der tatsächlichen Umsetzung oder in der Wartung. Hierbei kann sich vor allem die Verwendung von Testautomatisierungen als sehr nützlich erweisen.

Das von einem oder mehreren Testern entwickelte Testsystem sollte in der Lage sein das SUT bezüglich jeglicher gegebenen Anforderungsspezifikationen überprüfen zu können. Daher gilt es zur Optimierung der Entwicklung sowohl passende Testwerkzeuge zu finden als auch geeignete Testtechniken und -prozesse auszuwählen.

In den folgenden Abschnitten 2.2 und 2.3 wird dazu eine Auswahl an Testverfahren vorgestellt, welche obige Herausforderungen durch ihre jeweiligen Vor- und Nachteile unterschiedlich gut bewältigen.

2.2 Manuelles Testen

Beim manuellen Testverfahren handelt es sich um eine Vorgehensweise, bei welcher der Testentwurf und die Testdurchführung von einem Tester (z.B. Testingenieur) vorgenommen werden. Die Vorgehensweise gliedert sich dabei in folgende Teilschritte:

1. Entwurf der Testfälle und Erstellung eines Testplans
2. Manuelle Testausführung
3. Dokumentation der Testergebnisse

Grundlage für den ersten Schritt bildet die Anforderungsspezifikation des zu testenden Systems. Diese gibt Aufschluss darüber, welche Eigenschaften bzw. welches Verhalten das entwickelte System aufweisen muss. Passend zu diesen Vorgaben werden entsprechende Testfälle manuell abgeleitet und in einem Testplan sortiert zusammengefasst.

Nach Abschluss der Planung, kann der manuelle Test durchgeführt werden. Der Tester überprüft dabei händisch durch schrittweises Abarbeiten der Testfallspezifikation, ob das gezeigte Systemverhalten mit den spezifizierten Anforderungen übereinstimmt. Das heißt, ähnlich zur Benutzung

durch einen gewöhnlichen Anwender, wird direkt am System und unter Verwendung derselben Benutzerschnittstellen getestet. Im Gegensatz zu einem gewöhnlichen Anwender stehen dem Tester jedoch häufig zusätzlich Möglichkeiten zur Fehlereinspeisung bzw. normalerweise nicht zugängliche Messpunkte zur Verfügung.

Die aus der Testdurchführung resultierenden Ergebnisse geben Auskunft über die Konformität der Systemimplementierung bzgl. der Anforderungen und werden zum Abschluss von jedem Testfall in einem Testprotokoll festgehalten (vgl. [UL07]).

Vorteile

Wie oben beschrieben sind als Voraussetzung für die Durchführung manueller Tests im Wesentlichen das zu testende System und ein Tester nötig. Daher liegen die Hürden bzgl. Vorbereitungsaufwand und Einstiegskosten (abgesehen von der Testfallerstellung, -planung) bei diesem Testverfahren relativ niedrig.

Weiterhin sind einfache manuelle Tests häufig in Testszenarien einsetzbar, welche nicht oder nur mit sehr großem Aufwand zu automatisieren wären. Im hardwarenahen Bereich finden sich dazu Beispiele wie das Testen des Systemverhaltens bei ausgefallenen Bus- oder Netzwerkverbindungen. Solche Fehlerfälle lassen sich von einem Tester relativ einfach durch das Ziehen der entsprechenden Stecker herstellen. Der Aufwand für eine Automatisierung wäre dagegen unverhältnismäßig hoch.

Ein weiterer Vorteil liegt in der Flexibilität des manuellen Testens. Es bietet dem Tester die Freiheit neben den geplanten Tests spontan direkt weitere „ad-hoc“ Tests durchzuführen und gegebenenfalls auch direkt den ursprünglichen Testplan zu erweitern. Ebenso erlaubt die manuelle Durchführung die umgehende Anpassung der Testläufe an Veränderungen am zu testenden System.

Nachteile

Die Ausführung manueller Tests kann viel Zeit in Anspruch nehmen. Die gesamte Testdauer ist dabei von der Anzahl und Größe der Testläufe abhängig. Demnach steigt der Testaufwand proportional zum Umfang des Systems. Dies führt dazu, dass das manuelle Testen bei umfangreicheren Systemen schnell an seine Grenzen stößt.

Das Wiederholen von Testsitzungen kann für den Tester sehr ermüdend sein, wodurch sich unbewusst leicht Fehler während der Testausführung einschlei-

chen können. Auch das Reproduzieren des jeweils exakt gleichen Testvorgangs (z.B. durch Einhaltung der gleichen Testgeschwindigkeit, aufwendige Vorbereitung der Testumgebung, etc.) kann schwierig umzusetzen sein (vgl. [UL07, Sof12b]). Diese Punkte erschweren die Durchführung von Regressionstests, bei denen für jede neue Systemversion die exakt gleiche Testreihe durchzuführen ist, um negative Auswirkungen der neu eingebrachten Implementierungsänderungen identifizieren zu können. Ferner kann sich die manuelle Ausführung als nicht umsetzbar erweisen, wenn z.B. extreme Lasttests oder nicht umsetzbare Präzisionen (z.B. im Millisekunden-Bereich) gefordert sind.

2.3 Automatisiertes Testen

Beim automatisierten Testen von Soft- oder Hardware werden Teilaufgaben des Testentwurfs und der -ausführung von einem Werkzeug übernommen. Dazu werden folgende Ziele angestrebt (vgl. [Hom03]):

- **Effektive Testprozessgestaltung**: Um Automatisierungen im Testprozess anwenden zu können, werden Werkzeuge gezielt eingesetzt. Um diese sinnvoll integrieren zu können, muss der gesamte Testprozess in einzelne Vorgänge gegliedert werden. Die erfolgte Strukturierung fördert dabei einen schnelleren und effektiveren Testvorgang, sowie eine effizientere Wiederholung der Testprozesse.
- **Abnahme arbeitsintensiver Aufgaben**: Sich wiederholende, manuelle Aufgaben können sich als sehr arbeitsintensiv bzw. lästig herausstellen. Diese bieten das Potential den zu investierenden Zeit- und Kostenfaktor, sowie durch Routine entstehende Fehler, anhand von Automatisierungen zu minimieren. Ebenso können dadurch Ressourcen (z.B. Zeit- oder Arbeitsressourcen) eingespart werden, welche anderweitig genutzt werden können.
- **Höhere Fehlererkennungsrate**: Durch das Investieren der eingesparten Ressourcen in die Testfallplanung können mittels Automatisierungen Tests größeren Umfangs erstellt werden.

Bei der Umsetzung der Testautomatisierung können folgende Aufgabenbereiche durch entsprechende Werkzeuge übernommen werden (vgl. [Tes12]): Die Testfall-, Testdaten- und Testskripterstellung, sowie die Testdurchführung, -auswertung, -dokumentation und -administration.

Erst nachdem festgelegt worden ist, welche Aufgaben zu automatisieren sind, sollte ein entsprechendes Werkzeug gewählt werden. Dazu muss die Art der Testautomatisierung bestimmt werden. Die in der Literatur gängigen Arten von Testautomatisierungen werden in den folgenden Abschnitten näher erörtert.

2.3.1 Capture & Replay Testen

Die Capture & Replay Testmethode, welche im Deutschen auch als Testroboter bekannt ist, zeichnet die manuell durchgeführten Testläufe durch ein entsprechendes Werkzeug auf und speichert diese in Form von Testskripten ab. Die Anwendung des Verfahrens wird dabei wie folgt in den gesamten Testverlauf integriert:

1. Entwurf der Testfälle und Erstellung eines Testplans
2. Manuelle Testausführung, welche durch ein entsprechendes Werkzeug aufgezeichnet wird
3. Automatisierte Wiederholungen mit Ergebnisprotokollierung

Zu Beginn des teilautomatisierten Testverfahrens wird analog zum manuellen Testen vorgegangen. Die Anforderungsspezifikationen des zu testenden Systems dienen als Grundlage bei der Erstellung der Testplanung. Sie geben das verlangte Systemverhalten bzw. die Systemeigenschaften wieder, die in den Testfällen zu berücksichtigen sind. Die sortierten Testfälle werden im Testplan festgehalten.

Anschließend werden diese Testfälle einmal händisch durchgeführt. Während dieser manuellen Testsitzung zeichnet ein unterstützendes Werkzeug alle Eingaben exakt auf und speichert diese in Form eines Skripts ab. Da es sich hierbei um Vorgänge handelt, die von einem Werkzeug reproduziert werden sollen, müssen die Eingaben von diesem auch zu bewältigen sein. Im Normalfall handelt es sich dabei um Tastatureingaben oder Mausklicks.

Nachdem die Skripte erstellt worden sind, kann die Testausführung beliebig oft auf dem zu testenden System ausgeführt werden. Die während der Ausführung erhaltenen Ergebnisse werden jeweils in einem Testprotokoll erfasst (vgl. [UL07]).

Vorteile

Die Methode ermöglicht auf Basis vorab aufgezeichneter Testläufe eine automatisierte Testausführung, wodurch im Vergleich zum manuellen Testen eine deutliche Zeitersparnis erreicht wird. Langwierige und womöglich monotone Tests werden hierbei von einem entsprechenden Werkzeug übernommen.

Des Weiteren unterstützt die Aufzeichnung eines als Referenz betrachteten Testablaufs die Durchführung von Regressionstests, mit deren Hilfe die Auswirkungen von Modifikationen im System identifiziert werden können (vgl. [Sch08]).

Die Capture & Replay Methode wird vorrangig für den Test von (graphischen) Benutzerschnittstellen eingesetzt, wofür meist die Aufzeichnung und Wiedergabe von Mausklicks, Mausbewegungen und Tastatureingaben genügt. Die Methode ist dadurch unabhängig von der verwendeten Programmiersprache, welche für die Implementierung des SUT verwendet wurde (vgl. [Nef03]).

Nachteile

Beim Capture & Replay Testen handelt es sich um ein statisches und somit unflexibles Verfahren. Werden Modifizierungen am SUT vorgenommen, so können die aufgezeichneten Testläufe unbrauchbar werden (z.B. bei veränderten Positionen von Bedienelementen auf einer graphischen Benutzeroberfläche). In diesem Fall müssen die Testsitzungen stets erneut manuell aufgezeichnet werden. Bei häufigen Änderungen an der Benutzerschnittstelle kann die durch die Automatisierung gewonnene Zeitersparnis so an Nutzen verlieren (vgl. [Hay04, Nef03]).

2.3.2 Skriptbasiertes Testen

Beim skriptbasierten Testverfahren werden durch einen Tester Skripte implementiert, welche alle benötigten Instruktionen zur vollständigen Testausführung beinhalten. Diese Skripte können dann automatisiert auf das zu testende System angewendet werden. Das Verfahren gliedert sich in folgende Teilschritte:

1. Entwurf der Testfälle und Erstellung eines Testplans
2. Implementierung des Testskripts
3. Automatisierte Ausführung des Skripts inkl. Ergebnisprotokollierung

Wie bei den zuvor genannten Verfahren werden auch hier zu Beginn aus den Anforderungsspezifikationen die Testfälle abgeleitet, sowie ein Testplan erstellt.

Anschließend folgt die sich daraus ableitende Implementierung des Testskripts. Dazu müssen ausreichend Systemkenntnisse vorhanden sein, um mit allen benötigten Schnittstellen von Testumgebung und SUT korrekt umgehen zu können. In den Skripten müssen letztendlich alle Aufrufe enthalten sein, um einen vollständigen Testdurchlauf ausführen zu können, ohne vorher oder gar währenddessen weitere Eingaben vornehmen zu müssen.

Nachdem die manuelle Implementierung eines Skripts abgeschlossen ist, kann es mit Hilfe eines Ausführungswerkzeugs auf das SUT angewendet werden.

Die Ausführung kann hierbei beliebig oft wiederholt werden. Die jeweilig resultierenden Ergebnisse werden in einem Protokoll festgehalten (vgl. [UL07]).

Vorteile

Im Vergleich zum vorherigen Testverfahren werden auf Grund der Implementierung der Testskripte keine manuellen Testsitzungen mehr benötigt. Somit müssen bei Modifizierungen im System nur die betroffenen Codestellen in den Skripten angepasst werden.

Das skriptbasierte Testverfahren eignet sich ebenfalls für die Durchführung von Regressionstests. Die manuell erstellten, statischen Skripte dienen hierbei jeweils als Referenz. Diese sind in der Lage Abweichungen vom bisherigen Systemverhalten während eines Testlaufs selbständig zu erkennen und zu protokollieren.

Nachteile

Als nachteilig erweist sich der hohe Aufwand, welcher für die Implementierung der Skripte nötig ist. Jeder einzelne Testschritt muss manuell im Skript implementiert sein, um dem Ausführungswerkzeug die alleinige Testausführung zu ermöglichen. Dazu werden vom Tester Programmierfähigkeiten verlangt.

Zudem sind für die Skriptimplementierung meist gründliche Kenntnisse über das zu testende System erforderlich, um die gewünschten Testszenarien realisieren zu können.

Zur Wartung müssen für jede vorgenommene Systemmodifikation entsprechende Anpassungen in den Testskripten manuell durchgeführt werden. Analog zur eigentlichen Skripterstellung kann dies sehr aufwändig sein, da kleine Änderungen am SUT eventuell größere Anpassungen bzw. Anpassungen an vielen Stellen eines oder mehrerer Testskripte nötig machen. Abhilfe können hierbei Testimplementierungen schaffen, die weitestmöglich die Testfunktionalitäten abstrahieren, wobei die Umsetzung solcher jedoch von den jeweiligen Programmierfertigkeiten des Entwicklers und der zur Verfügung stehenden Zeit abhängen (vgl. [UL07]).

2.3.3 Schlüsselwortgetriebenes Testen

Das schlüsselwortgetriebene Testen stellt eine erweiterte Form des skriptbasierten Testens dar. Hierbei werden jegliche für den Test des SUT benötigte Testfunktionalitäten separiert. Diese Testfunktionalitäten werden in

einer sogenannten Testbibliothek zur Verfügung gestellt, aus welcher diese anhand von eindeutigen Schlüsselwörtern aufgerufen und als einzelne Testschritte ausgeführt werden. Die Vorgehensweise unterteilt sich in folgende Schritte:

1. Entwurf der Testfälle und Erstellung eines Testplans
2. Implementierung einer Bibliothek mit Zugriff durch Schlüsselwörter über einen entsprechenden Adapter
3. Erstellung der Testfälle anhand der Schlüsselwörter
4. Automatisierte Ausführung der Skripte über den Adapter inkl. Erstellung der Dokumentation über die Testergebnisse

Zu Beginn werden die abzuleitenden Testfälle und der Testplan erstellt. Dieser Vorgang geschieht auch bei diesem Verfahren identisch zu den vorherigen Methoden.

Sobald der Testplan entwickelt worden ist, leitet der Testingenieur anhand der geforderten Tests die benötigten Testfunktionalitäten ab. Entsprechend diesen werden separate Testsequenzen implementiert, die schließlich zusammengefasst in einer Bibliothek über zugewiesene Schlüsselwörter aufrufbar sind. Diese Schlüsselwörter werden meistens in Excel, Word, HTML oder CSV gespeichert und verwaltet. Anhand dieser Listen können mittels Schlüsselwörtern die entsprechenden Testfunktionalitäten aufgerufen werden.

Nachdem die Testbibliothek implementiert worden ist, folgt die Testskripterstellung anhand der Auflistung der zum Testlauf notwendigen Schlüsselwörter, wozu vom Tester keine tiefgehenden Programmierkenntnisse verlangt werden.

Sobald die Testbibliothek, der Adapter und der Test mit der enthaltenen Schlüsselwortliste erstellt worden sind, kann die Ausführung des Tests auf dem zu testenden System erfolgen. Dazu muss die Bibliothek über den Adapter erreichbar sein, so dass das über ein Werkzeug ausgeführte Schlüsselwortskript die den Schlüsselwörtern entsprechenden Sequenzen ausführt. Bei jeder Ausführung werden die vom Testlauf resultierenden Ergebnisse in einem Protokoll festgehalten (vgl. [UL07]).

Vorteile

Analog zu den vorherigen Testansätzen, können auch beim schlüsselwortgetriebenen Testen die Testläufe automatisiert und reproduzierbar mittels entsprechender Werkzeuge ausgeführt werden.

Weiterhin erlaubt die Abstraktheit des Testverfahrens eine vereinfachte Testfallerstellung, zu der nur geringe Programmierfähigkeiten und keine tiefgreifenden Systemkenntnisse (z.B. über die korrekte Benutzung der Kommunikationsschnittstellen des SUT, den Zugriff auf interne SUT Variablen oder die Realisierung von Fehlerinjektionen) zur Verfügung stehen müssen. Dies ist möglich, da solche Details in der Testbibliothek verborgen und in Form von einfach zu handhabenden Testfunktionalitäten zur Verfügung gestellt werden. Zur Erstellung der Testfälle ist es also ausreichend die diesen wiederverwenbaren Testfunktionalitäten zugewiesenen Schlüsselwörter sinnvoll zu parametrisieren und aneinanderzureihen, um so das gewünschte Testszenario zu realisieren.

Ferner ermöglicht die Abstraktheit eine effiziente Wartbarkeit. Die Anpassung der Tests an das modifizierte System ist auf der hohen Abstraktionsebene der jeweiligen Testskripte (aus Schlüsselwörtern) in der Regel einfach umzusetzen. Zudem ermöglichen Anpassungen der zugrundeliegenden Testbibliothek die indirekte, gleichzeitige Aktualisierung aller Testskripte, welche die entsprechenden Funktionalitäten der Testbibliothek verwenden. Dieser Vorgang folgt dem sogenannten Kapselungsprinzip. Daher benötigt dieses Vorgehen weniger Aufwand als beim skriptbasierten Testen, bei welchem jegliche von der SUT-Änderung betroffenen Stellen in den Testskripten selbst angepasst werden müssen.

Vergleichbar zu den vorherigen Testmethoden besteht auch hier die Möglichkeit zum Regressionstesten (vgl. [Sof12a]).

Nachteile

Um von den oben genannten Vorteilen profitieren zu können, muss zunächst die Bibliothek mit den Testsequenzen implementiert werden. Dazu wird ein Testingenieur benötigt, der über ausreichende Systemkenntnisse verfügt, um die Schnittstellen sowohl des SUT als auch der restlichen Testumgebung (z.B. HW-Simulator) korrekt benutzen zu können. Über die Schlüsselwörter sollten alle gewünschten Testsequenzen ausführbar sein. Das heißt, dass die Bandbreite der aus der Bibliothek zu erreichenden Systemfunktionalität entsprechend groß sein muss. Die Umsetzung einer guten Testbibliothek kann einen erheblichen Aufwand erfordern, so dass der Einstieg in diesen Testansatz selten schnell umzusetzen ist (vgl. [UL07]).

2.3.4 Modellbasiertes Testen

Beim modellbasierten Testen (MBT) werden Testfälle anhand von Modellen generiert, welche das geforderte Verhalten des SUT bzw. Teilaspekte davon abbilden. Die Vorgehensweise des MBT gliedert sich dabei in folgende

Schritte:

1. Überführung der Anforderungen in ein Testmodell
2. Generierung der Testfälle, dann der Testskripte
3. Anbindung der Testskripte über einen Adapter an das Testausführungstool
4. Automatisierte Ausführung und Auswertung der Testfälle inkl. Erstellung der Dokumentation über die Testergebnisse

Zu Beginn des MBT werden wie bei den oben genannten Testmethoden die Anforderungsspezifikationen des zu testenden Systems herangezogen. Im Gegensatz zu den vorherigen Vorgehensweisen wird zunächst der Testplan in Form eines Testmodells erstellt. Dieses repräsentiert als Verhaltensmodell das spezifizierte SUT-Verhalten.

Für die Spezifikation des Modells stehen unterschiedliche Modellierungsnotationen zur Auswahl, auf welche in Abschnitt 2.5 näher eingegangen wird.

Anhand des erstellten Testmodells werden zunächst mittels eines Testfallgenerators die möglichen Testfälle aus dem Modell generiert. Diese sollen dabei den gewählten Abdeckungskriterien zum Modell nachkommen. Anschließend werden die entsprechenden Testskripte unter Verwendung eines Testskriptgenerators erstellt.

Zur Ausführung der Tests auf dem SUT wird eine Anbindung der Testskripte über einen Adapter an das Testausführungstool benötigt. Falls das Typensystem und die Schnittstellen des Testmodells mit dem des SUT vereinbar sind, können dort die Testbefehle integriert werden, so dass anhand der Skriptgenerierung der vollständige Testcode resultiert. Andernfalls muss der verbindende Adapter zum Testausführungstool über die benötigten SUT-spezifischen Befehle verfügen, so dass die in den Testfällen enthaltenen Aktionen über den Adapter weitergeleitet werden können.

Nach Implementierung der Anbindung zum zu testenden System kann die automatisierte Ausführung der Testskripte durchgeführt werden. Dazu werden die Skripte sequentiell durchlaufen. Dabei wird an entsprechenden Stellen der Testläufe jeweils das geforderte Systemverhalten am SUT überprüft.

Während der Ausführung der Tests werden die Ergebnisse mitprotokolliert. Falls ein Testlauf auf Grund nicht übereinstimmenden Verhaltens des SUT fehlschlägt, kann im Modell die entsprechende Position im Testablauf festgestellt werden. Ein nicht erfolgreich durchlaufener Test kann bedeuten, dass

sich das SUT nicht spezifikationskonform verhält, dass ein Fehler bei der Testerstellung unterlaufen ist oder dass die Systemanforderungen nicht korrekt sind (vgl. [UL07]).

Vorteile

Neben der automatisierten Ausführung der Testläufe (analog zu den vorherigen Methoden) können hierbei reproduzierbare, systematische, automatisierte Testfallableitungen aus einem abstrakten Verhaltensmodell, sowie die dazugehörigen Testskripte generiert werden. Anhand dessen können im Vergleich zu den anderen Verfahren wesentlich mehr Testläufe bei gleichzeitig geringerem Aufwand erzielt werden.

Ein weiterer Vorteil liegt in der leicht durchzuführenden Wartung. Werden Veränderungen am zu testenden System bzw. der Anforderungsspezifikation vorgenommen, so können die notwendigen Anpassungen leicht umgesetzt werden. Dazu müssen die betroffenen Stellen des Testmodells lediglich modifiziert werden, was auf Grund der abstrakten Darstellung des Testmodells einfach umzusetzen ist.

Des Weiteren ist von Vorteil, dass aus einem gleichbleibenden Modell (unter Verwendung eines deterministischen Algorithmus) identische Testfälle generiert werden. Auf Grund dessen können die nach jedem Testdurchlauf resultierenden Ergebnisprotokolle effektiv zum Regressionstesten verwendet werden.

Zudem könnte der modellbasierte Ansatz genutzt werden um eine in Teilen automatisch durchgeführte Erstellung der Testdokumentation (Testergebnisse, Testablauf und Anforderungsüberdeckung) zu ermöglichen. Mittels der Auswertung der Testlogs durch das MBT-Werkzeug könnte dokumentiert werden, welche Testfälle aus dem Testmodell erfolgreich ausgeführt wurden und welche gescheitert sind. Zusätzlich könnte anhand dieser Testlogs der Testablauf im Modell visualisiert werden. Darüber hinaus könnte auch die Dokumentation der Anforderungsüberdeckung unterstützt werden, indem während der Modellierung Beziehungen zwischen dem Testmodell und den entsprechenden davon abzudeckenden Systemanforderungen hergestellt werden (z.B. durch Verwendung von Referenzen im Modell). Dadurch ließe sich die erfolgreiche Ausführung der Tests direkt mit der erfolgreichen Validierung der entsprechenden Systemanforderungen verknüpfen und automatisch dokumentieren.

Nachteile

Als eindeutiges Problem des modellbasierten Testansatzes stellt sich die Zustandsraumexplosion bei komplexeren Systemen dar. Dabei entstehen unübersichtliche, schwer zu handhabende Modelle, welche zu einer immensen Anzahl von Testfällen führen, deren Abarbeitung durch die begrenzte, verfügbare Rechenleistung ggf. schwer zu bewältigen ist.

Als ebenso nachteilig erweist sich der Aufwand, der in die Entwicklung des Testmodells und die Implementierung des Adapters investiert werden muss. Dieser entspricht etwa dem Aufwand des schlüsselwortgetriebenen Testens. Zur Erstellung müssen die nicht trivialen Fähigkeiten zur Modellierung, sowie Programmierfertigkeiten vom Tester beherrscht werden. Je höher die Qualität des Modells ist, desto besser lassen sich Wartungen und Erweiterungen im späteren Verlauf vornehmen (vgl. [Kla05, UL07]).

2.4 Vergleich der Testmethoden

Die oben genannten Testmethoden bieten jeweils unterschiedliche Vor- und Nachteile. Die folgende Tabelle gibt eine Übersicht über ausgewählte Eigenschaften:

	Manuell	**Capture- & Replay**	**Skript-basiert**	**Schlüssel-wortgetr.**	**Modell-basiert**
Einstieg	schnell	schnell	langsam	langsam	langsam
Automation (Testentwurf)	nein	nein	nein	nein	ja
Automation (Testdurchf.)	nein	ja	ja	ja	ja
Abstraktion	nein	nein	ja/nein	ja	ja
Wartbarkeit	nein	nein	ja/nein	ja	ja
Erstellung weiterer Tests	durch-schnittl.	durch-schnittl.	langsam	schnell	schnell
Erzielung vieler Tests	langsam	langsam	langsam	durch-schnittl.	schnell

Tabelle 1: Vergleich unterschiedlicher Testansätze

Abhängig davon welche Testziele verfolgt werden, kann jede Testmethode von Vorteil sein. Die in Tabelle 1 dargestellten Merkmale der einzelnen Verfahren sollten bei der Auswahl einer oder mehrerer Methoden berücksichtigt werden.

In der Aufgabenstellung dieser Arbeit (vgl. Abschnitt 1.4) wurde die Verwendung des modellbasierten Testansatzes angeregt. Durch die Betrachtung und den Vergleich der verschiedenen Testansätze in diesem und den vorangehenden Abschnitten sollte vorab nochmals überprüft werden, ob sich der MBT-Ansatz im Hinblick auf weitere Testvorgehensweisen zur Durchführung dieser Studie eignet.

Ziel war es ein möglichst effizientes Testverfahren zu identifizieren, mit dessen Hilfe sich möglichst viele der in der Tabelle genannten Merkmale optimal umsetzen lassen. Wie sich aus der Tabelle entnehmen lässt, bietet der modellbasierte Testansatz diesbezüglich tatsächlich die größte Übereinstimmung. Aus diesem Grund wird das MBT in dieser Arbeit für die Umsetzung der Aufgabenstellung aus Abschnitt 1.4 herangezogen.

Im nächsten Abschnitt werden für das modellbasierte Testen unterschiedlichen Formalismen präsentiert, die zur Modellierung des spezifizierten Systemverhaltens verwendet werden können.

2.5 Methoden zur Beschreibung von Modellen

Die Qualität der Modellierung ist von der Wahl einer passenden Methode zur Modellbeschreibung abhängig. Diese bestimmt wie detailliert, aussagekräftig und anschaulich das Modell die gewünschten Informationen wiedergeben kann.

Für die Modellierung von funktionalem Systemverhalten besteht eine große Auswahl an Notationen. Das Buch *Practical Model-Based Testing: A Tools Approach* [UL07] gibt über die möglichen Formalismen eine Übersicht.

1. Zustandsbasierte Notation
2. Transitionsbasierte Notation
3. Funkionelle Notation
4. Historienbasierte Notation
5. Operationelle Notation
6. Statistische Notation
7. Datenfluss Notation

2.5.1 Zustandsbasierte Notation

Ein Modell, das der zustandsbasierten Notation folgt, bildet die internen oder nach außen sichtbaren Zustände des betrachteten Systems in abstrakter Form ab. Die im Modell enthaltenen Zustände entsprechen jeweils einer

konkreten Momentaufnahme des zu modellierenden Systems. Zur repräsentativen Abbildung eines Zustands sind diesem aussagekräftige parametrisierte Variablen zugeordnet. Anhand gegebener Vor- und Nachbedingungen zu den Zuständen können die Variablen über Operationen modifiziert werden, so dass neue Zustände erlangt werden können.

Zur Anwendung der zustandsbasierten Notation können u.a. Sprachen wie Spec#, B, OCL (UML Object Constraint Language), VDM oder Z verwendet werden (für nähere Informationen siehe [UL07], S.62).

2.5.2 Transitionsbasierte Notation

Im Gegensatz zur zustandsbasierten Notation werden bei der transitionbasierten Notation die Übergänge (Transitionen) zwischen zwei Zuständen fokussiert. Modelle dieser Art werden meist graphisch mit Knoten und Kanten dargestellt. Die Knoten repräsentieren dabei die relevanten Zustände, die Kanten wiederum die Aktionen oder Operationen des Systems. Ferner können textuelle oder tabellarische Notationen angewendet werden, um die Transitionen zu spezifizieren.

Darüber hinaus können in der praktischen Anwendung die transitionsbasierten Notationen erweitert werden, so dass weiterführende Informationen (z.B. Zustandsvariablen) in die Modelle integriert werden können.

Zur Anwendung der Notation können u.a. endliche Automaten (FSMs), Zustandsübergangsmaschinen (z.B. UML-Zustandsdiagramme oder Simulink Stateflow Diagramme), markierte Transitionssysteme (LTSs) oder Eingabe/Ausgabe Automaten (I/O automata) verwendet werden.

2.5.3 Weitere Notationen

Bei der Anwendung der modellbasierten Testmethode wird die genutzte Modellnotation in der Regel in Abhängigkeit zum MBT-Werkzeug gewählt. Dabei finden die zustands- und transitionsbasierte Notationen die häufigste Anwendung, um das Verhalten des zu testenden Systems abbilden zu können. Auf Grund der Abhängigkeit entscheidet sich erst im späteren Verlauf dieser Arbeit mit welcher Notation gearbeitet wird. Die weiteren Notationen aus der Auflistung, welche seltener zum MBT verwendet werden, werden an dieser Stelle daher nur kurz erläutert.

Funktionelle Notation

Diese Notation unterstützt die Beschreibung mathematischer Funktionen. Diese können bspw. aus der Prädikatenlogik oder Funktionen höherer Ord-

nung, welche Funktionen als Eingabe oder Ausgabe verarbeiten, stammen. Da algebraische Spezifikationen meist sehr abstrakt und nicht einfach zu notieren sind, wird diese Form der Notation im MBT-Bereich selten genutzt.

Historienbasierte Notation

Die historienbasierte Notation ermöglicht die Beschreibung eines erlaubten Verhaltens über eine fortlaufende Zeit.

Des Weiteren sind darunter auch die Nachrichten-Reihenfolge-Diagramme (MSCs, engl. Message Sequence Charts) einzuordnen, welche die Interaktionen zwischen verschiedenen Komponenten besonders gut veranschaulichen. Das Verhalten innerhalb der einzelnen Komponenten wird dabei vernachlässigt.

Zur Umsetzung der MSCs können UML-Sequenzdiagramme verwendet werden.

Operationelle Notation

Anhand dieser Notation werden Prozesse beschrieben, die zeitgleich ausgeführt werden können. Insbesondere für verteilte Systeme und Kommunikationsprotokolle kann diese Form der Notation nützlich sein.

Mögliche Darstellungsformen sind hierbei Petri-Netze oder Arten der Prozessalgebra, wie CSP (eng.: Communicating Sequential Processes) oder CSS (engl.: Calculus of Communicating Systems).

Statistische Notation

Mit dieser Notation werden Wahrscheinlichkeitsaussagen eines Systems beschrieben. Besonders gut lassen sich die Ereignis- und Eingabewerteverteilungen des zu testenden Systems spezifizieren. Weniger gut lässt sich damit die Voraussage eines erwarteten Ausgabewerts notieren. Auf Grund dessen wird für die automatisierte Voraussage des Ausgabewerts eine Kombination dieser Notation mit einer weiteren Notation verwendet, die das Verhalten des SUT beschreibt.

Datenfluss Notation

Bei dieser Notationsform wird nicht der Kontrollfluss, sondern der Fluss der Daten innerhalb des zu testenden Systems fokussiert. Mögliche Darstellungsformen sind u.a. Lustre oder Block Diagramme, die z.B. in Matlab Simulink für kontinuierliche Systeme verwendet werden.

3 FMEA/MBT Methode zur automatisierten Testfallgenerierung

Nachdem im vorherigen Kapitel darauf eingegangen wurde, welche Ansätze zum Testen von IT-Systemen bestehen, werden in diesem Kapitel Methoden zur Durchführung von Fehleranalysen vorgestellt. Die Ergebnisse solcher Fehleranalysen dienen in dieser Arbeit als Anforderungsspezifikationen, welche direkt in die Modellierung der sicherheitskritischen Fehlerzustände eines Systems einfließen. Die dabei gewonnenen Modelle wiederum bilden die Grundlage um entsprechende (zu den Fehlerzuständen passende) Testfälle automatisiert generieren zu können.

Die beiden gebräuchlichsten Arten der Fehleranalyse in Bezug auf sicherheitskritische Systeme sind die FMEA und die FTA, welche im Folgenden näher erörtert werden.

3.1 Allgemeines zur FMEA/FMECA

Die Fehlerauswirkungsanalyse FMEA (engl.: Failure Mode and Effects Analysis) wurde zum ersten Mal im Jahr 1949 im Bereich des Militärs angewendet, um dortige Fehlermöglichkeiten analysieren zu können. Dadurch sollte der Verlust von Personen und Ausstattungen reduziert werden.

In den 1960ern wurde das FMEA-Verfahren von der NASA für den sicherheitskritischen Bereich der Raumfahrt adaptiert. Schließlich wurde es in den 1970ern ebenfalls von großen Automobilkonzernen (Ford, Chrysler und General Motors) übernommen.

Heutzutage ist der Einsatz der FMEA sehr weit verbreitet, insbesondere in sicherheitskritischen Bereichen von Industrien wie der Luft- und Raumfahrt, der Automobilindustrie und der Pharmazie (vgl. [Con12]).

3.1.1 Definition

Die FMEA ist ein systematisches Analyseverfahren zur Ermittlung von Fehlerzuständen im System mit Untersuchung der Ursachen und Folgen. Sie ist die am häufigsten verwendete Methode zur Qualitätsanalyse (vgl. [Sch10]).

Bei der FMECA (engl.: Failure Mode and Effects Criticality Analysis) handelt es sich um eine erweiterte Form der FMEA, welche in der Analyse die Kritikalität der Ausfälle mit einschließt (vgl. [IST07]).

Die Analyseverfahren FMEA bzw. FMECA werden zu folgenden Zwecken angewendet (vgl. [IST07, Sch10]):

- Zum Erhalt wichtiger Ergebnisse zur Risikoanalyse/zum Risikomanagement.
- Zur Ermittlung des Systemverhaltens im Fall von möglichen Fehlerzuständen.
- Zum Priorisieren der möglichen Ausfallrisiken.
- Zur Minimierung des Ausfallrisikos anhand der analysierten Kritikalität der Systems.
- Zur Eliminierung möglicher Fehlerursachen durch geeignete Maßnahmen.
- Zur Erkennung und Eliminierung potentielle Fehler in einem möglichst frühen Entwicklungsstadium.
- Wenn obligatorische oder gesetzliche Anforderungen zu berücksichtigen sind.

3.1.2 Vorgehensweise

Adam Moik hat im Jahr 1999 den Artikel *Strukturierte Erstellung von formalen Sicherheitsmodellen für Automatisierungssysteme mit Sicherheitsverantwortung* (vgl. [Moi99]) mit Inhalten zur FMEA und deren Durchführung veröffentlicht. Er unterteilt das Verfahren der FMEA in die folgenden sequentiellen Schritte:

1. Strukturanalyse
2. Funktionsanalyse
3. Fehleranalyse
4. Bewertung der Kritikalität (FMECA)
5. Optimierung

Im ersten Schritt des Analyseverfahrens wird die Strukturanalyse durchgeführt. Dazu wird das betrachtete System in eine hierarchische Struktur zerlegt, an der sich das FMEA-Modell orientieren soll.

Im Anschluss an die Strukturanalyse folgt die Anwendung der Funktionsanalyse. Hierbei wird zu den Systemteilen, die in der Strukturanalyse festgelegt

worden sind bestimmt, welche Anforderungen und Funktionen erfüllt werden sollen.

Nachdem die ersten beiden Analysen abgeschlossen worden sind, werden zu den zugeordneten Anforderungen und Funktionen die möglichen Fehlfunktionen bestimmt. In dieser sogenannten Fehleranalyse werden daraufhin die herausgefundenen Fehlerursachen und -effekte in die FMEA-Tabellen eingetragen. Dabei ist zu beachten, dass nach dem Prinzip der FMEA in der Regel je Fehlerfall nur Fehlerursachen einer Quelle analysiert werden. Ausnahmen bilden Fehlerfälle, die ausschließlich durch Kombinationen mehrerer Fehlerursachen zu erreichen sind.

Der vierte Schritt, der nur bei der FMECA vorgenommen wird, erfolgt über eine Bewertung der Kritikalität, welche anhand des Schweregrads des Fehlereffekts, der Auftrittswahrscheinlichkeit der Fehlerursache und der Entdeckungswahrscheinlichkeit der aufgetretenen Fehlerursache berechnet wird.

Im letzten Schritt wird eine Optimierung durchgeführt, bei der das Ausfallrisiko des Systems reduziert werden soll. Dazu werden im System Methoden integriert, um gegen die sicherheitskritischen Effekte im System vorzugehen, so dass das Risiko minimiert wird.

Die soeben beschriebene Vorgehensweise zur Durchführung der FMEA wird üblicherweise frühest möglich im Projektverlauf durchgeführt. Auf diese Weise können vorhersehbare Fehlerzustände bereits frühzeitig während der Entwicklung eines Systems berücksichtigt werden.

3.1.3 Vor- und Nachteile

An dieser Stelle werden die Vor- und Nachteile von FMEA/FMECA erläutert, welche auch bereits vom ISTQB (vgl. [IST07]) näher betrachtet wurden.

Wie bereits erwähnt worden ist, ermöglicht die FMEA eine vorsorgende Fehlervermeidung. D.h., dass die provozierten Systemausfälle durch Fehlerzustände im System oder durch Anwendungsfehler bereits in der Anfangsphase des Projektes durch die FMEA identifiziert und bewertet werden können. Somit entsteht die Fähigkeit fehlerhafte Effekte des zu entwickelnden Systems schon im frühen Stadium zu vermeiden und das Ausfallrisiko zu senken. Folglich wird die Stabilität des Systems durch die FMEA gefördert.

Zur weiteren Unterstützung der Systemstabilität können die Ergebnisse der FMEA auch in einer Art und Weise verwendet werden, dass die in der FMEA erkenntlichen kritischen Bereiche des Systems anschließend intensiver getestet werden.

Bei systematischer Anwendung der FMEA, kann sie zudem zur Analyse des Gesamtsystems herangezogen werden. Sie kann mit ihrem Verfahren u.a. Auskunft über das Fehlerrisiko geben und die Vollständigkeit der Fehlerberücksichtigungen nachweisen. Zudem kann die FMEA bei Entwurfsentscheidungen und/oder -begründungen sinnvolle Beiträge leisten.

Bei Anwendung der FMEA ist zu berücksichtigen, dass in den meisten Fällen die Betrachtung der Folgefehler (also die Fehlerfortpflanzung) selten integriert wurde bzw. die Identifizierung derer kompliziert sein kann. Somit kann sie oftmals dem Anspruch an Vollständigkeit nicht gerecht werden.

Ferner ist zu beachten, dass die Durchführung der FMEA sehr zeitintensiv ist und gründliche Systemkenntnisse zu allen Systemteilen verlangt.

3.2 Inhalt und Zweck der FTA

Neben der FMEA/FMECA wird zur Fehleranalyse auch die FTA (engl.: Fault Tree Analysis, übers.: Fehlerbaumanalyse) verwendet. Sie wurde 1961 von den Bell Laboratories in Zusammenarbeit mit der U.S. Air Force, sowie von Boeing eingeführt (vgl. [Eri99]).

Die FTA stellt eine Methode zur systematischen Ermittlung von Schwachstellen im System bereit. Mit ihr können mögliche Systemausfälle bzw. die zugrundeliegenden Fehlerursachen aufgedeckt werden.

Bei der Analyse werden zu einem spezifizierten Fehlerzustand die dazu notwendigen Ausfallkombinationen (Fehlerursachen) identifiziert. In der FTA-typischen Baumdarstellung repräsentiert der betrachtete Fehlerzustand die Wurzel, während die Fehlerursachen als Knoten unter der Wurzel abgebildet werden. Durch logische Verknüpfungen zwischen den Fehlerursachen, kann die Fehlerherkunft präzise dargestellt werden.

Somit handelt es sich um ein Top-Down Verfahren, wogegen die FMEA ein Bottom-Up Verfahren darstellt (vgl. [Sch10]).

Die von der Fehlerbaumanalyse verfolgten Basisziele sind

- die Frühzeitige Identifizierung potentieller Fehler und Schwachstellen,
- die Eliminierung möglicher Fehlerursachen und
- die Berechnung der Zuverlässigkeit/Belastbarkeit.

Neben der FMEA stellt die FTA eine präzise Fehleranalyse dar. In dieser Arbeit werden Anforderungsspezifikationen zu dem geforderten Systemverhalten in Bezug auf mögliche Fehlerzustände benötigt, um aus diesen entsprechende Testfälle generieren zu können. Diese Angaben können gleichermaßen aus der FMEA, wie der FTA entnommen werden. Die FMEA ist dabei in der Lage durch das Bottom-Up Verfahren das nach einem Fehlerzustand folgende Systemverhalten tabellarisch einfach und verständlich wiederzugeben. Die FTA fokussiert durch ihre ggf. komplexe Baumstruktur eher die Fehlerherkunft, so dass das Systemverhalten nach einem betrachteten Fehlerzustand schwieriger zu ermitteln ist (vgl. [Ham10, Ves05, MRG11]).

Es hat sich herausgestellt, dass bei der heutigen Durchführungen einer Fehleranalyse tendenziell auf die FMEA zurückgegriffen wird, statt auf die nun weniger gängige FTA. Da diese Arbeit dem Stand der aktuellen Technik entsprechen soll, wird daher im folgenden Verlauf die FMEA als Systemanforderungsspezifikation herangezogen.

3.3 Vorgehensweise zur Testerstellung im Projekt Galileo

Die Entwicklung der ersten vier Satelliten für die IOV-Phase von Galileo wurde durch die Astrium GmbH im Jahr 2012 abgeschlossen. Im Zuge dieser Entwicklung durchlief die erstellte Satellitensoftware mehrere Validierungsphasen. Dazu wurde als Testansatz das skriptbasierte Testen (vgl. Abschnitt 2.3.2) verwendet.

Die Testskripte wurden dabei manuell von Testingenieuren in Java implementiert und innerhalb der Galileo Testumgebung ausgeführt. Zusammen mit einer Simulation der Satellitenhardware wurden auf diese Weise automatisierte Testläufe ermöglicht (nähere Erläuterungen zur Testumgebung in Abschnitt 5.5).

Im Projekt Galileo wurde für die On-Board Software ein einheitliches Verfahren zur Testerstellung festgelegt. Dabei wurden zu jeder vorhandenen Komponente im System ein oder mehrere Testskripte erstellt. Die Entwicklung dieser Skripte wurde stets nach Abschluss von (Teil-)Implementierungen von Komponenten vorgenommen, um die Korrektheit der Software in Abhängigkeit zu den gegebenen Systemanforderungen von Beginn an testen zu können. Im Falle von Erweiterungen, Fehlerkorrekturen oder Wartungsarbeiten, mussten die entsprechenden Testskripte jeweils manuell angepasst werden.

Zur Implementierung der Testskripte waren tiefgehende Systemkenntnisse bezüglich der Galileo Satelliten Software und Hardware gefordert. Diese waren notwendig, um das System in die Testfall-relevanten Zustände versetzen zu können.

Mit der hier beschriebenen Testmethode konnte auf Grund des immensen Zeitaufwandes oftmals nur ein Teil aller theoretisch möglichen Testfälle erstellt werden (insbesondere bezogen auf Szenarien mit Kombinationen von Fehlerfällen). Dennoch konnten alle relevanten Testfälle durch sinnvolle Auswahl exemplarischer Testszenarien abgedeckt werden.

Obwohl sich dieses Vorgehen also in der Praxis bewährt hat, wären dennoch Verbesserungen in vielen Bereichen wünschenswert. Dazu zählen eine bezüglich des zeitlichen Aufwands effizientere Erstellung von Tests, die Reduzierung des Wartungsaufwandes für die Tests sowie eine effiziente Möglichkeit, um Kombinationen von Fehlerfällen testen zu können.

3.4 Neue Vorgehensweise basierend auf FMEA und MBT

Die im Vergleich zum letzten Abschnitt neue Vorgehensweise, welche in dieser Arbeit anhand der Galileo Satelliten untersucht wird, greift auf den modellbasierten Testansatz (vgl. Abschnitt 2.3.4) zurück. Die bei diesem Ansatz zur Modellerstellung benötigten Spezifikationen der Systemanforderungen werden hierbei aus der FMEA entnommen.

Diese Vorgehensweise auf Basis von FMEA und MBT gliedert sich in folgende Schritte (vgl. Abbildung 1):

1. Sammlung der FMEA-Anforderungen zum gewählten Systembereich
2. Modellierung des zu testenden Systemverhaltens
3. Generierung der Testfälle und des Testcodes
4. Anbindung an die Testumgebung
5. Ausführung der Tests

Zu Beginn des Verfahrens wird aus dem betrachteten System ein Bereich (Teilsystem) ausgewählt, den es zu Testen gilt. Sobald dieser festgelegt worden ist, werden alle relevanten Informationen bezüglich des Teilsystems aus der FMEA gesammelt, um das erwartete Verhalten des Systems zu identifizieren. Die erlangten Informationen sollten hierbei auf einem vollständigen und aktuellen Stand sein.

Als nächstes wird auf Basis der ermittelten Anforderungen der FMEA ein entsprechendes Verhaltensmodell des ausgewählten Teilsystems erstellt. Auf Grund der Prinzipien der FMEA, spiegelt dieses Testmodell das erwartete Systemverhalten für alle möglichen Fehlerzustände des Teilsystems wieder. Bestehendes Systemwissen sollte bei der Modellierung nicht berücksichtigt

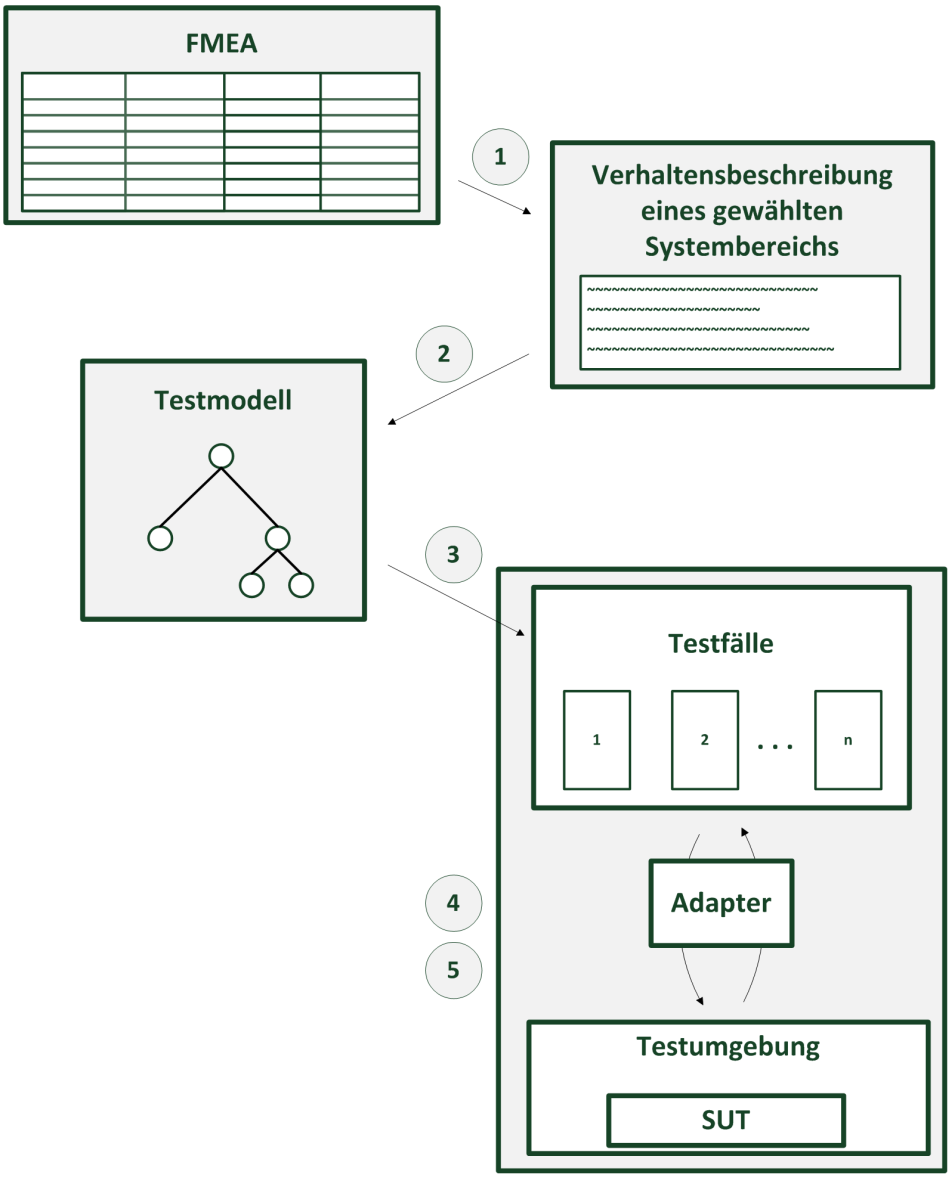

Abbildung 1: Verfahrensweise der FMEA/MBT-Methode

werden, um die Übernahme von potentiellen Systemfehlern in das Modell zu vermeiden. Zur Modellierung des Systemverhaltens kann prinzipiell zwischen diversen Modellierungsarten (z.B. endliche Automaten oder UML-Sequenzdiagramme) gewählt werden (vgl. Abschnitt 2.5). Bei der Wahl ist jedoch zu beachten, dass alle notwendigen Angaben aus der FMEA (z.B. auch zeitliches Verhalten) abbildbar sein müssen.

Damit aus dem zuvor erstellten Testmodell alle möglichen Testfälle generiert werden können, wird ein Vorgehen zum Explorieren des Testmodells benötigt. Dazu kann entweder ein entsprechendes Werkzeug einbezogen werden (s. Kapitel 4) oder ein eigenes Programm mit dem entsprechenden Algorithmus entwickelt werden. Die erzeugten Testfälle geben dabei das geforderte bzw. erwartete sequentielle Verhalten des zu testenden Teilsystems an, welches den aus der FMEA abgeleiteten Spezifikationen entspricht.

Bevor die Tests ausgeführt werden können, muss eine Verbindung zwischen dem generierten Code der Testfälle und der (ursprünglichen Galileo) Testumgebung hergestellt werden. Diese Anbindung erfolgt über einen Adapter. Während der Ausführung der Testläufe wird über den Adapter und die Testumgebung mit dem SUT kommuniziert, so dass dessen Verhalten auf Korrektheit überprüft werden kann.

Bei der Ausführung der Tests werden durch die FMEA basierten Testfälle diejenigen Fehlerfälle vorgegeben, die im System provoziert werden sollen. Daraufhin reagiert das System entsprechend seinem implementierten Verhalten. Das beobachtete Systemverhalten wird anschließend mit den im generierten Testcode erwarteten Ergebnissen verglichen, d.h. es wird abgeglichen ob das Verhalten den Spezifikationen aus der FMEA entspricht. Bei Übereinstimmung kann von einem Systemverhalten ausgegangen werden, das den gestellten Anforderungen entspricht und somit korrekt ist.

Die hier genannte Methode profitiert von einer frühen Verfügbarkeit der FMEA. Dies erlaubt es bereits sehr früh mit der Erstellung von Testmodellen zu beginnen, auch wenn zu diesem Zeitpunkt eine tatsächliche Testdurchführung auf Grund des geringen Implementierungsgrades von SUT und Testumgebung noch nicht möglich sein sollte.

Auf die in diesem Abschnitt erläuterte Vorgehensweise der FMEA/MBT-Methode wird in den beiden Kapiteln 5 und 6 durch ihre Anwendung im Projekt Galileo noch näher eingegangen. Zuvor wird im nachfolgenden Kapitel 4 jedoch noch betrachtet, welche Werkzeugunterstützung für den modellbasierten Testansatz zur Verfügung steht.

4 Werkzeugunterstützung für MBT

4.1 Klassifizierung von MBT-Werkzeugen

Zur Unterstützung der modellbasierten Testentwicklung sind unterschiedliche Werkzeuge aus dem kommerziellen und nicht-kommerziellen Bereich verfügbar. Sie sollen dem Tester die Testentwicklung durch automatisierte Mechanismen erleichtern.

Ein MBT-Werkzeug (Tool) ist durch seinen Bezug zur modellbasierten Testentwicklung gekennzeichnet. Die folgenden Aufgaben können durch ein MBT-Tool unterstützt oder gar übernommen werden (vgl. [Pra12]):

- Modellierung der Testspezifikationen
- Generierung der Testfälle, dem Testcode
- Generierung der erwarteten Ein- und Ausgaben
- Ausführung der Tests
- Vergleich der tatsächlichen und erwarteten Ausgaben
- Dokumentationen (z.B. Anforderungsüberdeckungen)

Abgesehen von den zu berücksichtigenden Funktionalitäten und der Anwendungsfreundlichkeit, sollte das gewählte Werkzeug in der zu nutzenden Umgebung und der betroffenen Domäne einsetzbar sein.

4.2 Anforderungen an MBT-Werkzeuge bzgl. Testdokumentation

Neben der eigentlichen Durchführung eines Systemtests wird in der Praxis (insbesondere vom Kunden) eine begleitende Testdokumentation zu folgenden Punkten verlangt:

- Beschreibung der Testläufe (d.h. Testplan oder Testspezifikation)
- Nachweis der kompletten Anforderungsüberdeckung
- Protokollierung der Testdurchführungen und Testergebnisse

Die erstgenannte Anforderung betrifft die Dokumentation der Testläufe. Diese soll in Form eines Testplans oder einer Testspezifikation Aufschluss über Anzahl, Art und Inhalt der durchgeführten Tests geben. Dabei ist für jeden Testfall im Einzelnen zu dokumentieren, in welche Testschritte sich dieser

gliedert, welche Aktionen pro Teilschritt durchgeführt und welche Ergebnisse jeweils erwartet werden.

Um nachzuweisen, dass die durchgeführten Tests alle Systemanforderungen abdecken, ist eine Dokumentation nötig, welche es erlaubt die gesamte Anforderungsvalidierung nachzuverfolgen. Daher wird eine Zuordnung zwischen den Systemanforderungen und den korrespondierenden Stellen in den Testläufen benötigt. Anhand dieser Zuordnung muss erkenntlich sein, mit Hilfe welcher Testschritte das SUT auf die Einhaltung welcher Systemanforderungen überprüft wird.

Zudem ist die Dokumentation der erfolgten Testdurchführungen sowie deren Ergebnisse gefordert. Anhand von Testprotokollen soll der exakte Testablauf im Detail ersichtlich sein. Weiterhin sollen die Testprotokolle Aufschluss darüber geben, welche Testschritte erfolgreich ausgeführt werden konnten und welche gescheitert sind.

Im folgenden Abschnitt werden einige MBT-Werkzeuge vorgestellt und bewertet, welche zur Durchführung dieser Arbeit gesichtet und für die beabsichtigte Modellierung als potentiell geeignet eingestuft wurden.

4.3 Übersicht einer Auswahl von MBT-Werkzeugen

4.3.1 SpecExplorer

SpecExplorer ist ein Werkzeug zum modellbasierten Spezifizieren und Testen von umzusetzenden Systemanforderungsspezifikationen (vgl. [Res12]). Bei diesem Tool handelt es sich um eine Erweiterung von Microsofts Visual Studio, welche von den Entwicklern bei Microsoft Research kostenfrei zum Download angeboten wird. Zur Benutzung von SpecExplorer 2010 muss mindestens Microsoft Visual Studio 2010 Professional oder Visual Studio 2008 SP1 zur Verfügung stehen (vgl. [SMJU10, Hei09]).

Das Microsoft Research gibt folgende Kernaufgaben an, die durch SpecExplorer unterstützt werden (vgl. [Res12]):

- Modellierung des spezifizierten Systemverhaltens
- Erkundung der möglichen Testfälle zum vorgegebenen Systemverhalten
- Vergleich des Verhaltens zwischen Modell und dem zu testenden System

Bei der Modellierung des spezifizierten Systemverhaltens werden die betrachteten Systemanforderungen in ein Modell überführt, das der zustandsbasier-

ten Notation folgt (s. Abschnitt 2.5.1). Die Erstellung erfolgt über die Implementierung von Szenarien (in der Skriptsprache Cord, engl.: Coordination Language) und Regeln (in Spec#). Im Anschluss dazu ist SpecExplorer in der Lage das resultierende Modell graphisch darzustellen und den Testcode zu den darin enthaltenen Testfällen automatisiert zu generieren.

Nachdem über einen Adapter eine Verbindung zur Testumgebung geschaffen wurde, kann der zuvor durch SpecExplorer generierte Testcode ausgeführt werden. Dabei wird analysiert, ob das zu testende System das gleiche Verhalten aufweist wie das in dem Modell spezifizierte Verhalten.

Ferner bietet SpecExplorer die Möglichkeit zur Anforderungsverwaltung, so dass die Abdeckung der Anforderungen durch die jeweiligen Testfälle nachverfolgt werden kann (vgl. [SMJU10]).

Zur Benutzung des Tools finden sich im Internet zahlreiche Hilfeseiten auf Grund der bereits erfolgten, langjährigen Etablierung dieses Werkzeugs.

4.3.2 NModel

Das NModel ist ein modellbasiertes Test- und Analysewerkzeug, das im Internet frei zur Verfügung steht. Die Modellierung der Zustandsautomaten geschieht mittels der Programmiersprache C#. Die Prinzipien des NModels entsprechen denen des SpecExplorers, dessen Entwickler (Microsoft Research) ebenfalls zur Entwicklung beitragen (vgl. [SMJU10, Mic12]).

Ausführliche Beschreibungen können im Buch *Model-based Software Testing and Analysis with C#* (s. [JVCS07]) nachgelesen werden.

4.3.3 Conformiq Tool Suite

Die Conformiq Tool Suite (vormals als Conformiq Qtronic bekannt) ist ein modellbasierter Testgenerator kommerzieller Art für funktionale Tests. Die Modellierung kann textuell oder auch über interne oder externe Editoren graphisch durchgeführt werden. Das Modell basiert dabei auf QML (engl.: Qtronic Modeling Language), einer zugeschnittene Variante des UML (Unified Modelling Language). Die Conformiq Tool Suite generiert automatisiert anhand einer Zustandsraumanalyse die entsprechenden Testpläne, Testfälle und ausführbaren Testskripte (je nach Wahl in TTCN-3, TCL oder einigen anderen ausführbaren Formaten) (vgl. [SMJU10, Ver12]).

Zu Evaluierungszwecken kann eine voll funktionsfähige Version des Werkzeuges auf Zeit beantragt werden.

4.3.4 Fokus!MBT

Fokus!MBT ist eine auf Anfrage verfügbare, vom Fraunhofer-Institut FOKUS entwickelte, Werkzeugkette zur automatisierten Spezifikation, Entwicklung und Dokumentation von modellbasierten Testszenarien. Die Modellierung des Modells geschieht graphisch anhand von UTP (s. Abschnitt 4.4.1). Ferner können u.a. auch SysML, MARTE oder SoaML zur Modellierung herangezogen werden. Der resultierende, generierte Testcode greift hierbei auf die Testsprache TTCN-3 zurück (s. Abschnitt 4.4.2, vgl.[Wen12]).

In Fokus!MBT ist eine Service-orientierte Kommunikationsinfrastruktur zu unterschiedlichen Diensten (z.B. SpecExplorer) realisiert. Anhand dieser Kommunikationswege sind Interaktionen zwischen Fokus!MBT und den entsprechenden Diensten möglich, so dass die in Fokus!MBT entwickelten Modelle zwischen diesen Diensten automatisiert transferiert werden können. Somit ist Fokus!MBT (beispielsweise für die SpecExplorer Anbindung) in der Lage sowohl die Umwandlung des Modells in das entsprechende Format, als auch die vollständige Testausführung über SpecExplorer umzusetzen (vgl. [FOK11]).

Ziel des Fokus!MBT ist ein hoher Automatisierungsgrad, durch welchen dem Entwickler eine möglichst flexible Testentwicklung angeboten werden soll. Der Anwender soll dazu möglichst wenig UTP-Kenntnisse einbringen müssen.

4.4 Testsprachen

4.4.1 UTP

Das UML Testing Profile (UTP) wurde entwickelt, um das Entwerfen, Visualisieren, Spezifizieren, Analysieren und auch Konstruieren und Dokumentieren von Testmodellierungen zu unterstützen (vgl. [Aci05]).

Die Entwicklung wurde 2001 durch eine Ausschreibung initiiert, da das bis zu diesem Zeitpunkt entwickelte UML keine Möglichkeiten zur Spezifizierung von Testkonzepten anbot. Daher wurde an der Bildung des domänenunäbhängigen UTP unter Leitung des Fraunhofer-Instituts FOKUS in Zusammenarbeit mit IBM, Rational, Telelogic, Softeam, Ericsson, Motorola und der Universität Lübeck gearbeitet (vgl. [Sch05]).

Es setzt sich aus vier Konzeptgruppen zusammen (vgl. [Bak08]):

1. **Testarchitektur**: Dieses Konzept legt die Struktur und Konfiguration der Tests fest.

2. **Testdaten**: Mit diesem Konzept werden die benötigten Testdaten modelliert.
3. **Testverhalten**: Hier werden dynamische Aspekte des Testverfahrens, wie z.B. Stimuli, Überwachungen oder Aktivitäten, festgelegt.
4. **Testzeit**: Das Konzept der Testzeit dient der Definition von zeitabhängigen Testprozeduren (z.B. Bearbeitungsdauer oder Timeouts).

Da das UTP auf Basis des UML-Metamodells arbeitet, werden UML-konforme Testspezifikationen ermöglicht.

4.4.2 TTCN-3

Die TTCN-3 (engl.: Testing and Test Control Notation) ist nach TTCN und TTCN-2 (engl.: Tree and Tabular Combined Notation) die jüngste Entwicklungsstufe dieser Testsprache. Die erste Veröffentlichung (TTCN als ISO Standard) fand 1992 statt. Der Ursprung befindet sich im Bereich der Telekommunikationsbranche, wohingegen das TTCN-3 domänen-unspezifisch ist.

Entwickelt wurde es vom ETSI (European Telecommunication Standards Institute). Anschließend wurde es von der ITU (International Telecommunication Union) als Standard angenommen.

Die Testsprache TTCN-3 wurde zur Spezifizierung und Implementierung von Tests entwickelt. Diese sind insbesondere für reaktive und verteilte Systeme geeignet, welche wohl definierte Schnittstellen aufweisen. Dazu wird beim Testen nach der Übergabe von Eingaben beobachtet und analysiert, wie das zu testende System reagiert. Das TTCN-3 unterstützt dabei u.a. funktionale, Integrations-, System-, Interoperabilitäts-, und Konformitätstests (vgl. [Zec03, Hoh06]).

4.5 Auswahl eines MBT-Werkzeugs

Zur Anwendung der in dieser Arbeit betrachteten FMEA/MBT-Methode wird ein MBT-Werkzeug benötigt. Die Tabelle 2 bietet dazu eine Übersicht der im vorhergehenden Abschnitt betrachteten Werkzeuge (vgl. [SMJU10]).

Bei der Auswahl eines geeigneten Tools sind Kriterien wie eine einfache Handhabung, eine umfassende MBT Unterstützung, ein umfangreicher Support und ein unkomplizierter Zugang zum Tool zu berücksichtigen.

Es hat sich herausgestellt, dass SpecExlorer diesen Anforderungen entspricht. Dieses Werkzeug unterstützt alle in Abschnitt 4.1 genannten MBT-Aufgaben.

	SpecExplorer	**NModel**	**Conformiq Tool Suite**	**Fokus!MBT**
Modell-sprache	Spec# Cord Skript	C#	QML	UTP (u.a.)
Modellierung	textuell	textuell	textuell/ graphisch	graphisch
Verfügbar-keit	kostenlos zum Visual Studio	kostenlos	kosten-pflichtig	auf Anfrage
Unter-stützung	Internet-beiträge, Tool Support	Internetbeiträge	Tool Support, Bedienungs-anleitung	Tool Support, Bedienungs-anleitung in Bearbeitung

Tabelle 2: Übersicht zu MBT-Werkzeugen

Zum Support lassen sich im Internet zahlreiche informative Beitrage finden. Zudem kann es, eine vorhandene Installation von Visual Studio 2010 Professional vorausgesetzt, kostenfrei eingesetzt werden.

Neben dem SpecExplorer kommt in dieser Arbeit auch das Werkzeug Fokus!MBT zum Einsatz, um eine weitere Perspektive zur Anwendung der FMEA/MBT-Methode (aus Kapitel 3) zu erlangen. Die bei diesem Werkzeug verfügbare graphische Umsetzung mittels UTP entspricht im Vergleich zum SpecExplorer einer andersartigen Herangehensweise. Das Fraunhofer FOKUS ermöglichte hierzu den freien Zugang zum Tool.

5 Die FMEA/MBT-Methode im Kontext von Galileo

Bevor in Kapitel 6 die eigentliche Anwendung der FMEA/MBT-Methode exemplarisch an Teilsystemen der Galileo Satelliten demonstriert wird, werden in diesem Kapitel zur Vorbereitung einige Galileo spezifische Grundlagen erläutert.

Als erstes wird dazu kurz auf die Hard- und Software Architektur der Galileo Satelliten eingegangen. Anschließend werden das für das autonome Fehlermanagement von Galileo Satelliten verwendete FDIR-Konzept sowie die unterschiedlichen Betriebsmodi der Galileo Satelliten erläutert. Da zur Anwendung der FMEA/MBT-Methode im nächsten Kapitel die für Galileo spezifische FMEA benötigt wird, wird diese hier ebenfalls näher beschrieben und deren Beziehung zum FDIR-Konzept verdeutlicht. Dabei wird auch gezeigt, welche Inhalte die FMEA im Detail zur Verfügung stellt und welche davon für die Erstellung eines Testmodells jeweils zu extrahieren sind.

Zum Abschluss des Kapitels werden die ursprüngliche Galileo Testumgebung sowie deren Anbindung an das in Kapitel 4 ausgewählte MBT-Werkzeug SpecExplorer behandelt. Ziel ist dabei die Konstruktion eines vollständigen MBT-Testsystems für das Projekt Galileo.

5.1 Die HW/SW Architektur der Galileo Satelliten

Die Galileo Satelliten setzen sich im Wesentlichen wie in Abbildung 2 dargestellt zusammen (vgl. [Seg11a, Lei12]). Dabei lässt sich der Aufbau eines Galileo Satelliten grob in folgende drei Teilsysteme untergliedern (vgl. die drei weißen Blöcke in Abbildung 2):

- Nutzlasten
- Plattform
- Fluglageregelung

Zur Erläuterung der drei Teilsysteme wird an dieser Stelle mit den Nutzlasten begonnen. Die Nutzlast umfasst alle Komponenten, die benötigt werden, um die eigentliche Galileo Navigationsfunktionalität bereitzustellen. Kernkomponenten der Nutzlast bilden vier hoch genaue Uhren. Es handelt sich dabei um zwei redundante Rubidium-Atomuhren sowie zwei weitere, ebenfalls redundante Wasserstoff-Maser-Uhren. Die Gangstabilitäten dieser Uhren liegen im Bereich von 1 bis 10 Nanosekunden pro Tag. Auf Basis dieser Uhren und eines ebenfalls zur Nutzlast gehörenden Signalgenerators wird

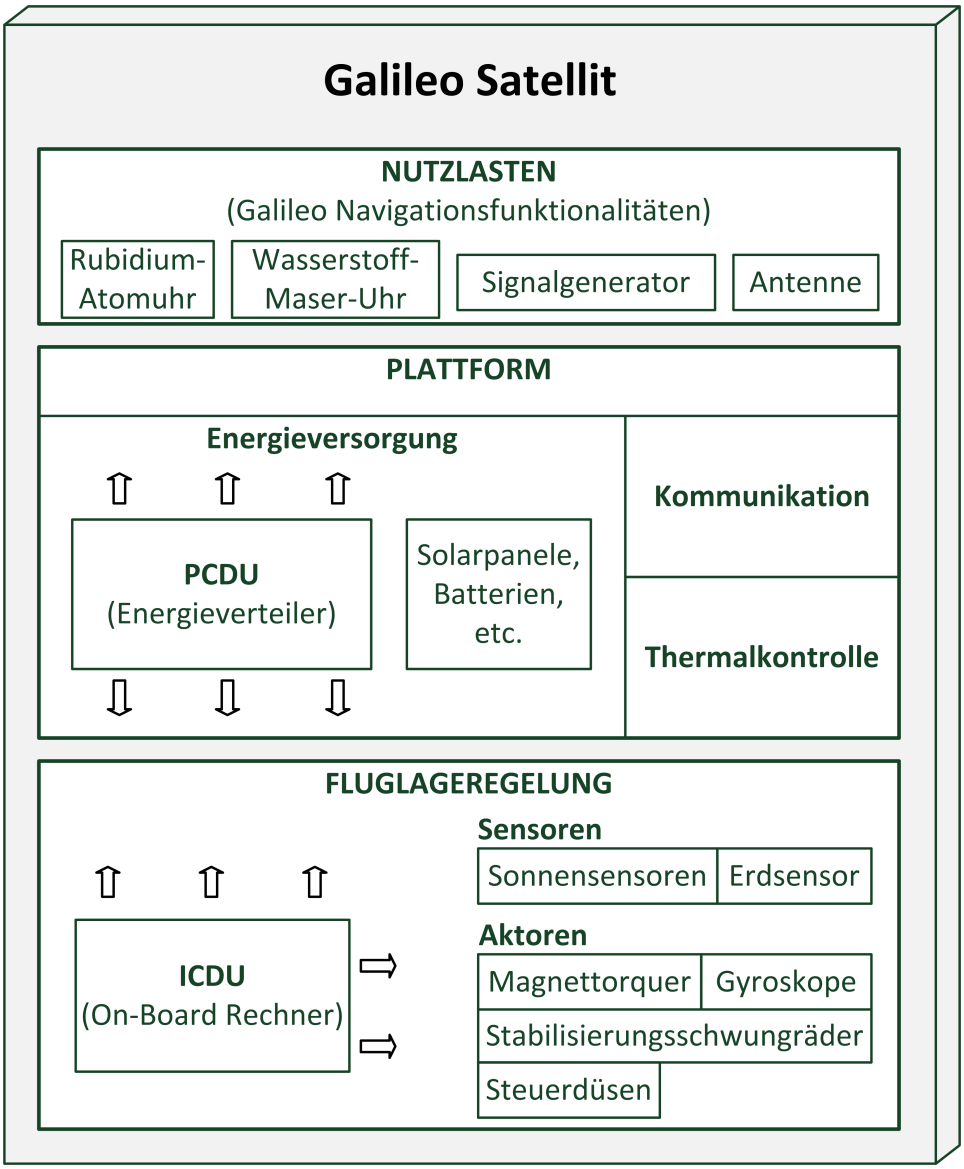

Abbildung 2: HW/SW-Architektur eines Galileo Satelliten

das Galileo Nutzdatensignal erzeugt. Weiterhin gehört zu den Nutzlasten eine L-Band Antenne, welche für die Übertragung der Signale vom Satelliten zur Erde zuständig ist (vgl. [ESA07]).

Das nächste Teilsystem, die Plattform, trägt mit der Energieversorgung, der Kommunikation, sowie der Thermalkontrolle zur Lebenserhaltung des Satelliten bei. Die PCDU (engl.: Power Conditioning & Distribution Unit) ist dabei für die Energieverteilung zu allen Untersystemen des Satelliten zuständig. Sie selbst erhält die zu verteilende Energie über die Solarzellen der Paneele oder die Batterien. Neben der Sicherstellung der Energieversorgung benötigt ein Satellit eine Sende- und Empfangsanlage zur Kommunikation mit der Bodenstation. Darüber werden Meldungen des Satelliten zur Bodenstation sowie Befehle von dort an den Satelliten gesendet. Die zuletzt genannte Thermalkontrolle hält die Temperaturen aller mechanischen, elektrischen und elektronischen Komponenten des Satelliten innerhalb der jeweils thermisch zulässigen Betriebsgrenzen. Dabei müssen extreme Temperaturschwankungen von über 100°C bis unter -100°C (je nach Position des Satelliten im Weltall) durch den Satelliten bewältigt werden können (vgl. [Lei12, KS07]).

Das in der Aufzählung zuletzt genannte Teilsystem wird durch die Fluglageregelung repräsentiert. Sie ermöglicht dem Satelliten eine rotationsstabilisierte räumliche Lage, sowie die Stabilisierung seiner Position im Orbit durch ggf. durchzuführende Steuermanöver. Dazu sind sowohl Sensoren, als auch Aktoren notwendig. Zur räumlichen Orientierung dienen die Sensoren, von denen zwei redundante Erdsensoren und zwei redundante Paare an Sonnensensoren zur groben und feinen Sonnendetektion vorhanden sind. Anhand der von den Sensoren gelieferten Orientierungsinformationen können die Aktoren den Satelliten entlang seiner Achsen stabilisieren. Dazu übertragen die Gyroskope (einfach redundant), Magnettorquer (einfach redundant) und Stabilisierungsschwungräder (drei aus vier Redundanz) die benötigten Drehmomente auf den Satellitenkörper. Bei Bedarf an größeren Positionskorrekturen, zur Änderung der Flugbahn des Satelliten, können die Steuerdüsen des Satelliten eingesetzt werden.

Ferner befindet sich ein On-Board Rechner (ICDU, engl.: Integrated Control and Data Unit) im Satelliten, der als zentrale Einheit den Satelliten steuert. Über ihn erfolgen Messungen und Auswertungen, Festlegungen der physikalischen Parameter des Satelliten (z.B. zur Lageregelung), sowie Stabilisierungen des Satelliten beim Auftreten eines Notfalls (vgl. [Lei12]).

Die Software-Architektur der Galileo Satelliten ist in ihrem Aufbau sehr eng an die beschriebene Hardware-Architektur angelehnt. Sie gliedert sich ebenso in die Teilbereiche für Nutzlasten, Plattform und Fluglageregelung. Die

in der Satelliten-Hardware vorhandenen Komponenten sind meist analog als Software-Komponenten der entsprechenden Teilbereiche wiederzufinden (z.B. der Erdsensor oder die Stabilisierungsschwungräder im Bereich der Fluglageregelung). Darüber hinaus wurden weitere, reine Softwarefunktionalitäten zur Verwaltung des Systems in einem vierten, separaten Teilbereich "System" zusammengefasst. In diesen Bereich fallen auch Funktionalitäten zu Fehlererkennung und Fehlermanagement des Satelliten, welche im nächsten Abschnitt näher betrachtet werden.

5.2 Das FDIR Konzept im Galileo Projekt

Aufgabe von FDIR (engl.: Failure Detection Isolation and Recovery) ist die Realisierung eines autonomen Fehlermanagements bzw. Selbstheilungsprozesses [Seg11b]. Innerhalb der Galileo Satelliten bildet FDIR ein relativ unabhängiges Teilsystem. Mit dessen Hilfe kann der Satellit autonom, d.h. ohne Eingriffe von außen, auf Probleme in einzelnen HW-Komponenten oder in ganzen Teilsystemen reagieren. Die Funktionalitäten von FDIR lassen sich in folgende drei Bereiche einteilen:

1. Fehlererkennung
2. Fehlerisolierung
3. Fehlerbehebung

Fehlererkennung

Die Fehlererkennung basiert auf der Überwachung von bestimmten, vorab festgelegten Systemparametern. Diese können sowohl die Software als auch die Hardware betreffen. Mittels sogenannter Monitorings werden die Werte dieser Systemparameter (z.B. die Temperatur oder der Stromverbrauch einer HW-Komponente) in regelmäßigen Zeitabständen mit definierten Referenzwerten verglichen. Sobald eine Abweichung zwischen aktuellem Wert und Referenzwert festgestellt wird, erfolgt die Einleitung von statisch zugeordneten Fehlerisolierungs- und Fehlerbehebungsmaßnahmen.

Dadurch, dass die Liste der Monitorings sequentiell durchlaufen wird, kann nie mehr als ein Monitoring gleichzeitig auf einen Fehler ansprechen. Zudem ist die Liste sinnvollerweise entsprechend der Relevanz der Monitorings sortiert. Somit werden systemkritische Defekte stets mit höherer Priorität behandelt als weniger folgenschwere Fehler. Nachdem jeweils eine Fehlerbehebung durchgeführt worden ist, können wieder alle Monitorings das System beobachten und bei Bedarf entsprechende Reaktionen auslösen.

Fehlerisolierung

Innerhalb des FDIR-Konzeptes von Galileo wird die Fehlerisolierung nicht völlig autonom durchgeführt. Grund dafür ist, dass eine Fehlerisolierung auf unterster Ebene im Allgemeinen eine detaillierte Analyse durch die Bodenstation erfordern würde. Ziel von FDIR in diesem Zusammenhang ist daher nicht die Isolierung des eigentlichen Fehlers selbst. Stattdessen besteht die Aufgabe in der Identifizierung und Isolierung einer Einheit, eines Interfaces, einer Systemfunktion oder eines Teilsystems, welches in einen Fehlerfall involviert ist.

Fehlerbehebung

Nachdem die Fehlerisolierung durchgeführt worden ist, muss zum Abschluss des Selbstheilungsprozesses die Fehlerbehebung erfolgen. Dabei muss der jeweiligen Fehlersituation entsprechend eine speziell dazu passende Fehlerbehebung stattfinden.

Da viele Komponenten des Satelliten in mehrfacher Ausführung vorhanden sind (s. Abschnitt 5.1), kann im einfachsten Fall auf die Redundanz umgeschaltet werden. Die Redundanz kann dabei einer physikalischen oder aber auch einer funktionalen Komponente entsprechen. Weitere Arten der Fehlerbehebung bestehen im Neustart des Prozessormoduls oder auch in der Verwendung des redundanten Prozessormoduls.

Ein wiederum anderes Fehlerbehebungsverfahren stellt der Wechsel in einen anderen Betriebsmodus (s. Abschnitt 5.3) des Satelliten dar. Dies bietet sich beispielsweise an, um den Satelliten in einen sicheren Betriebszustand zu bringen, falls die autonome Fehlerbehebung letztendlich scheitern sollte. Der Bodenstation wird damit Zeit verschafft, um eine weitergehende Analyse des Problems durchführen zu können.

Nachdem der Vorgang der Fehlerbehebung für ein auslösendes Monitoring durchlaufen wurde, sollte im Idealfall die ursprüngliche Fehlerursache beseitigt sein. Ist dies nicht der Fall, so wird FDIR das noch immer vorhandene Problem wiederum erkennen und versuchen mit weiteren Eskalationsstufen dagegen vorzugehen.

5.3 Die Betriebsmodi der Galileo Satelliten

Während des Betriebs eines Galileo Satelliten befindet sich dieser immer in genau einem von mehreren definierten Betriebsmodi. Welcher Modus aktiv ist hängt dabei davon ab, in welcher Phase seiner Mission (z.B. Start, Orbitalmanöver, Normalbetrieb, etc.) sich der Satellit gerade befindet. Dadurch

kann das Satellitensystem differenzieren welche Aufgaben und Funktionalitäten zulässig und auszuführen sind und welche in dem aktuell verwendeten Modus nicht erlaubt sind.

Im Einzelnen kann sich ein Galileo Satellit in folgenden Betriebszuständen befinden:

Am Boden	**Nominal**	**FDIR**
OFF Mode	Launch Mode (LM)	Launch Safe Mode (LSM)
Standby Mode (SBM)	Init Sequence (IS)	Init Safe Sequence (ISS)
	Sun Acquisition Mode (SAM)	Intermediate Safe Mode (ISM)
	Earth Acquisition Mode (EAM)	Safe Mode (SM)
	Normal Mode (NM)	
	Orbit Change Mode (OCM)	

Tabelle 3: Betriebsmodi eines Galileo Satelliten

Wie in der Tabelle 3 zu sehen ist, lassen sich die Betriebsmodi in drei verschiedene Gruppen einteilen (vgl. [Seg10]). Die erste Gruppe enthält Modi, welche verwendet werden, solange sich der Satellit am Boden befindet. Bei der zweiten Gruppe handelt es sich um die nominalen Betriebsmodi für die Startphase sowie den Aufenthalt des Satelliten im Orbit. In der dritten Gruppe befinden sich Modi, welche im Zuge von FDIR-Aktionen (s. Abschnitt 5.2) angenommen werden können, um den Satelliten in einen abgesicherten Betriebszustand zu bringen.

Im weiteren Verlauf dieser Arbeit kommen insbesondere die nominellen Betriebsmodi SAM, EAM, NM und OCM zum Einsatz. Diese werden bei Gebrauch durch den sogenannten AOC-Modus (AOC, engl.: Attitude & Orbit Control) des Satelliten repräsentiert. Auf Grund der weiteren Verwendung werden diese vier Modi im Folgenden etwas eingehender erläutert:

Sun Acquisition Mode (SAM)

Der Sonnensuchmodus SAM wird eingesetzt um den Satelliten zunächst in Richtung der Sonne auszurichten. In der Regel wird dieser Modus aktiviert nachdem die Trennung des Satelliten von der Trägerrakete erfolgt und die

anschließende Initialisierung des Bordrechners beendet ist.

Earth Acquisition Mode (EAM)

Nachdem mit Hilfe des SAM eine erste definierte Fluglage des Satelliten hergestellt worden ist, erfolgt als nächster Schritt eine verfeinerte Ausrichtung auf die Erde. In dieser Aufgabe besteht der Haupteinsatzzweck des Erdsuchmodus EAM. Ferner kann dieser Modus auch eingesetzt werden, um während der Lebenszeit des Satelliten kleinere Manöver zur Korrektur der Satellitenlage zu unterstützen.

Normal Mode (NM)

Der Modus NM kennzeichnet den normalen Arbeitsbetrieb des Satelliten. In diesem Modus erfüllt der Satellit seine eigentliche Aufgabe, welche im Senden des für die Positionsbestimmung auf der Erde nötigen Galileo Navigationssignals besteht. Der Satellit befindet sich dabei auf seiner korrekten Position in der Umlaufbahn in Ausrichtung zur Erde. Seine Fluglage wird währenddessen ohne Einsatz der Steuerdüsen geregelt.

Orbit Change Mode (OCM)

Der Orbit Change Modus dient zur Durchführung von Orbitalmanövern. Zu Beginn der Satellitenmission wird dieser verwendet, um den Satelliten in seine endgültige Orbitalposition zu bringen. Während der Lebenszeit des Satelliten werden mit Hilfe des OCM von Zeit zu Zeit korrigierende Manöver durchgeführt, um den Satelliten auf seiner korrekten Flugbahn zu halten. Diese Manöver erfolgen unter Einsatz der Steuerdüsen des Satelliten.

5.4 Beschreibung der Galileo spezifischen FMEA

5.4.1 Strukturierung der FMEA im Galileo Projekt

Die FMEA für die Galileo Satelliten ist in drei verschiedene Stufen unterteilt (vgl. Abbildung 3). Dabei analysiert die unterste Stufe N-3 solche Fehlerfälle, die in den Hardwarekomponenten selbst auftreten. Es geht beispielsweise um die Frage wie sich der Ausfall eines einzelnen Bauteiles auf die jeweilige Komponente auswirkt. Die Komponenten werden dabei isoliert voneinander betrachtet.

Die nächst höhere Stufe N-2 behandelt die Teilsysteme des Satelliten. Dazu gehören beispielsweise das Avionics-System für die Fluglageregelung oder die Nutzlastsysteme (Payload) des Satelliten (vgl. Abschnitt 5.1). Analog

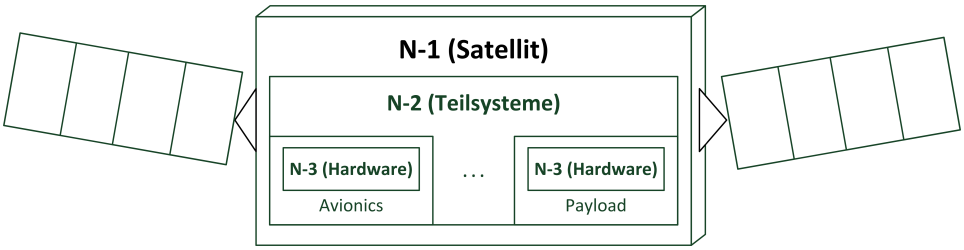

Abbildung 3: Aufteilung der FMEA in die Stufen N-1 bis N-3.

zu Stufe N-3 werden auch hier die Komponenten separat betrachtet. Beispielsweise enthält das Fluglageregelungssystem Komponenten wie den Erdsensor (engl. Earth Sensor) zum Auffinden der Position der Erde oder die Stabilisierungsschwungräder (engl. Reaction Wheels) zur Stabilisierung der Ausrichtung des Satelliten entlang seiner Achsen.

Den Bezug zum Gesamtsystem gibt die Stufe N-1 wieder. Sie nutzt die Kombination der unteren Teilsysteme und betrachtet deren Zusammenarbeit. Fehlerfälle auf dieser Stufe betreffen die Gesamtfunktionalität des Satelliten und bedeuten bei Eintreten häufig dessen Totalverlust.

Im weiteren Verlauf dieser Arbeit werden aus der Galileo FMEA hauptsächlich Analyseergebnisse aus der Stufe N-2, d.h. von Komponenten auf Teilsystemebene verwendet.

5.4.2 Inhalt der FMEA im Galileo Projekt

Die von der Astrium GmbH durchgeführte Entwicklung der ersten vier Satelliten für das Projekt Galileo ist abgeschlossen. Es liegt daher eine vollständig ausgearbeitete FMEA für diese Satelliten vor, an welcher keine zukünftigen Änderungen mehr zu erwarten sind.

In Abbildung 4 ist exemplarisch ein Auszug aus der Galileo FMEA dargestellt (vgl. [Sub09]). Die abgebildete Tabellenzeile enthält die Analyseergebnisse für einen spezifischen Fehlerfall der Erdsensor Komponente des Satelliten (FMEA-Stufe N-2, vgl. Abschnitt 5.4.1).

Earth Sensor (IRES N3) FMECA												
No	Item	FAILURE MODE	AOC MODE	RATE	LOCAL EFFECT	SYSTEM EFFECT / SYMPTOM	OFE	FAILURE INDICATION	COMPENSATION	SEVERITY	REMARK	FDIR Monitoring
ES-17	DC/DC Converter Status - Transformer, Flyback power stage, PWM Oscillator, Start-up supply unit	IRES N2 electronics is incorrectly powered. Serial digital I/F may not communicate correctly. Bulk temperature increase due to increased power dissipation ES-FMES-17	EAM-GT,GW,W; OCM-S,G; NM-R,T		Increased power consumption Serial digital I/F communication problems	EAM: Loss of earth acquisition. OCM: Loss of Attitude Determination. NM: Loss of Satellite Earth Pointing Possible loss of Pitch & Roll data and housekeeping data. Loss of correct Pitch (about Y axes) and Roll (about X axes) attitude measurements.		Increased power consumption: TM: ES 1 current is obtained via the PCDU (corresponding LCL current) Increased temperature: TM: ES1 ES Temp No serial digital I/F communication acknowledge (Secondary voltage out of specification on-ground only)	AOC ASW ES data acquisition unsuccessful, shall set the ES as not available. ES 1 current monitoring will detect power consumption outside of monitoring limits. ES1-2 - temperature monitoring would detect ES Temp limit exceeded. Reconfigure to ES2.	IIIR	Overcurrent detection provisions provided by the PCDU	FDIRREQ-571 - ES1-3 - ES 1 Current FDIRREQ-571 - ES1-2 - ES1 temperature

Abbildung 4: Auszug aus der Galileo FMEA

Die FMEA-Tabellen aller Stufen bzw. aller Komponenten sind gleichartig aufgebaut. Eine Tabellenzeile entspricht dabei jeweils einem betrachteten Fehlerfall. Die Spalten einer Zeile wiederum entsprechen einem für alle Tabellenzeilen einheitlichen Satz an Attributen zur Beschreibung eines kompletten Fehlerfalles. Die einzelnen Attribute sowie deren Bedeutung sind in Tabelle 4 aufgeführt.

NO	Die bezeichnende Nummer des Fehlers
ITEM	Welche technische Komponente betroffen ist
FAILURE MODE	Welcher Fehler aufgetreten ist
AOC MODE	Betriebsmodus des Satelliten
LOCAL EFFECT	Beschreibung wie sich der Fehler im Fehlerursprung auswirken kann
RATE	Auftrittswahrscheinlichkeit (in der Regel unbeachtet)
SYSTEM EFFECT/ SYMPTOM	Beschreibung wie sich der Fehler auf das gesamte System auswirken kann
OFE	Output Feared Event
FAILURE INDICATION	Wodurch der Fehler festgestellt werden kann
COMPENSATION	Vorgegebene Korrekturmaßnahmen
SEVERITY	Schweregrad des Fehlers
REMARK	Zusätzliche Bemerkungen
FDIR MONITORING	Auflistung der betroffenen FDIR-Monitorings

Tabelle 4: Inhalt der FMEA im Projekt Galileo

Im Detail betrachtet, wurde für das Projekt Galileo eine erweiterte Form der FMEA, die FMECA erstellt. Diese berücksichtigt zusätzlich die Kritikalität (*SEVERITY*) der in einem Fehlerfall auftretenden Auswirkungen auf das untersuchte System. Die FMECA enthält also für jeden Fehlerzustand zusätzlich eine Tabellenspalte mit einer Einstufung entsprechend seines Schweregrads.

Zur Festlegung dieser Schweregrade existieren fünf Abstufungen I - V (I := katastrophal, II := kritisch, III := bedeutend, IV:= unwesentlich, V := vernachlässigbar). Zusätzlich kann der Schweregrad mit dem Buchstaben R ergänzt sein (vgl. Abbildung 4). Dies bedeutet, dass der entsprechende Fehler auf Grund bestehender Redundanzen in der Ausstattung des Satelliten tolerierbar ist (vgl. [Seg11c]). Im Fehlerfall kann also einfach auf das redundante Teilsystem umgeschaltet werden.

Da bei der Anwendung der FMEA/MBT-Methode (Beschreibung in Abschnitt 3.4) unabhängig vom jeweiligen Schweregrad der zu testenden Fehler alle entsprechenden Testfälle generiert werden sollen, kommen die zusätzlichen Angaben der FMECA in dieser Arbeit nicht zu tragen.

Um die FMEA/MBT-Methode anwenden zu können, werden je Fehlerfall folgende Angaben aus der FMEA des Galileo Projektes benötigt:

- Für Fehlerfall relevante Betriebsmodi
- Merkmale des Fehlerfalls
- Vorgehensweise zur Fehlerbehebung

Die FMEA der Galileo Satelliten bietet zu den drei genannten Punkten alle benötigten Informationen.

Zur Spezifizierung der Betriebsmodi, in welche ein betrachteter Fehlerfall auftreten kann, verfügt die FMEA über die entsprechenden Informationen. Sowohl die Angaben der AOC-Modi (s. Abschnitt 5.3) als auch der AOC-Sub-Modi sind in der Spalte *AOC MODE* aufgelistet.

Weitere Informationen, welche die kennzeichnenden Merkmale eines Fehlerfalls wiedergeben, können aus der Spalte *FAILURE INDICATION* entnommen werden. Anhand der dort genannten Spezifikationen, kann ein eingetretener Fehlerzustand identifiziert werden. Diese Merkmale äußern sich in der Regel durch vom nominellen Verhalten des Systems abweichende Systemzustände.

Bei den letzten noch benötigten Angaben handelt es sich um Informationen darüber, welche Aktionen zur Fehlerbehebung vom System erwartet werden. Die geforderten Abläufe können aus der FMEA-Spalte *COMPENSATION* entnommen werden.

Zusätzlich können notwendige Nebenbedingungen, welche zum Zeitpunkt eines Fehlerfalls erfüllt sein müssen, in der Spalte *REMARK* aufgeführt sein (z.B. Angaben darüber, dass ein bestimmter anderer Fehler als Vorbedingung für den aktuell betrachteten Fehlerfall bereits eingetreten sein muss).

5.4.3 Beziehung zwischen FMEA, FDIR und Tests

Zwischen der FMEA, dem FDIR und den Tests für die Galileo Satelliten bestehen bestimmte Zusammenhänge, die im Folgenden erläutert werden. Dazu wird zunächst einmal auf die Umsetzung des FDIR eingegangen.

Bei der Implementierung des FDIR-Konzepts für die Galileo Satelliten wurden alle möglichen Fehlerzustände berücksichtigt, die das System mittels Überwachungs- und Fehlerbehebungsfunktionalitäten bewältigen können sollte. Die im FDIR implementierten Funktionalitäten korrespondieren also direkt mit den in der FMEA analysierten Fehlerzuständen. Da diese Funktionalitäten spezifisch und gut abgrenzbar sind, wurden sie separat von anderen Satellitenfunktionalitäten getestet, so dass sich auch die zugehörigen Testskripte klar von denen anderer Systemfunktionalitäten trennen ließen.

Um abzuschätzen wie viele Tests durch Anwendung der FMEA/MBT-Methode abgedeckt werden können, wurde ermittelt wie hoch der Anteil an FDIR Tests bezogen auf die Gesamtanzahl aller im Projekt Galileo manuell erstellen Testskripte ist. Als Ergebnis dieser Auswertung ergab sich, dass die FDIR Tests etwa 30% aller Tests ausmachen. Nur für diese 30% ist also die direkte Anwendung der FMEA/MBT-Methode sinnvoll. Für die restlichen 70% kann der MBT Teil der Methode dennoch nützlich sein. Jedoch müssen die bei der Modellierung einfließenden Anforderungen dann aus anderen Spezifikationen als der FMEA ableitbar sein.

5.5 Die Testumgebung von Galileo

Das Testen der On-Board Software für die Galileo Satelliten wird größtenteils nicht direkt auf der echten Satelliten HW durchgeführt. Die Gründe dafür liegen in den hohen Kosten sowie der zahlenmäßig eingeschränkten Verfügbarkeit dieser HW. Stattdessen kommt eine Testumgebung zum Einsatz deren Kernstück eine Simulation der gesamten Hardwarefunktionalität des Satelliten bildet. Diese Simulation setzt sich aus folgenden Hauptkomponenten zusammen:

- dem Simulator ObcSIM,
- dem Simulator RTS und
- der Steuerungskonsole SimOPS.

Der Simulator ObcSIM stellt dabei eine Simulation des Bordrechners dar. Diese umfasst den Hauptprozessor selbst sowie funktionelle Modelle von internen Bussen, ASICs, Controllern und Speichereinheiten, etc.

Ergänzt wird ObcSIM durch den Simulator RTS, welcher für die Bereitstellung der gesamten restlichen Satelliteninfrastruktur zuständig ist (z.B. für die Simulation von Untereinheiten des Satelliten wie Aktoren und Sensoren oder auch von ganzen Teilsystemen der Satellitennutzlast). Weiterhin stellt RTS realitätsnahe Verhaltensmodelle zur Verfügung, welche den Energieverbrauch oder die Wärmeverteilung innerhalb des Satelliten nachbilden.

Für die Kommunikation mit beiden Simulatoren ist die Steuerungskonsole SimOPS zuständig. Diese erlaubt das Laden, Bearbeiten, Speichern und Ausführen von Testskripten.

SimOPS arbeitet mit Java Testskripten, welche Befehle (sog. Telekommandos, TCs) an das (simulierte) SUT schicken, Antworten (sog. Telemetrien, TM) als Rückmeldung empfangen und darüber hinaus auch interne Zustände des SUT abfragen können, um so die Korrektheit des beobachteten Systemverhaltens zu überprüfen. Falls dabei ein fehlerhaftes Verhalten des SUT festgestellt wird, so erfolgt ein entsprechender Eintrag in eine Log-Datei.

Nach Hochfahren des simulierten Satelliten befindet sich dieser jeweils in demselben definierten Ausgangszustand. Um die Voraussetzungen für ein bestimmtes Testszenario zu schaffen, ist es deshalb nötig, dass das jeweilige Testskript zunächst bestimmte Initialisierungsaufrufe (z.B. An- oder Abschalten bestimmter Komponenten im SUT, Setzen bestimmter Werte im RTS Simulator, ...) durchführt. Im Anschluss daran können dann die einzelnen Testschritte basierend auf einem wohldefinierten Testszenario durchgeführt werden.

5.6 Anbindung von SpecExplorer an die Testumgebung

In diesem Abschnitt geht es darum, zu klären, wie das für die Erstellung der Testmodelle ausgewählte Werkzeug SpecExplorer (s. Abschnitte 4.3.1 und 4.5) an die im vorgehenden Abschnitt beschriebene Testumgebung des Galileo Projekts angebunden werden kann.

Auf Seiten des SpecExplorer ergibt sich diesbezüglich folgende Ausgangssituation. Bei der Testerstellung in SpecExplorer werden Testläufe erzeugt, welche im Wesentlichen aus einzelnen Funktionsausführungen mittels abstrakter Funktionsaufrufe bestehen. Diese abstrakten Funktionsaufrufe lassen sich mit Hilfe eines in C# geschriebenen Adapters in Aufrufe an externe Testumgebungen übersetzen. Demnach lässt sich SpecExplorer prinzipiell mit Schnittstellen beliebiger Testumgebungen verbinden.

Wie im vorhergehenden Abschnitt erläutert, erfolgt die Kommunikation mit der Simulation des Satelliten jedoch ausschließlich über Testskripte, welche mit Hilfe der Steuerkonsole SimOPS ausgeführt werden. Um die Galileo Testumgebung Nutzen zu können, muss am Ende der modellbasierten Testfallgenerierung deshalb ein Java Testskript stehen, welches schließlich direkt mit dem SUT kommuniziert. Es ist daher nicht möglich den oben beschriebenen Adapter direkt mit der Testumgebung zu verbinden, d.h. direkt Methoden aus der Galileo Testumgebung aufzurufen.

Um dieses Problem zu lösen, wurde auf das Kernprinzip des unter Abschnitt 2.3.3 beschriebenen Ansatzes des schlüsselwortgetriebenen Testens zurückgegriffen. Die Realisierung geschieht über ein Skript, welches in SimOPS gestartet wird und auf eingehende Befehle wartet. Dieses Skript fungiert also als Befehlsinterpreter, welcher die eingehenden Befehle (Schlüsselwörter) auswertet und nachfolgend aus SimOPS heraus die entsprechenden zugeordneten Testfunktionalitäten aus der Galileo Testumgebung aufruft. Diese Funktionalitäten sind spezifisch für das zu testende System ausgelegt und in einer oder mehreren Testbibliotheken zusammengefasst. Zur Erstellung dieser Testbibliotheken sind im Allgemeinen sehr gute Systemkenntnisse notwendig.

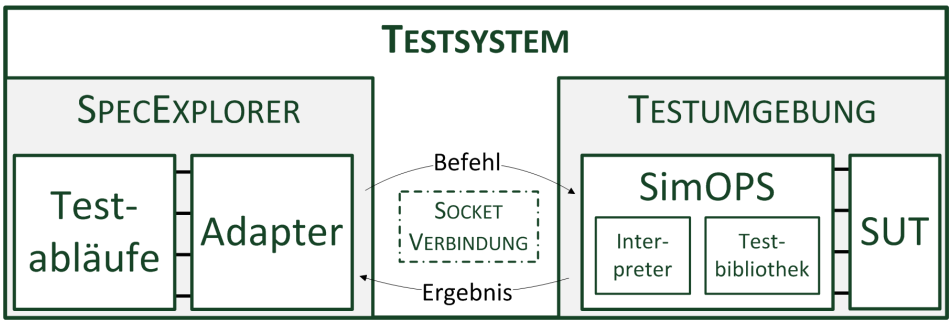

Abbildung 5: Anbindung von SpecExplorer an Galileo Testumgebung

Mit dieser Lösung (vgl. Abbildung 5) können nun also über den Adapter konkrete Befehle an das wartende Interpreter-Skript geschickt werden, wel-

ches dann auf die Simulation des SUT zugreift. Zu dieser Vorgehensweise müssen Kommunikationswege zwischen Adapter und Befehlsinterpreter geschaffen werden. In dieser Arbeit wurde dazu ein einfaches Client-Server Modell auf Basis von TCP/IP-Sockets realisiert. Dabei fällt dem Adapter die Rolle des Clients und dem Befehlsinterpreter die Rolle des Servers zu.

Zu Beginn der Ausführung eines Tests baut der Client zunächst eine Verbindung zum Server auf. Danach werden an den Server die in den Testläufen enthaltenen Schlüsselbegriffe (Befehle) zur Abarbeitung gesendet. Der Client sendet dabei jeweils nur einen Befehl. Jeder Befehl wird vom Server anhand der entsprechenden Testfunktionalitäten aus der Testbibliothek unter Nutzung von Methodenaufrufen aus der Galileo Testumgebung (s.o.) direkt in SUT-spezifische Kommandos umgesetzt. Der Server wartet danach auf die Ausführung und eine Rückmeldung durch das SUT. Solch eine Rückmeldung beinhaltet Informationen darüber, welches Systemverhalten bzw. welcher Systemstatus sich ergeben hat (z.B. Ein/Ausschaltzustand von bestimmten Komponenten, Werte von Systemparametern, etc.). Diese Informationen gibt der Server nach Erhalt je gesendetem Befehl wiederum an den Client zurück.

Zur Festlegung der erwarteten Ergebnisse müssen im Adapter die benötigten Statusvariablen aus dem Testmodell mitgeführt und nach jedem Testschritt entsprechend der erfolgten Zustandsänderungen aktualisiert werden. Damit lassen sich die vom Client über den Server erhaltenen Ergebnisse des SUT innerhalb des Adapters auf ihre Konformität überprüfen. Entsprechen diese Ergebnisse den aus dem Testmodell abgeleiteten Erwartungen, so wird der Testlauf im SpecExplorer fortgesetzt, also der nächste Befehl zum Interpreter gesendet. Andernfalls, d.h. im Falle einer fehlenden Übereinstimmung, wird der gerade ausgeführte Testlauf abgebrochen und die Testausführung des nächsten Testfalls angestoßen.

Die beschriebene Kopplung von Adapter und Befehlsinterpreter durch Verwendung von TCP/IP-Sockets hat sich als sehr robust und unproblematisch in der Handhabung erwiesen. Weiterhin erlaubt es dieser netzwerkbasierte Ansatz auf einfache Weise das Testsystem auf mehrere Rechner aufzuteilen. So könnte beispielsweise eine leistungsfähige Servermaschine für die Bereitstellung des Befehlsinterpreters sowie der Galileo Testumgebung sorgen, welche dann aus der Ferne von mehreren mit SpecExplorer ausgestatteten Arbeitsplatzrechnern genutzt werden könnte.

Im nächsten Kapitel wird die hier erläuterte FMEA/MBT-Methode auf konkrete Komponenten aus dem Fluglageregelungssystem der Galileo Satelliten angewendet.

6 Anwendung der FMEA/MBT-Methode im Projekt Galileo

In diesem Kapitel wird die FMEA/MBT-Methode beispielhaft für den Test zweier Teilsysteme der Fluglageregelung der Galileo Satelliten angewendet. In Abschnitt 6.1 geschieht dies für den Erdsensor zunächst mittels des MBT-Werkzeuges SpecExplorer. Danach wird dieser Sensor in Abschnitt 6.2 zusätzlich mit dem Werkzeug Fokus!MBT modelliert, um den Einfluss unterschiedlicher Werkzeuge auf die FMEA/MBT-Methode zu untersuchen. In einem weiteren Beispiel wird die Methode zum Test von Aktoren der Satelliten, nämlich den Stabilisierungsschwungrädern (Abschnitt 6.3) verwendet.

6.1 Beispiel 1.1: Erdsensor (SpecExplorer)

Der Erdsensor ist Bestandteil des Teilsystems der Fluglageregelung (vgl. Abschnitt 5.1). Während des Satellitenbetriebs im Weltall liegt die Hauptaufgabe dieser Sensoreinheit im Scannen des Erdhorizonts unter Nutzung seiner pyroelektrischen Infrarotsensoren (Pyrometrie: berührungslose Temperaturmessung). Mit dieser Funktion trägt der Erdsensor im Betriebsmodus EAM, wie bereits in Abschnitt 5.3 erläutert, zum Ausrichten des Satelliten bei. Bezüglich der Galileo FMEA findet sich dieser Sensor auf Stufe N-2 wieder (vgl. Abschnitt 5.4.1).

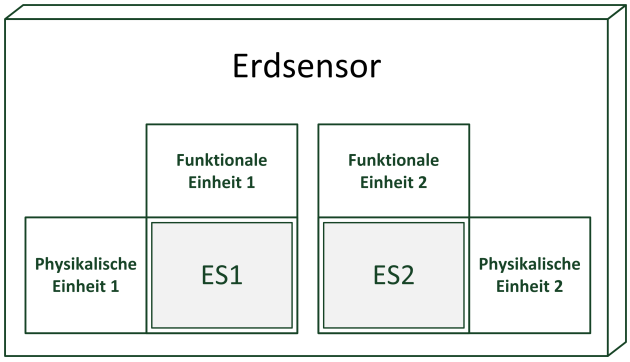

Abbildung 6: Einteilung des Erdsensors in physikalische und funktionale Einheiten

Wie man in Abbildung 6 erkennen kann, besteht der Erdsensor aus zwei identischen physikalischen Einheiten (ES1 und ES2). Auf Software Ebene sind analog dazu zwei funktionale Einheiten gegeben, denen jeweils eine physikalische Einheit direkt zugeordnet ist. Bei Betrieb des Satelliten ist zu jedem Zeitpunkt stets nur eine funktionale Einheit aktiv (vgl. Tabelle 5) geschaltet und die andere Einheit (die sogenannte Redundanz) verbleibt im passiven Zustand.

	ES1	**ES2**
Funktionale Einheit 1	X	
Funktionale Einheit 2		X

Tabelle 5: Zuordnung von physikalischen zu funktionalen Erdsensor Einheiten

Im initialen Zustand, d.h. nach Start des Satelliten, ist in der Regel die funktionale Einheit ES1 aktiviert. Somit verbleibt ES2 zunächst als Redundanz. Bei Auftreten eines Problems am Erdsensor-Teilsystem reagiert das FDIR-System des Satelliten (s. Abschnitt 5.2). Wird von diesem die aktive Sensoreinheit als fehlerhaft identifiziert, so erfolgt in einfachen Fällen nur die Umschaltung auf die Redundanz. Dieser Umschaltvorgang findet im Falle von FDIR autonom statt, kann aber bei Bedarf auch jederzeit manuell über Telekommandos der Bodenstation angestoßen werden.

6.1.1 FMEA basierte Modellierung mit SpecExplorer

Schritt 1: Ableitung der Anforderungen aus der FMEA

Im vorherigen Kapitel wurden in Abschnitt 5.4 die Inhalte der Galileo FMEA erläutert. Diese werden nun herangezogen, um daraus die für den Erdsensor bzgl. der FDIR-Funktionalität relevanten Anforderungen abzuleiten.

Fasst man die entsprechenden den Erdsensor betreffenden Analyseergebnisse aus der Galileo FMEA [Sub09] zusammen, so ergeben sich die in Tabelle 6 aufgeführten Anforderungen. Dabei handelt es sich zum einen um die relevanten Fehlerzustände bzw. Effekte und zum anderen um das geforderte Verhalten des Systems in Form der entsprechenden Fehlerbehandlungen.

Das x in den Fehler-IDs (wie z.B. in ESx_1) repräsentiert hierbei die jeweils betroffene Einheit des Erdsensors. Würde also beispielsweise ein Power Status Fehler am ES1 festgestellt werden, so entspräche dies der Fehler ID ES1_1.

Neben den AOC-Modi (vgl. Abschnitt 5.3) werden in der FMEA zusätzlich verschiedene AOC-Submodi betrachtet. Diese Detailspezifizierungen können hier jedoch vernachlässigt werden. Es ist ausreichend exemplarisch einen der Submodi in den Tests zu berücksichtigen. Für das Systemverhalten bzgl. FDIR sind also nur die unterschiedlichen AOC-(Haupt-)Modi relevant.

Fehler ID	**AOC Modus**	**Fehler-indizierung**	**Fehler-behandlung**
ESx_1	EAM; OCM; NM	Erwarteter und tatsächlicher Power Status stimmen nicht überein	Rekonfiguration auf redundanten ES
ESx_2	EAM; OCM; NM	Temperatur des Sensors außerhalb des gültigen Bereichs	Rekonfiguration auf redundanten ES
ESx_3	EAM; OCM; NM	Strom des Sensors außerhalb der zulässigen Grenzwerte	Rekonfiguration auf redundanten ES
ESx_4	EAM; OCM; NM	Erkennung fehlerhafter Housekeeping Bits	Rekonfiguration auf redundanten ES
ESx_5	EAM; OCM; NM	Erkennung fehlerhafter Housekeeping Bits (nach ES Rekonfiguration)	Rekonfiguration auf red. RIM (gleicher ES)
ESx_6	OCM; NM	Verlust der Erdpräsenz	Rekonfiguration auf redundanten ES
ES_ DEQF	EAM; OCM; NM	Beide ES als nicht funktionsfähig markiert	Umschalten des Prozessors

Tabelle 6: Aus der FMEA abgeleitete Anforderungen bzgl. Erdsensor

Der in der letzten Zeile von Tabelle 6 unter der Fehler ID "ES_DEQF" genannte Fehlerfall repräsentiert den sogenannten *Double Equipment Failure* (DEQF), welcher eintritt, sobald beide ES-Einheiten als defekt erkannt wurden. In der Regel scheitert dabei zunächst die aktive ES-Einheit und im weiteren Lebenszyklus des Satelliten (d.h. innerhalb eines unbestimmten Zeitraums) auch die Redundanz.

In der originalen Galileo FMEA wurden im Gegensatz zu obiger Zusammenfassung jeweils nur die Rekonfigurationen von ES1 auf ES2 betrachtet. Die jeweils umgekehrten Vorgänge, also die Rekonfigurationen von ES2 auf ES1, sind als implizit in der FMEA enthalten zu betrachten. Diese wurden nicht explizit aufgeführt, um den Erstellungsaufwand und Umfang der FMEA zu verringern. Verallgemeinert kann man also formulieren, dass im entsprechenden Fehlerfall jeweils auf die zu diesem Zeitpunkt redundante Einheit umzuschalten ist.

Weiterhin ist zu erwähnen, dass darüber hinaus im textuellen Bereich der FMEA festgelegt worden ist, nach welcher Zeit die autonome Korrekturmaßnahme spätestens erfolgt sein muss. Die Dauer beträgt in diesen Fällen

maximal 14 Sekunden.

Schritt 2: Auswahl der zu modellierenden Systemteile

Zunächst muss entschieden werden, welche Eigenschaften des Systems modelliert werden müssen, um das geforderte Verhalten korrekt abbilden zu können. Dabei sollen die Implementierung bzw. verfügbare Kenntnisse bzgl. des Systems außer Acht gelassen und ausschließlich auf die gegebenen Anforderungen aus der FMEA zurückgegriffen werden. Anhand der erarbeiteten Informationen kann danach mit der Modellierung des Testmodells begonnen werden.

Zur Definition des Erdsensorzustandes im Modell muss angegeben werden, welche ES-Einheit an- und welche ausgeschaltet ist. Zudem muss jeweils definiert werden, ob die Einheit als funktionsfähig oder fehlerhaft gilt, um Rekonfigurationen ein- oder ausschließen zu können. Für den Fall, dass die Rekonfiguration nach Auftreten fehlerhafter Housekeeping Bits bereits einmal stattgefunden hat (vgl. ESx_4 in Tabelle 6), muss auch darüber Auskunft gegeben werden können, so dass bei erneuter Erkennung des Fehlers auf die Redundanz der RIM-Karte[2] umgeschaltet wird (vgl. ESx_5 in Tabelle 6). Zudem wird Information darüber benötigt, in welchem AOC-Modus sich das System befindet, da gewisse Modi bestimmte Funktionalitäten ausschließen.

Statusvariable	**Erläuterung**
AOC-Modus	*NM*, *OCM* oder *EAM*
ES1 Powerstatus	*ON* oder *OFF*
ES1 Fehlerstatus	*HEALTHY* oder *FAILED*
ES2 Powerstatus	*ON* oder *OFF*
ES2 Fehlerstatus	*HEALTHY* oder *FAILED*
Flag für Monitoring zu ESx_4	Erst nachdem das Monitoring zu ESx_4 ausgelöst wurde (*TRUE*, sonst *FALSE*), darf das Monitoring zu ESx_5 angewendet werden.

Tabelle 7: Statusvariablen des Erdsensors

[2]Dem Erdsensor und anderen HW-Komponenten vorgeschaltete Interface Karte für Datensammlung und Datentransfer (engl. Remote Interface Module).

Die sich aus diesen Überlegungen für die Modellierung des Erdsensors ergebenden Statusvariablen sind zum Überblick in Tabelle 7 zusammengefasst, die Bedeutungen der einzelnen Stati sind in Tabelle 8 aufgeführt.

Variablenzustand	**Erläuterung**
ON	Der ES ist angeschaltet.
OFF	Der ES ist ausgeschaltet.
HEALTHY	Der ES wurde als gesund markiert.
FAILED	Der ES wurde als defekt markiert.
NM/OCM/EAM	Der aktuell verwendete Betriebsmodus.
Flag zu ESx_4	Das Monitoring zu ESx_4 wurde zuvor ausgelöst (TRUE)/ nicht ausgelöst (FALSE).

Tabelle 8: Erläuterung der Zustandswerte der ES Statusvariablen

Schritt 3: Umsetzen der Anforderungen in ein Testmodell

Zu Beginn der Erstellung des Testmodells ist zu beachten, dass die Werte der Statusvariablen des modellierten Systems noch nicht festgelegt sind. Daher müssen diese zunächst sinnvoll initialisiert werden. Dazu wird ES1 aktiviert (ES1: *ON*, ES2: *OFF*) und beide Einheiten werden als funktionstüchtig (ES1+2: *HEALTHY*) markiert.

Auf Grund der Tatsache, dass nach der Initialisierung stets ES1 aktiv ist, könnte im Modell nicht der Fall erreicht werden, dass beide ES-Einheiten noch gesund sind, aber anstatt ES1 die Einheit ES2 aktiv ist[3]. Deshalb wird im Modell direkt nach der Initialisierung eine Umschaltmöglichkeit auf ES2 vorgesehen (ES1: *OFF*, ES2: *ON*).

Zur Realisierung eines Fehlerzustands, werden nach Initialisierung des SUT die unterschiedlichen Fehlerfälle (s. Tabelle 6) explizit injiziert. Beispielsweise wird zur Einspeisung des Fehlerfalls ESx_2 (Temperatur außerhalb des gültigen Bereichs) im Simulator die Temperatur an der ES-Einheit über/unter das gültige Limit gesetzt. Nach der Fehlerinjizierung wird im Testsystem 14 Sekunden abgewartet, woraufhin die On-Board Software der Galileo Satelliten autonom den Fehlerzustand behandelt haben sollte. Das erwartete Verhalten wird nach der Wartezeit anhand der im Testmodell festgelegten Statusvariablen überprüft.

[3] Z.B. nach manuellem Umschalten durch die Bodenstation.

In Abbildung 7 ist der grobe Gesamtablauf im Testmodell für einen einzelnen Fehlerfall am ES exemplarisch dargestellt. Dadurch, dass sich das SUT für die betrachteten Fehlerfälle in einem von drei verschiedenen AOC-Modi befinden kann, spaltet sich das Testmodell nach der Initialisierung in drei separate Bereiche auf. Nach diesem Testschritt befindet sich das System also entweder im AOC-Modus NM, OCM oder EAM (vgl. Abbildung 8).

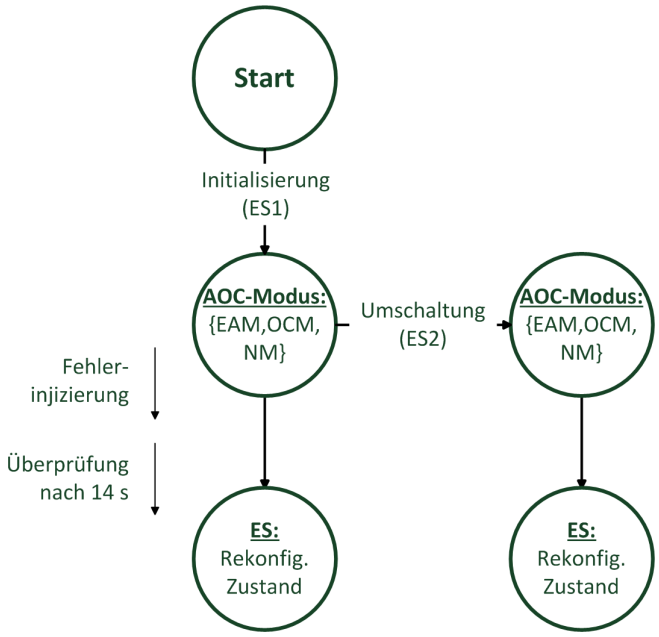

Abbildung 7: Übersicht Gesamtablauf des Testmodells

Nach Setzen eines dieser drei AOC-Modi werden im Modell bis auf eine Ausnahme alle aus Tabelle 5 bekannten Fehlerfälle berücksichtigt (vgl. Abbildung 9). Bei dieser Ausnahme handelt es sich um den Fall ESx_6 (Verlust der Erdpräsenz), welcher im EAM-Modus (d.h. während der Suche nach der Erde) nicht sinnvoll ist und für den daher das entsprechende Monitoring im SUT nicht aktiv ist.

Weiterhin ist zu beachten, dass die Überwachung für ESx_5 nur nach einer bereits erfolgten Fehlerbehebung für ESx_4 erlaubt ist. Deshalb wird für diese beiden Fehlerfälle ein eigenständiger sequentieller Pfad modelliert. Nach Einspeisung des ersten Housekeeping Bit Fehlers (für ESx_4) wird ein Flag gesetzt, so dass anschließend das Monitoring ESx_5 berücksichtigt werden kann (mittlerer Pfad auf Abbildung 9).

Grundsätzlich werden in der FMEA nur Fehlerfälle mit einer singulären Fehlerursache betrachtet. Eine Ausnahme davon bilden Fehlerfälle, die sich

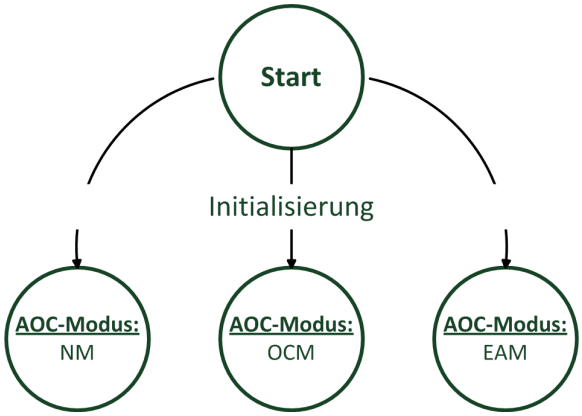

Abbildung 8: Die drei möglichen Initialisierungen in die AOC-Modi NM, OCM und EAM.

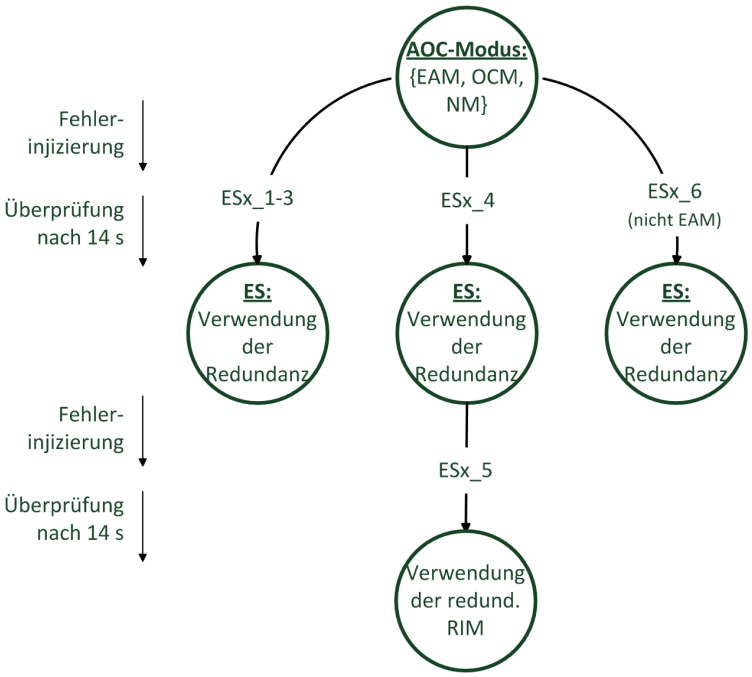

Abbildung 9: Ausnahme für EAM Modus bei Fehler ID ESx_1-6

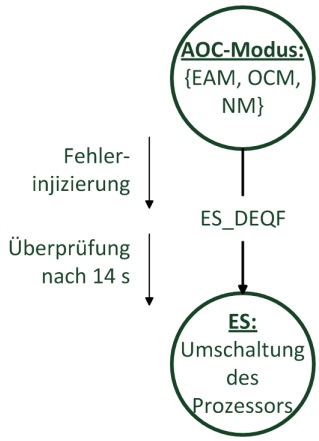

Abbildung 10: Spezielle Fehlerinjektion für die Fehler ID ES_DEQF

ausschließlich über eine kombinierte Fehlerursache herleiten lassen. Zu diesen Spezialfällen gehört der DEQF, bei dem beide ES-Einheiten ausgefallen sind. Auf direktem Wege (d.h. durch sequentielles Durchlaufen zweier Einzelfehler) lässt sich dieser Fall nicht in das Testmodell integrieren, da die entsprechenden Pfade der Einzellfehler jeweils bereits nach dem Ausfall einer ES-Einheit und erfolgtem Umschalten auf die Redundanz terminieren. Daher wird in das Testmodell parallel zu den übrigen Überwachungen eine separate Fehlerinjizierung für den ES_DEQF eingefügt (Abbildung 10). Diese beinhaltet, dass gleichzeitig und in einem Schritt beide ES-Einheiten als ausgefallen markiert werden.

Schritt 4: Erstellung des Testmodells in SpecExplorer

In SpecExplorer werden Testmodelle in Form von endlichen Automaten erstellt. Anhand von Szenarien können konkrete Verhaltensmuster des SUT abgebildet werden, welche individuell zusammengesetzt werden. Die entwickelten Szenarien können wiederum beliebig miteinander verknüpft werden, wodurch sich unterschiedlichste Gesamtszenarien konstruieren lassen. Darüber hinaus lassen sich Parameterräume definieren, sowie Einschränkungen oder Erweiterungen der Szenarien durch die Verwendung von Regeln spezifizieren. Anhand all dieser Vorgaben erstellt SpecExplorer den entsprechenden endlichen Automaten (das Testmodell).

Bei Erstellung eines neuen SpecExplorer Modells im Visual Studio, werden automatisch die dafür benötigten Dateien angelegt. Dazu gehören u.a. die Datei *Config.cord* zur Beschreibung der Szenarien und der Parameterräume sowie die Datei *Model.cs* zur Festlegung der benötigten Regeln.

Funktionsname	**Funktionalität**
Init(aocMode)	Initialisierung mit AOC-Modus
Fail_PowerStatus(esNum)	Fehlererzeugung für ESx_1
Fail_TempOverHL(esNum)	Fehlererzeugung für ESx_2 Überschreitung des oberen Limits
Fail_TempBelowLL(esNum)	Fehlererzeugung für ESx_2 Unterschreitung des niederen Limits
Fail_CurrOverHL(esNum)	Fehlererzeugung für ESx_3 Überschreitung des oberen Limits
Fail_CurrBelowLL(esNum)	Fehlererzeugung für ESx_3 Unterschreitung des niederen Limits
Fail_hkWord1(esNum)	Fehlererzeugung für ESx_4
Fail_hkWord2(esNum)	Fehlererzeugung für ESx_5
Fail_EarthPresence(esNum)	Fehlererzeugung für ESx_6
Check_EsSwitchOver()	Überprüfung ob Rekonfiguration auf redundanten ES erfolgreich war
Check_RimSwitchOver()	Überprüfung ob Rekonfiguration auf redundante RIM erfolgreich war
Check_PmSwitchOver()	Überprüfung ob Rekonfiguration auf redundantes PM erfolgreich war
SwitchOver()	Umschalten auf die Redundanz

Tabelle 9: Die Funktionen für das Erdsensor Testmodell

Im ersten Schritt zur Testmodellerstellung werden die Funktionen festgelegt, welche im späteren Testmodell enthalten sein sollen. Die Funktionen für das betrachtete Erdsensor Beispiel sind in Tabelle 9 zusammengefasst. Dabei spezifizieren die verwendeten Parameter auf welcher Erdsensor Einheit der Fehlerzustand ausgelöst (*esNum*) bzw. in welchen Betriebsmodus (*aocMode*) der Galileo Satellit versetzt werden soll. Diese Parameter werden vom Adapter in den Befehlen an den Befehlsinterpreter durchgereicht, so dass der Befehlsinterpreter wiederum in der Galileo Testumgebung die entsprechenden, passend parametrisierten Testsequenzen (nach schlüsselwortgetriebenem Testansatz, s. Abschnitt 2.3.3) auslösen kann.

Zu den entsprechenden Funktionen können mit den Bedingungen

Condition.In(aocMode, "EAM", "OCM", "NM");
und
Condition.In(esNum, "ES1", "ES2");

die jeweils erlaubten Parameterräume definiert werden.

Listing 1: Definition der Parameterräume der Init- und einer beispielhaften Fehlererzeugungsmethode

```
machine Program() : Main
{
  construct model program from CombinationConfig
}

config CombinationConfig : Main
  {
  action static void Adapter.Init(string aocMode)
    where {.
      Condition.In(aocMode, "EAM", "OCM", "NM");
    };

  action static void Adapter.Fail_TempOverHL(string
     esNum)
    where {.
      Condition.In(esNum, "ES1", "ES2");
    };

  ...
}
```

Zur Veranschaulichung ist in Listing 1 beispielhaft die Implementierung zur Definition des erlaubten Parameterraums der *Init*- sowie der Fehlererzeungsmethode *Fail_TempOverHL* dargestellt.

Neben der Definition der Parameterräume werden in der Konfiguration die möglichen Szenarien festgelegt. Diese können aus den verfügbaren Funktionen zusammengesetzt werden. Das grundlegende Szenario stellt die Fehlererzeugung und die darauf folgende Überprüfung der dazugehörigen autonomen Fehlerbehebung dar (vgl. Listing 2).

Das in Listing 2 verwendete Semikolon bewirkt einen sequentiellen Ablauf der damit getrennten Funktionen (kann auch für Szenarien verwendet werden). Werden Elemente hingegen durch das Verkettungszeichen '|' getrennt, so können diese parallel angewandt werden, so wie es in diesem Beispiel für die sieben Fehlererzeugungsmethoden der Fall ist.

Da nach Auslösen des Monitorings zu ESx_5 eine individuelle Fehlerbehebung erwartet wird, wurde für diesen Fall ein eigenes Szenario erstellt (Listing 3). Der dort verwendete Stern führt zu einem Zyklus nach Beendigung des sequentiellen Ablaufs der beiden Methoden. Dies wird angewendet, da sich die Statusvariablen des ES hierbei nicht weiter verändern und somit

Listing 2: Szenario der Standard Fehlererzeugung und der darauf folgenden Überprüfung

```
machine FailScenario(): Main
{
  (Fail_TempOverHL|Fail_TempBelowLL|
   Fail_CurrOverHL|Fail_CurrBelowLL|
   Fail_hkWord1|Fail_EarthPresence|Fail_PowerStatus);

   Check_EsSwitchOver
}
```

zum vorhergehenden Zustand zurückgekehrt wird.

Listing 3: Szenario zum Monitoring ESx_5

```
machine hk2FailScenario(): Main
{
  (Fail_hkWord2; Check_RimSwitchOver)*
}
```

Zusätzlich wird noch das Szenario für den DEQF benötigt (Listing 4).

Listing 4: Szenario für den DEQF

```
machine FailDeqfScenario(): Main
{
  Fail_DEQF; Check_PmSwitchOver
}
```

Zur Erstellung des Gesamtszenarios werden die oben beschrieben Einzelszenarien zusammengeführt. Dies ist in Listing 5 dargestellt. Das darin beim Szenario *SwitchOver* verwendete Fragezeichen bedeutet, dass das dazugehörige Szenario (bzw. diejenige Funktion) eingesetzt werden kann, aber nicht muss. Die darauf folgenden Szenarien sind dadurch in der Lage sowohl vor als auch nach diesem optionalen Szenario stattzufinden.

Listing 5: Gesamtszenario

```
machine TotalScenario() : Main
{
  Init;
  SwitchOver?;
  (FailScenario|FailDeqfScenario);
  hk2FailScenario
}
```

Um das Modell mit den zuvor definierten Parameterräumen vervollständigen zu können, muss das Gesamtszenario mittels zweier Verkettungszeichen '||' mit der Direktive *Program* verknüpft werden (Listing 6). Die unter *Program* zu findenden Parameterräume werden dadurch im Testmodell mitberücksichtigt.

Listing 6: Verknüpfung des Gesamtszenarios mit den möglichen Parameterräumen

```
machine TotalModel() : Main
{
  TotalScenario || Program
}
```

Zur weiteren Komplettierung des Modells, müssen noch Einschränkungen in Form von Regeln hinzugefügt werden. Beispielsweise darf das Monitoring ESx_6 nicht im AOC-Modus EAM überprüft werden (s. Listing 7). Solche und ähnliche Regeln können im *Model.cs* mit Hilfe von Kontrollvariablen definiert werden.

Listing 7: Ausschluss des Monitorings ESx_6 im AOC-Modus EAM.

```
static string intern_aocMode;

[Rule(Action = "Init(aocMode)")]
static void Init(string aocMode)
{
  intern_aocMode = aocMode;
}

[Rule(Action = "Fail_EarthPresence(esNum)")]
static void Fail_EarthPresence(string esNum)
{
  Condition.IsTrue(intern_aocMode != "EAM");
}
```

Zudem kann nach der Initialisierung die Umschaltung der ES-Einheiten erfolgen bevor die Fehlerinjizierungen und -überwachungen im Modell eintreten. Die Umschaltung von ES1 auf ES2 bzw. auf die nun aktuelle ES-Einheit wird in der internen Zustandsvariablen *intern_esNum* festgehalten (s. Listing 8).

Listing 8: Die optionale Erdsensor Umschaltung nach der Initialisierung.

```
static int intern_esNum = 1;

[Rule(Action = "SwitchOver()")]
static void SwitchOver()
{
  Condition.IsTrue(intern_esNum == 1);
  intern_esNum = 2;
}
```

Diese Variable unterstützt weitere Einschränkungen, so dass Fehlerinjizierungen nur auf der aktuell aktiven ES-Einheit angewendet werden können. Fehlereinspeisungen auf die passive Einheit werden nicht benötigt.

Ansonsten ist als weitere Einschränkung noch hinzuzufügen, dass nur bei Fortbestehen des Housekeeping Bitfehlers nach ES-Rekonfiguration eine RIM-Rekonfiguration durchgeführt werden darf.

Wurden alle relevanten Informationen eingefügt, so kann das gesamte Modell exploriert werden, das heißt, dass der komplett definierte Zustandsautomat durch SpecExplorer erzeugt werden kann. Das vollständige sich dabei ergebende Modell, welches aus 49 Zuständen und 82 Zustandsübergängen besteht, ist in Abbildung 11 dargestellt.

Schritt 5: Generierung der Testfälle

Nachdem das Testmodell erstellt worden ist, können alle darin enthaltenen Testfälle durch SpecExplorer generiert werden. Dazu kann in der Konfiguration zwischen den beiden Varianten *LongTests* und *ShortTests* gewählt werden (Listing 9). Bei der Variante LongTests werden aus dem Testmodell möglichst lange Testfälle berechnet, wohingegen die Variante ShortTests Testfälle mit minimaler Länge anstrebt. Bei beiden Methoden terminieren die Testfälle nur in gültigen Endzuständen. Dabei handelt es sich um diejenigen Zustände, welche nach Abschluss der Szenarien auftreten. Sowohl bei den Long- als auch bei den ShortTests werden alle Pfade des Testmodells abgedeckt.

Listing 9: Wahl der Variante ShortTests zur Generierung der Testfälle

```
machine TestSuite() : Main
{
    construct test cases where Strategy = "ShortTests"
    for TotalModel
}
```

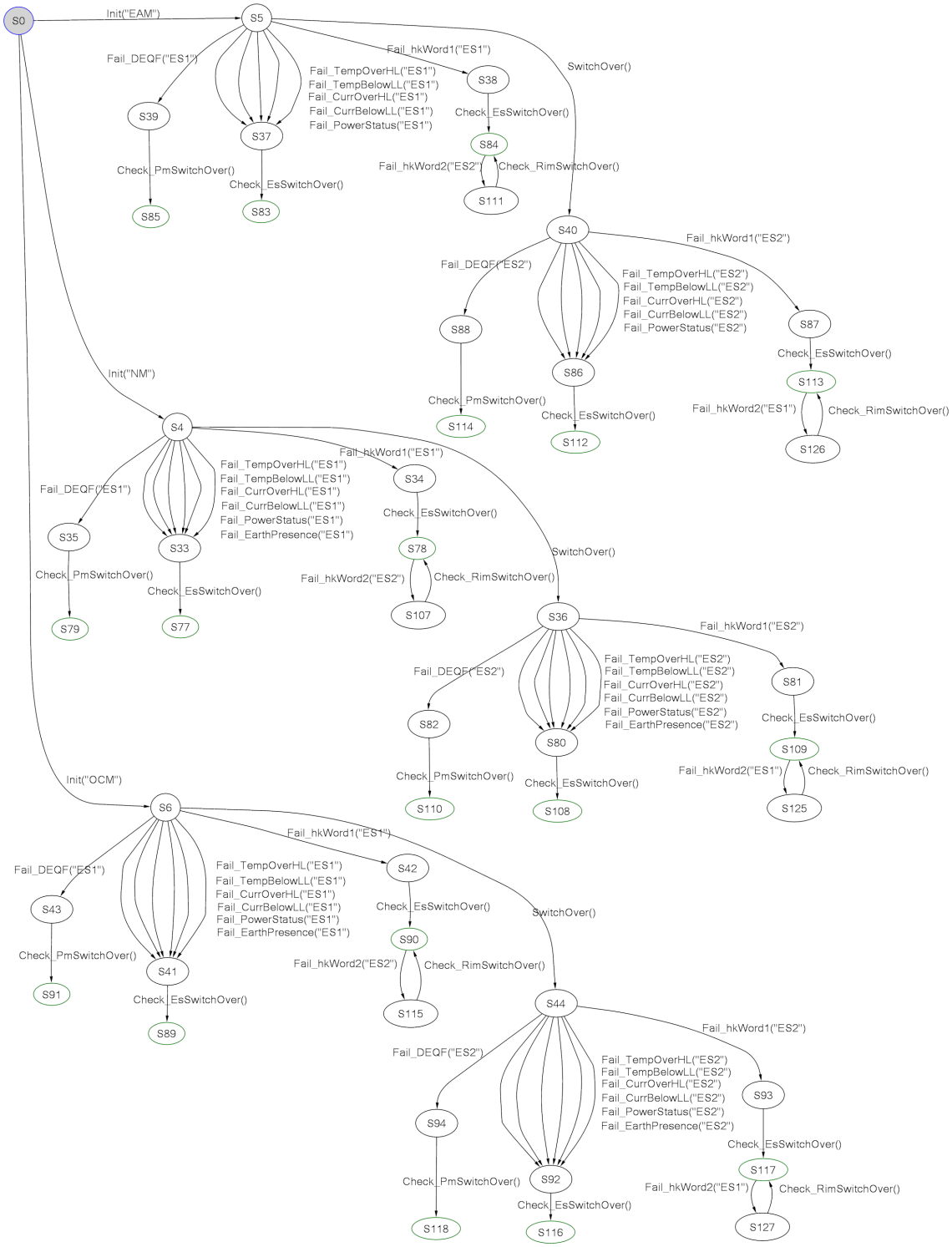

Abbildung 11: Komplettes ES-Testmodell, links oben befindet sich der Startzustand mit den drei ausgehenden Initialisierungsmöglichkeiten der AOC-Modi.

SpecExplorer ist in der Lage mit seiner Testfallgenerierung unterschiedliche Ziele zu unterstützen. Bei den ShortTests können die Überprüfungen einzelner Funktionalitäten präziser untersucht werden, wohingegen bei den LongTests Folgefehler im Gesamtverhalten besser detektiert werden können.

Die aus dem Erdsensor Testmodell in der ShortTests Variante generierten Testfälle sind in einer Gesamtübersicht in Abbildung 13 dargestellt. Insgesamt wurden 46 Testfälle automatisiert erstellt. Zur besseren Darstellung ist ein Ausschnitt davon in Abbildung 12 vergrößert abgebildet.

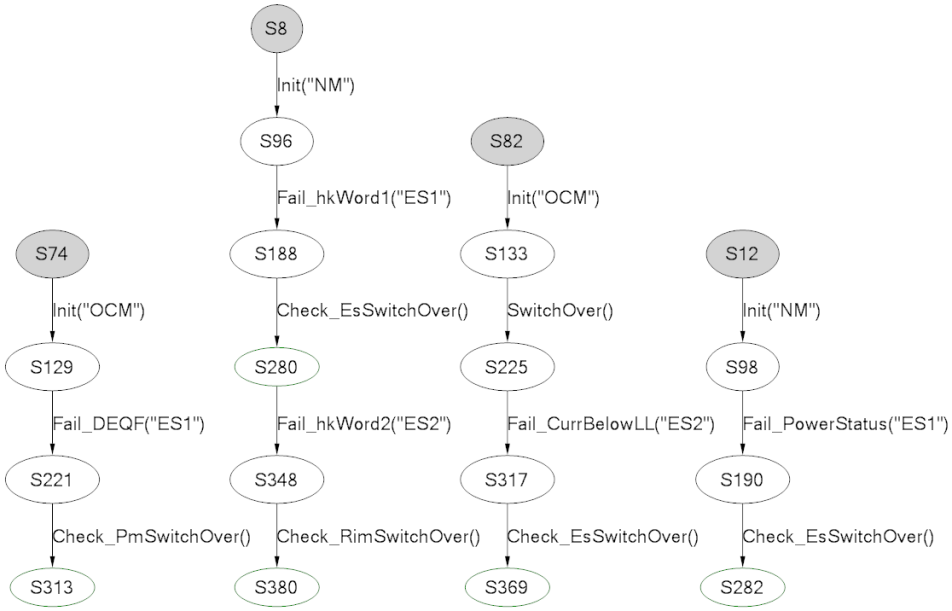

Abbildung 12: Markierter Auszug aus Abbildung 13.

Entsprechend den graphisch dargestellten Testfällen wird durch SpecExplorer der Code zu den Testläufen generiert, so dass diese sequentiell ausgeführt werden können. Somit ließ sich durch die bisher angewandte Vorgehensweise bereits Testcode zu jeglichen Testfällen erfolgreich automatisiert generieren, welcher den Anforderungsspezifikationen aus der FMEA folgt. Bevor die Testläufe an dem zu testenden System jedoch tatsächlich durchgeführt werden können, muss noch die Anbindung an das SUT bzw. die Galileo Testumgebung realisiert werden.

6.1 Beispiel 1.1: Erdsensor (SpecExplorer)

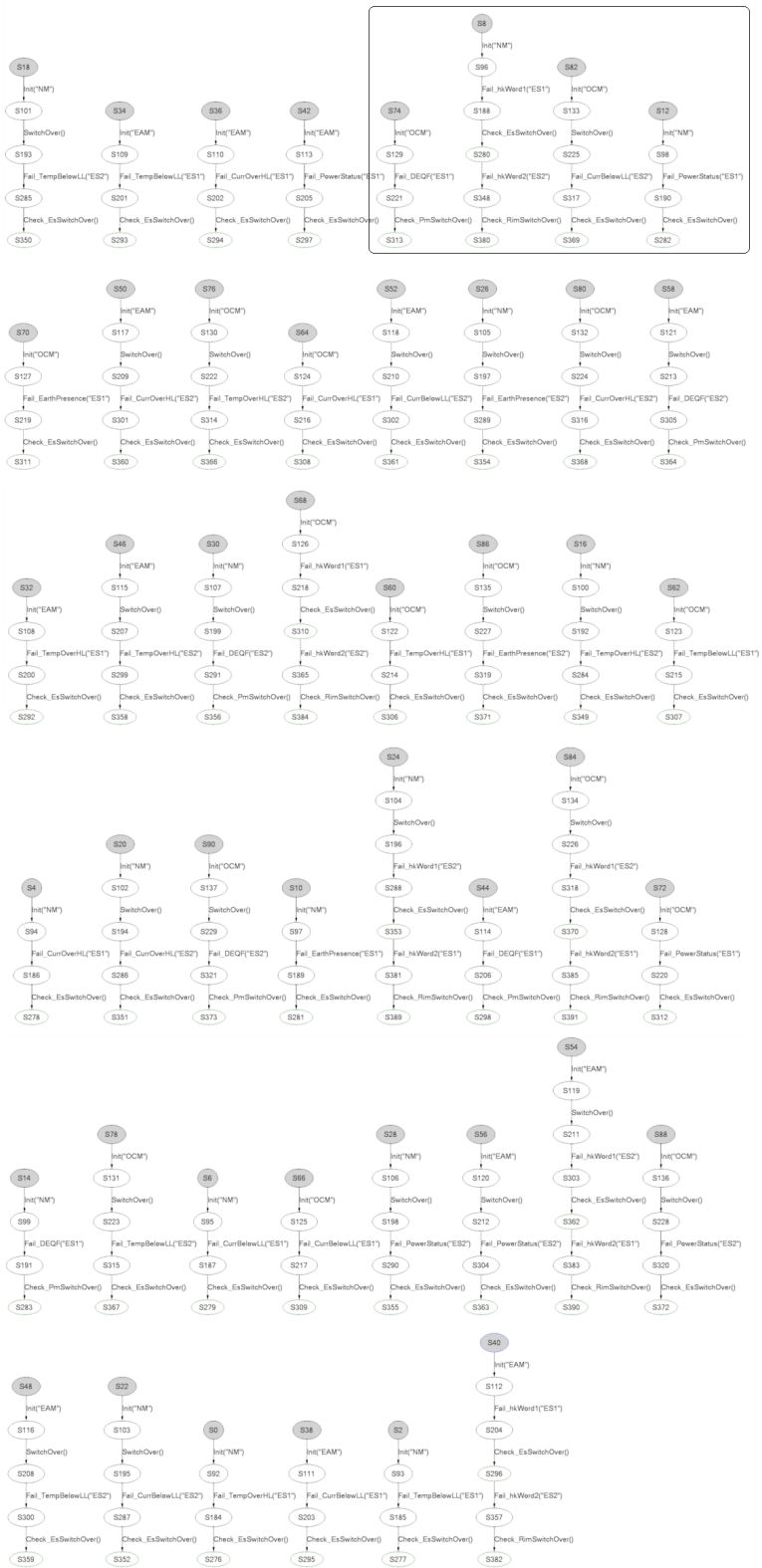

Abbildung 13: Alle Testfälle, markierter Bereich ist vergrößert in Abbildung 12 dargestellt.

Schritt 6: Definition spezifischer SUT-Steuerungsbefehle

In diesem Schritt werden die spezifischen SUT-Steuerungsbefehle definiert, welche anhand eines Befehlsinterpreters entsprechend umgesetzt werden sollen. Dazu wurde auf Seiten der Galileo Testumgebung ein in Java implementierter Befehlsinterpreter entwickelt, der über eine Socketverbindung Schlüsselwörter von SpecExplorer erhalten können soll. Diese Schlüsselwörter sollen dazu führen, dass die entsprechenden Methoden am SUT ausgeführt werden (s. Abbildung 14).

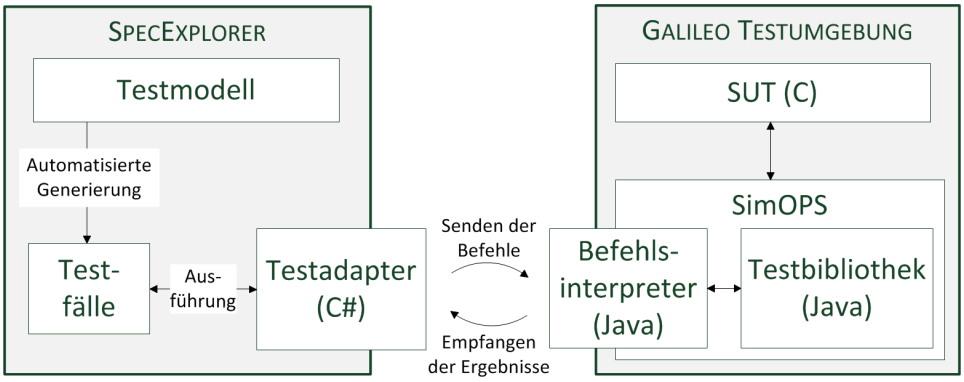

Abbildung 14: Schematischer Ablauf zur Durchführung der Testläufe

Die Steuerungsbefehle orientieren sich an den im Modell verwendeten Funktionen (s. Tabelle 9). Zusätzlich geben sie die ES-Einheit oder den geforderten AOC-Modus an, damit der Befehlsinterpreter zuordnen kann, auf welcher Einheit der Befehl auszuführen ist bzw. welcher AOC-Modus eingenommen werden soll. Bei Entgegennahme eines Steuerungsbefehls/Schlüsselworts im Befehlsinterpreter wird nach Übersetzung des Befehls der entsprechende Code generisch ausgeführt. Dies geschieht über eine eigens implementierte Testbibliothek, die über jegliche generische Testfunktionalitäten passend zu den Steuerungsbefehlen verfügt.

Beispielsweise werden nach Erhalt des Steuerungsbefehls *init EAM* alle notwendigen Initialisierungen des Satelliten ausgeführt, so dass sich die Satellitensimulation nach Abarbeitung des Befehls im AOC-Modus EAM befindet und standardmäßig ES1 aktiv ist.

Bei Bearbeitung des Steuerungsbefehls *ES1_Fail_TempOverHL* wird am ES1 ein Fehler durch eine Parametermodifizierung injiziert, so dass dort die Temperatur das obere zulässige Limit überschreitet.

Der Steuerungsbefehl *Check_EsSwitchOver* veranlasst zunächst eine War-

tezeit von 14 Sekunden, so dass das SUT auf einen injizierten Fehler mit der geforderten ES-Rekonfiguration reagieren kann. Anschließend wird überprüft, ob ein Umschalten auf die redundante ES-Einheit erfolgt ist.

Nach Abschluss eines jeden Befehls können die Werte von relevanten Statusvariablen des Systems (z.B. ob ES1 angeschaltet ist oder ob ES1 als fehlerhaft markiert ist) über den Befehlsinterpreter zurück an den SpecExplorer gesendet werden.

Analog zu obigen Beispielen wird mit den restlichen Steuerungsbefehlen verfahren.

Damit die Steuerungsbefehle innerhalb der Testläufe überhaupt zum Befehlsinterpreter gesendet werden können, bedarf es der Nutzung eines Adapters im SpecExplorer. Dieser enthält zu allen im Modell verwendeten Funktionen (Übersicht: Tabelle 9) Methoden, welche die entsprechenden Steuerungsbefehle an den Befehlsinterpreter senden.

Schritt 7: Ausführung der Testläufe

Nachdem das gesamte, zur tatsächlichen Durchführung von Testläufen benötigte Testsystem erstellt ist, kann mittels der Steuerungskonsole SimOPS der Befehlsinterpreter geöffnet und gestartet werden. Dieser kann innerhalb der Galileo Testumgebung auf die Simulation des Galileo Satelliten, d.h. auf das eigentliche SUT, zugreifen.

Sobald der Befehlsinterpreter gestartet und hochgefahren wurde, wartet dieser auf eingehende Steuerungsbefehle. Daraufhin werden nun auch die Testläufe im SpecExplorer gestartet, welche mittels des Adapters die entsprechenden Befehle über die Socket-Verbindung an den Befehlsinterpreter senden. Dieser wiederum führt den Befehl mittels der Testbibliothek auf dem SUT aus.

Nach Ausführung des Befehls werden die für diesen relevanten Statusvariablen des Systems an den Adapter zurückgesendet, wo sie mit den mitgeführten Modell-Statusvariablen verglichen werden. Bei Übereinstimmung der Statusvariablen kann davon ausgegangen werden, dass das SUT den spezifizierten Anforderungen entspricht. In diesem Fall wird der Testlauf fortgesetzt. Andernfalls wird er als fehlerhaft gekennzeichnet und es wird mit dem nächsten Testlauf fortgefahren.

Wie Abbildung 15 zu entnehmen ist, ließen sich alle der 46 generierten Testläufe erfolgreich am SUT durchführen, so dass davon ausgegangen werden kann, dass die Systemsoftware der Galileo Satelliten die Anforderungen aus

der FMEA bzgl. des Erdsensors einhält.

Test Results

o27474280@WOX50930 2012-10-18 | Run | Debug

Test run completed Results: 46/46 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
Passed	TestSuiteS56	Fmea_Es.TestSuite	
Passed	TestSuiteS46	Fmea_Es.TestSuite	
Passed	TestSuiteS48	Fmea_Es.TestSuite	
Passed	TestSuiteS70	Fmea_Es.TestSuite	
Passed	TestSuiteS32	Fmea_Es.TestSuite	
Passed	TestSuiteS82	Fmea_Es.TestSuite	
Passed	TestSuiteS14	Fmea_Es.TestSuite	
Passed	TestSuiteS74	Fmea_Es.TestSuite	
Passed	TestSuiteS34	Fmea_Es.TestSuite	
Passed	TestSuiteS20	Fmea_Es.TestSuite	

Abbildung 15: Ergebnisse der Testläufe

6.1.2 Erweiterung des FMEA-Modells mit SpecExplorer

Das entwickelte Modell des vorherigen Abschnitts basiert strikt auf den Ergebnissen der FMEA. Dies hat zur Folge, dass nicht das gesamte Verhaltensspektrum des SUT in Bezug auf mögliche Fehlerfälle berücksichtigt wird. Der Grund hierfür ist der, dass in der FMEA hauptsächlich nur singuläre Fehlerursachen betrachtet werden. Eine Ausnahme bildet das Einbeziehen von Fehlerzuständen, welche ausschließlich über das Zusammenspiel mehrerer Fehlerursachen erreicht werden können (Beispiel: Fehlerfall ESx_5). Da im Satellitensystem die Einschränkung der FMEA nicht gilt und beliebig hintereinander folgende Fehlerursachen auftreten können, wird das FMEA-Modell an dieser Stelle erweitert.

Ziele des erweiterten FMEA-Modells

Ziel der hier vorgestellten Erweiterung des FMEA-Modells ist eine natürlichere und realitätsnähere Abbildung des tatsächlichen Systemverhaltens. Diese soll sich aus den Ergebnissen der FMEA und zusätzlich unter Zuhilfenahme von wenigen, weiteren Systemkenntnissen ableiten lassen. Die Modellierung von Verhalten, welches nicht dem Systemverhalten entspricht (z.B. die explizite Fehlereinspeisung des DEQF) soll hierbei möglichst vermieden werden.

Grundsätzlich bedarf es der Möglichkeit, dass die Fehler aus Tabelle 6 entsprechend ihrer Regeln beliebig oft hintereinander auftreten können. Auf diese Weise werden im erweiterten FMEA-Modell jegliche Fehlersituationen einbezogen, die im Laufe der Betriebszeit eines Galileo Satelliten auftreten

können. Anhand dessen ergibt sich für den DEQF automatisch (dem natürlichen Systemverhalten entsprechend) die Situation, dass nach zwei aufgetretenen Fehlerfällen die beiden ES-Einheiten als fehlerhaft markiert sind. Dadurch muss dieser Fehler nicht mehr, entgegen dem tatsächlichen Systemverhalten, ausdrücklich erzeugt werden.

Weiterhin wurde bisher nach Ausführung eines Testlaufes erneut mit dem Startzustand begonnen. Dies war nötig, um eine definierte Ausgangssituation für die Einzelfehlerbetrachtungen aus der FMEA zu schaffen. Allerdings entspricht dieses Vorgehen nicht wirklich den tatsächlichen Abläufen im SUT. Dort erfolgt im operativen Betrieb natürlich nicht nach Auftreten eines Fehlers jeweils ein Rücksetzen in einen fehlerfreien Idealzustand. Die Fehler treten vielmehr nacheinander ein und bewirken jeweils einen veränderten Systemzustand, welcher wiederum den Ausgangszustand bei Eintreten des nächsten Fehlers darstellt.

Als Einschränkung ist zu erwähnen, dass nach Auftreten des DEQF davon ausgegangen wird, dass das darauf folgende Umschalten des Prozessormoduls diesen Fehlerzustand beheben kann, so dass in diesem Fall jeweils der Initialzustand erlangt wird und die Durchführung weiterer Testfälle möglich bleibt.

Umsetzung des erweiterten FMEA-Modells

Zur Anpassung des bestehenden Testmodells, werden folgende Änderungen am Testmodell vorgenommen:

- Integration aufeinanderfolgender Fehler
- DEQF als Folge von aufeinanderfolgenden Fehlern
- nach Umschalten des Prozessors: Erreichen des initialen Zustands

Dazu werden in SpecExplorer die Szenarien *TotalScenario* (vgl. Listing 5) und *TotalModel* (vgl. Listing 6) des vorherigen Models entsprechend Listing 10 modifiziert.

Listing 10: Neue Zusammensetzung des Gesamtszenarios

```
machine DoubleFailScenario() : Main
{
  Init;
  (SwitchOver?; FailScenario; hk2FailScenario?;
     FailScenario; Check_PmSwitchOver)*
}
```

```
machine TotalModel() : Main
{
  DoubleFailScenario || Program
}
```

Zu Beginn des gesamten Testmodells wird die Initialisierung (*Init*) zur Festlegung des anzuwendenden AOC-Modus vorgenommen. Im Anschluss dazu erfolgt analog zum ursprünglichen Modell die optionale Umschaltung (*SwitchOver*) auf die redundante Einheit ES2. Daraufhin können alle betrachteten Fehlerfälle injiziert werden (*FailScenario*), und im Falle eines vorhergehenden Housekeeping Bit Fehlers (ESx_4) der Fehlerfall ESx_5 (*hk2FailScenario*) folgen kann.

Zu diesem Zeitpunkt ist nun bereits eine Erdsensor-Einheit als defekt markiert worden. Dies wird innerhalb des Testablaufs durch die Überprüfung der ES-Umschaltung (*Check_EsSwitch-Over*) sowie der Fehlermarkierung innerhalb des Szenarios *FailScenario* (vgl. oben) nachgewiesen. Daraufhin wird die als gesund verbliebene ES-Einheit verwendet, auf welche dann ebenfalls jegliche Fehlerfälle (*FailScenario*) angewendet werden können.

Nach diesen zwei Schritten sollten beide ES-Einheiten vom SUT als defekt gekennzeichnet worden sein. D.h. der Fehlerzustand DEQF ist jeweils ohne explizites Injizieren erreichbar. Die für diesen erwartete autonome Fehlerbehandlung verlangt die Umschaltung (*Check_PmSwitchOver*) des Prozessormoduls, so dass nach weiteren 14 Sekunden jeweils wieder der Initialzustand erlangt wird. Der Stern nach der Klammerung (vgl. Listing 10) ermöglicht die Rückkehr in diesen gewünschten Zustand.

Zur Erstellung des erweiterten Modells mussten zusätzlich einige Änderungen in den Regeln (z.B. dass nach Prozessorumschaltung ES1 aktiviert ist), als auch Änderungen im Adapter (bezüglich der mitgeführten Statusvariablen) vorgenommen werden. Ebenso mussten entsprechende Erweiterungen zur Testbibliothek hinzugefügt werden.

Das dazu in SpecExplorer erstellte Testmodell besteht aus 61 Zuständen und 174 Zustandsübergängen. Um einen Eindruck der Größendimension zu vermitteln, wird das Modell in Abbildung 16 dargestellt.

Anhand des erstellten Modells konnte SpecExplorer mit der ShortTests Methode insgesamt 114 Testläufe generieren, welche allesamt erfolgreich durchgeführt werden konnten (s. Abbildung 17).

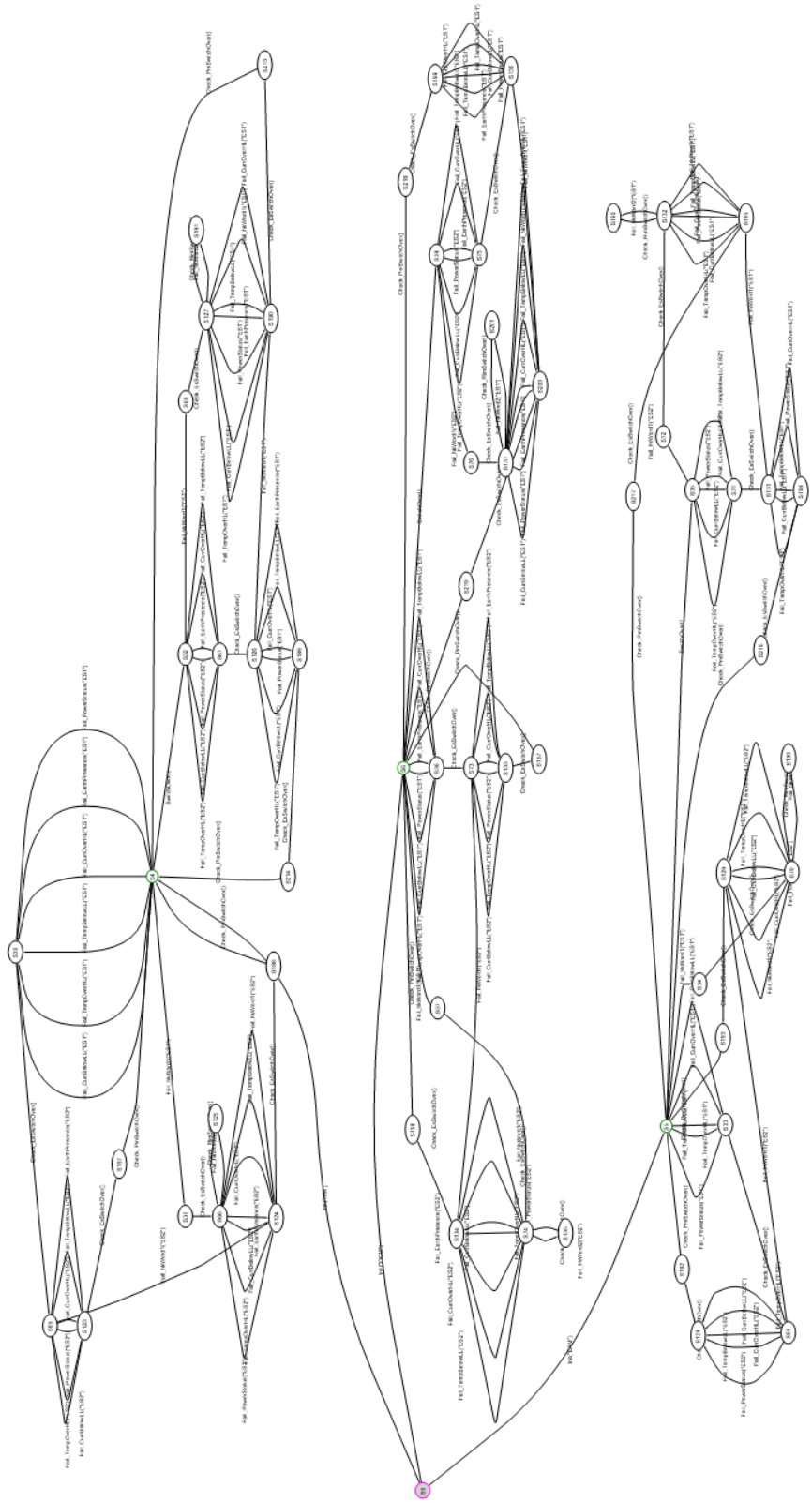

Abbildung 16: Das erweiterte FMEA-Modell des Erdsensors.

Test Results

o27474280@WOX50930 2012-10-19 | Run | Debug

Test run completed Results: 114/114 passed; Item(s) checked: 0

Result	Test Name	Project	Error Message
Passed	TestSuiteS218	Fmea_EsExt.TestSuite	
Passed	TestSuiteS216	Fmea_EsExt.TestSuite	
Passed	TestSuiteS196	Fmea_EsExt.TestSuite	
Passed	TestSuiteS14	Fmea_EsExt.TestSuite	
Passed	TestSuiteS62	Fmea_EsExt.TestSuite	
Passed	TestSuiteS48	Fmea_EsExt.TestSuite	
Passed	TestSuiteS118	Fmea_EsExt.TestSuite	
Passed	TestSuiteS8	Fmea_EsExt.TestSuite	
Passed	TestSuiteS224	Fmea_EsExt.TestSuite	
Passed	TestSuiteS28	Fmea_EsExt.TestSuite	

Abbildung 17: Die 114 erfolgreichen Testläufe in der Ergebnisliste.

6.2 Beispiel 1.2: Erdsensor (Fokus!MBT)

Das im vorherigen Abschnitt mittels SpecExplorer modellierte, erweiterte Erdsensor Beispiel wird nun unter Nutzung des Fokus!MBT Tools (vgl. Abschnitt 4.3.4) erneut durchgeführt. Dadurch kann die Anwendung der FMEA/MBT-Methode an zwei verschiedenen Werkzeugen mit unterschiedlichen Herangehensweisen erprobt werden. Die Modellierung desselben Teilsystems ermöglicht dabei einen direkten Vergleich.

Fokus!MBT verfügt über eine Anbindung zum SpecExplorer. Anhand dieser können die in Fokus!MBT erstellten Modelle automatisch in die entsprechende Modellimplementierung des SpecExplorers umgewandelt werden. In dieser Arbeit wird ausschließlich die Modellierungsfunktionalität von Fokus!MBT und dessen Schnittstelle zu SpecExplorer genutzt. Auf die weiteren, sehr umfangreichen Funktionalitäten des Fokus!MBT wird hierbei nicht eingegangen, so dass diese nicht weiter beurteilt werden können.

Im Gegensatz zum SpecExplorer erfolgt hier die Modellierung anhand einer graphischen Vorgehensweise unter Zuhilfenahme der integrierten Toolbox. Bei Veränderungen abhängiger Teile der Testspezifikation, werden die entsprechenden Anpassungen in der gesamten Testspezifikation automatisch durchgeführt.

Als erstes wird mit der UML-typischen Spezifikation der Schnittstellen (Interfaces) begonnen. Analog zur textuellen Vorgehensweise des SpecExplorers, werden die im Testmodell zu verwendenden Funktionen inklusive Parameter sowie die entsprechenden Parameterräume (s. Beispiel in Abbildung 18) definiert.

Anschließend werden Datenpartitionen (Teilmengen der Parameterräume)

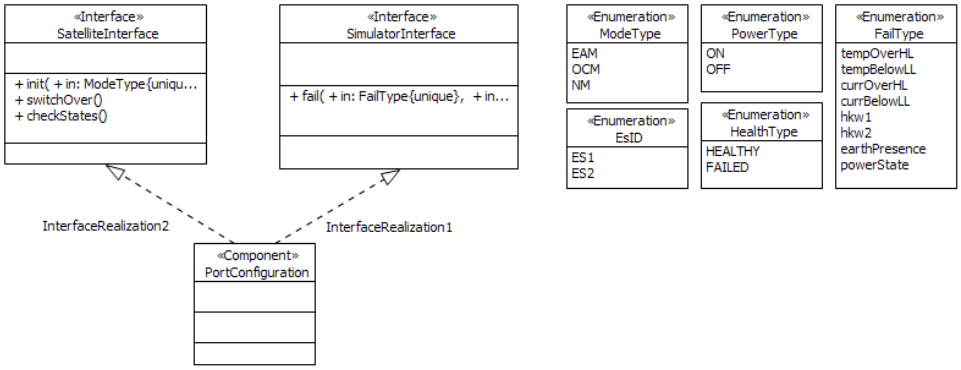

Abbildung 18: Fokus!MBT: Spezifizierung der Schnittstellen und Parameterräume

festgelegt, so dass zu den später angewendeten Funktionen/Situationen im Testmodell die entsprechenden Partitionen verwendet werden können. Diese sind in Abbildung 19 dargestellt. Dadurch können z.B. entsprechend den verschiedenen Fehlerszenarien die dort möglichen Fehlerinjizierungen spezifiziert werden.

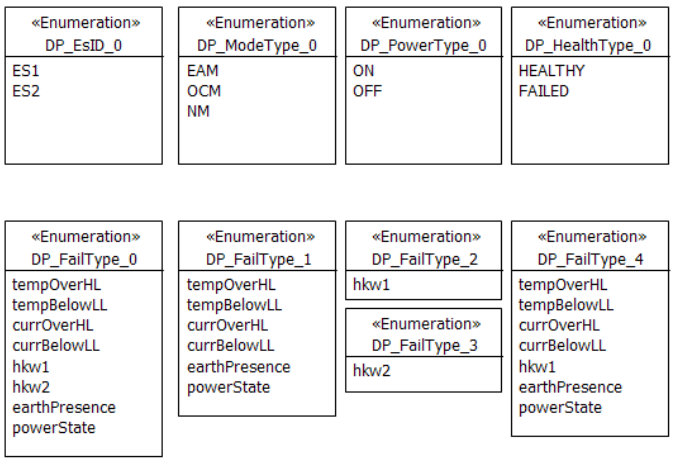

Abbildung 19: Fokus!MBT: Spezifizierung der Parameter Partitionen

Zur Erstellung des tatsächlichen Modells werden die zuvor definierten Funktionen und Datenpartitionen angewendet. Im entsprechenden Fenster zur Modellierung können anhand der gegebenen Werkzeugpalette Zustände, Zustandsübergänge, etc. ausgewählt werden, um das Verhalten des Testmodells spezifizieren zu können. Basierend auf den in Abschnitt 6.1 beschriebenen Anforderungen (bezogen auf das erweiterte FMEA-Modell), ließ sich das auf Abbildung 20 dargestellte Testmodell erstellen.

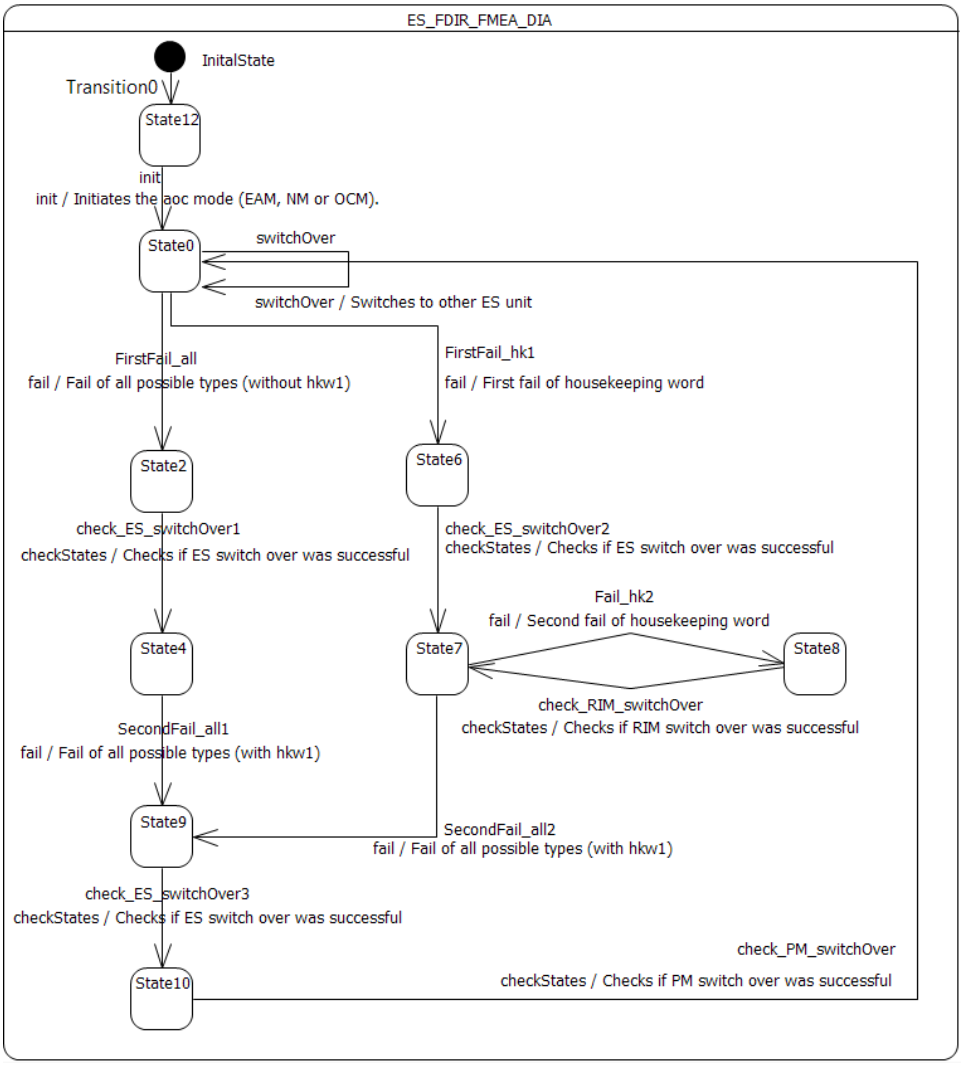

Abbildung 20: Erweitertes FMEA-Modell des Erdsensors in Fokus!MBT

Die Umsetzung von speziellen Anforderungen, wie z.B. dass der Fehler ESx_6 nicht im AOC-Modus EAM eintreten kann oder dass nur auf den aktuell aktiv verwendeten Erdsensor-Einheiten Fehler injiziert werden sollen, wird in den Hintergrundeinstellungen des Testmodells realisiert. Dies geschieht über Regelimplementierungen, die für den SpecExplorer verständlich sind (s. Listing 11).

Listing 11: Regel zu einer Fail-Funktion im Fokus!MBT

```
if(failType == FailType.earthPresence)
  Condition.IsTrue(aocMode != ModeType.EAM);

Condition.IsTrue(esID == current_esID);
```

Das nun im Testmodell abgebildete Verhalten entspricht den im vorherigen Abschnitt in SpecExplorer implementierten Szenarien und Regeln.

Nach Beendigung der Modellierung kann Fokus!MBT automatisiert die entsprechenden Implementierungen für SpecExplorer generieren. Das daraus resultierende Testmodell in SpecExplorer ist inhaltlich weitestgehend identisch zum Testmodell aus Abschnitt 6.1.2. Der einzige Unterschied betrifft direkt nach der Initialisierung die explizite Umschaltung (*SwitchOver*) der ES-Einheiten, welche eine manuell durch die Bodenstation vorgenommene Umschaltung repräsentiert.

Bisher konnte dabei ausschließlich von ES1 auf die Redundanz ES2 umgeschaltet werden (vgl. Abbildung 7). In diesem Testmodell kann die Umschaltung nun auch in entgegengesetzter Richtung (von ES2 auf ES1) erfolgen. Dies ist in Abbildung 20 durch die Eigenschleife an dem Zustand direkt nach der Initialisierung erkennbar. Dadurch ergeben sich zusätzlich zu den 114 Testläufen aus dem ersten Beispiel in Abschnitt 6.1.2 drei weitere Testläufe (einer je AOC-Modus), so dass insgesamt 117 Testläufe generiert werden.

Des Weiteren unterscheidet sich das Testmodell zum Vorherigen durch die anderen Bezeichner, sowie darin, dass Testmodellmethoden zusammengefasst wurden, so dass die gewünschten Funktionalitäten nun über die spezifizierten Parameter bestimmt werden. Zudem wird hierbei grundsätzlich der Parametertyp *Enum* statt *String* verwendet.

Aufgrund der erfolgten Schnittstellenänderungen war es weiterhin nötig Anpassungen am Adapter vorzunehmen, so dass auch mit den veränderten Funktionen und Parametern die passenden Befehle an den Befehlsinterpreter in der Galileo Testumgebung gesendet werden konnten. Dadurch war es möglich denselben Befehlsinterpreter mit der gleichen Testbibliothek weiterzuverwenden.

Die Ausführung aller automatisiert generierten Testläufe durch Fokus!MBT, konnte erfolgreich umgesetzt werden. Analog zum vorhergehenden Abschnitt kann auch hier gefolgert werden, dass das Erdsensorsystem den aus der FMEA abgeleiteten Anforderungen entspricht.

6.3 Beispiel 2: Stabilisierungsschwungräder

In diesem zweiten Anwendungsbeispiel wird die FMEA/MBT-Methode zum Test der Stabilisierungsschwungräder (engl.: Reaction Wheels, Abkürzung: RW) eingesetzt. Wie schon der Erdsensor gehört auch dieses Teilsystem zur Fluglageregelung des Satelliten und die relevanten Fehlerfallbetrachtungen finden sich ebenso wie beim Erdsensor auf Stufe N-2 der Galileo FMEA.

Die Aufgabe der Stabilisierungsschwungräder liegt in der Stabilisierung der Satellitenlage. Durch die rotierenden Schwungräder werden entlang der Rotationsachsen des Satelliten Drehmomente aufgebracht um damit von außen auf den Satelliten einwirkende Störmomente auszugleichen.

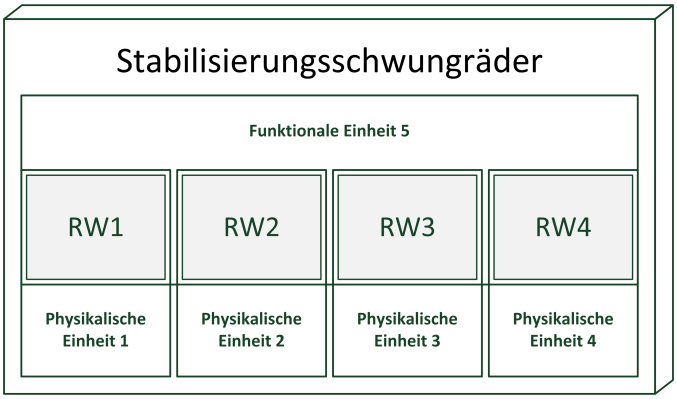

Abbildung 21: Stabilisierungsschwungräder bestehend aus vier physikalischen Einheiten gruppiert als funktionale Einheit 5.

Wie auf Abbildung 21 zu erkennen ist, bestehen die Stabilisierungsschwungräder aus vier physikalischen Einheiten. Anders als beim Erdsensor sind immer mindestens drei der vier physikalischen Einheiten aktiv und bilden zusammen jeweils eine funktionale Einheit. Alle sich daraus ergebenden funktionalen Einheiten sind in Tabelle 10 aufgelistet. Von diesen wird nach dem Start des Satelliten als Grundeinstellung die funktionale Einheit 5 verwendet, bei welcher alle vier physikalischen Einheiten aktiv genutzt werden.

	RW1	RW2	RW3	RW4
Funktionale Einheit 1		X	X	X
Funktionale Einheit 2	X		X	X
Funktionale Einheit 3	X	X		X
Funktionale Einheit 4	X	X	X	
Funktionale Einheit 5	X	X	X	X

Tabelle 10: Zuordnung von physikalischen RWs zu funktionalen Einheiten

6.3.1 FMEA basierte Modellierung mit SpecExplorer

Schritt 1: Ableitung der Anforderungen aus der FMEA

Aus der Galileo FMEA (vgl. [Sub09]) lassen sich für die Stabilisierungsschwungräder die in Tabelle 11 aufgelisteten Anforderungen ableiten.

Das x unter den Fehler-IDs steht hier analog zum Erdsensor Beispiel als Platzhalter für die jeweils betroffene physikalische Einheit, also für RW1 bis RW4. Ebenso wie beim Erdsensor Beispiel wurde auch hier auf die Berücksichtigung der AOC-Submodi verzichtet, da diese aus denselben Gründen vernachlässigbar sind. Aus der Aussage der FMEA, dass RWx_4 und RWx_6 höchstens einmal auftreten dürfen (s. Spalte Effekt in Tabelle 11) lässt sich schließen, dass das Monitoring dieser Fehlermöglichkeiten nach einmaligem Auftreten deaktiviert wird.

Bei RWx_6 handelt es sich insofern um einen Sonderfall, dass im Gegensatz zu den anderen singulären Fehlern nicht notwendigerweise eine Rekonfiguration auf die drei restlichen RWs erfolgt. Falls nämlich noch alle RWs fehlerfrei sind, verbleibt man in der vorherigen funktionalen Einheit (F5). Wenn nur noch drei RWs verwendet werden und ein nicht in der aktiven funktionalen Einheit enthaltenes RW angeschaltet ist, so wird dieses daraufhin ausgeschaltet.

Ebenso wie bei den autonomen Fehlerbehebungen am Erdsensor, soll auch hier ein eingetretener Fehlerzustand jeweils innerhalb von 14 Sekunden vom SUT behoben werden.

An dieser Stelle ist zu erwähnen, dass die FMEA der Stabilisierungsschwungräder (wie schon die FMEA des Erdsensors) in einer reduzierten Form angelegt wurde. Die Fehlerfälle wurden ausschließlich für RW1 analysiert, welches allerdings auf Grund der Gleichheit aller vier physikalischen Einheiten repräsentativ für alle RWs zu betrachten ist.

Fehler ID	**AOC Modus**	**Effekt**	**Behandlung**
RWx_1	EAM; OCM; NM; SAM	Temperatur befindet sich außerhalb des gültigen Bereichs	Rekonfiguration auf die drei verbliebenen RWs (RWx ist aus)
RWx_2	EAM; OCM; NM; SAM	Verlust des Power Status	Rekonfiguration auf die drei verbliebenen RWs (RWx ist aus)
RWx_3	EAM; OCM; NM; SAM	Strom ist außerhalb der zulässigen Grenzwerte	Rekonfiguration auf die drei verbliebenen RWs (RWx ist aus)
RWx_4	EAM; NM; SAM	Reibungsmoment ist im ungültigen Bereich (höchstens einmal)	Rekonfiguration auf die drei verbliebenen RWs (RWx bleibt an)
RWx_5	EAM; OCM; NM; SAM	Geschwindigkeit ändert sich nicht trotz Befehl oder unerwartete Geschwindigkeit	Rekonfiguration auf die drei verbliebenen RWs (RWx bleibt an)
RWx_6	EAM; OCM; NM; SAM	Geschwindigkeit befindet sich nicht im gültigen Bereich (höchstens einmal)	Rekonfiguration auf alle gesunde RWs (nur benötigte RWs bleiben an)
RW_DEQF1	EAM; SAM	Zwei RWs als fehlerhaft markiert	Ausschalten aller RWs
RW_DEQF2	OCM; NM	Zwei RWs als fehlerhaft markiert	Umschalten des Prozessors

Tabelle 11: Aus der FMEA abgeleitete Anforderungen bzgl. der Stabilisierungsschwungräder

Schritt 2: Auswahl der zu modellierenden Systemteile

Vor der Erstellung des Testmodells werden diejenigen Systemteile ausgewählt, welche nötig sind um das in der FMEA verlangte Verhalten in Form von Statusvariablen abbilden können.

Dazu werden für alle physikalischen Einheiten (RW1 bis RW4) jeweils Informationen darüber benötigt, ob die jeweilige Einheit aktiv (*ON*), inaktiv (*OFF*), fehlerfrei (*HEALTHY*) oder defekt (*FAILED*) ist. Des Weiteren muss festgehalten werden, ob die Monitorings zu RWx_4 und RWx_6 bereits einmal ausgelöst wurden, d.h. ob die entsprechenden Fehlerzustände bereits einmal eingetreten sind. Zusätzlich muss in den Statusvariablen auch der AOC-Modus abgebildet sein, da die Gültigkeit bzw. Aktivierung einiger Monitorings damit verknüpft ist.

Statusvariable	**Erläuterung**
AOC-Modus	*NM*, *OCM*, *EAM* oder *SAM*
RW1 Powerstatus	*ON* oder *OFF*
RW1 Fehlerstatus	*HEALTHY* oder *FAILED*
...	...
RW4 Powerstatus	*ON* oder *OFF*
RW4 Fehlerstatus	*HEALTHY* oder *FAILED*
Flag RWx_4	*TRUE* oder *FALSE*
Flag RWx_6	*TRUE* oder *FALSE*

Tabelle 12: Statusvariablen der Reaction Wheels

Variablenzustand	**Erläuterung**
ON	Das RW ist angeschaltet.
OFF	Das RW ist ausgeschaltet.
HEALTHY	Das RW wurde als gesund markiert.
FAILED	Das RW wurde als defekt markiert.
TRUE	Das Monitoring hat den Fehler bereits gemeldet.
FALSE	Das Monitoring hat den Fehler noch nicht gemeldet.

Tabelle 13: Erläuterung der Zustandswerte der RW Statusvariablen

Eine Übersicht der benötigten Statusvariablen ist in Tabelle 12 gegeben. In Tabelle 13 werden die möglichen Werte dieser Statusvariablen näher erläutert.

Schritt 3: Umsetzen der Anforderungen in ein Testmodell

In diesem Abschnitt werden die oben gesammelten Anforderungen aus der FMEA in ein Testmodell überführt.

Im Testmodell wird nach dem Startzustand zunächst die Initialisierung in den jeweiligen AOC-Modus durchgeführt. Im Anschluss dazu folgen die Fehlerinjizierungen zum Erzeugen von Fehlerzuständen, welche durch das SUT innerhalb von 14 Sekunden autonom behandelt werden sollen. Nach dieser Wartezeit wird der rekonfigurierte Zustand des SUT auf seine Korrektheit überprüft.

Der abstrakte Gesamtablauf des Testmodells ist in Abbildung 22 dargestellt.

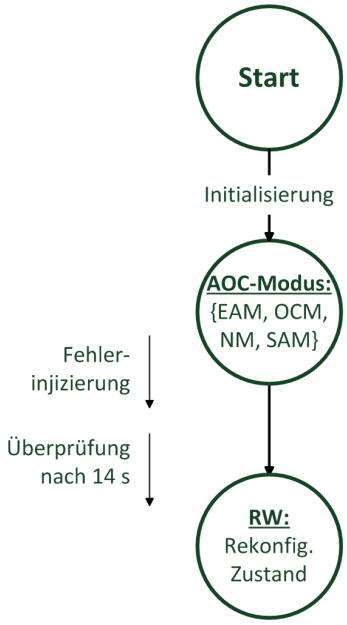

Abbildung 22: Übersicht Gesamtablauf des Testmodells

Die Stabilisierungsschwungräder werden vom Satelliten in den vier AOC-Modi EAM, OCM, NM und SAM verwendet. Im Testmodell wird daher während der Initialisierung der Stabilisierungsschwungräder u.a. einer dieser Modi aktiviert (s. Abbildung 23).

Nach der Initialisierung erfolgt die Einspeisung von Fehlern (Fehlerinjizierungen) am SUT, auf welche das System entsprechend reagieren soll. Die nach Eintreten der unterschiedlichen Fehler jeweils erwarteten Systemzustände sind in Abbildung 24 dargestellt.

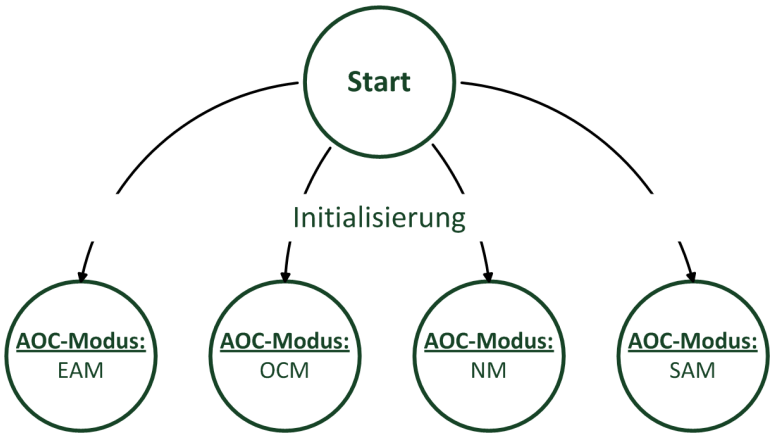

Abbildung 23: Initialisierung der Stabilisierungsschwungräder

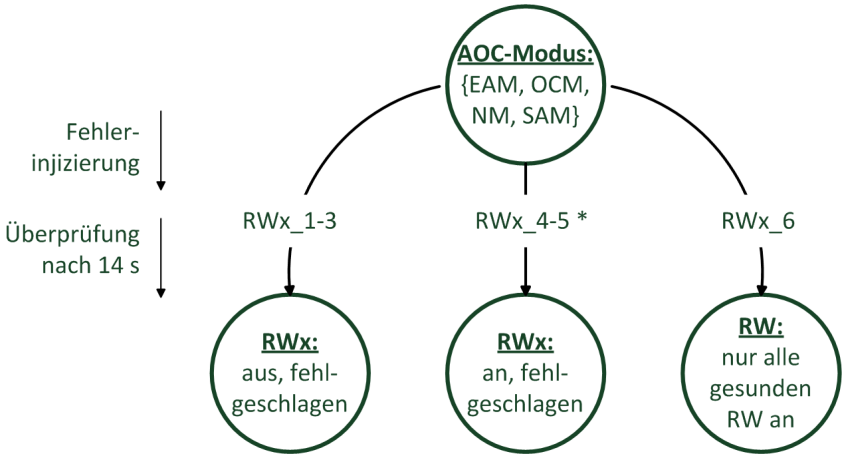

Abbildung 24: RWx_1-6: Entsprechend des Fehlerfalls werden unterschiedliche Endzustände erreicht. Für den mit einem Stern gekennzeichneten mittleren Pfad ist im AOC-Modus OCM das Eintreten des Fehlerfalls RWx_4 ausgeschlossen.

Die Fehlerfälle RWx_1 - 6 können direkt nach der Initialisierung als Einzelfehler provoziert werden. Bei den Fehlerfällen RW_DEQF1 - 2 ist im Gegensatz dazu das Vorliegen eines Doppelfehlers notwendig. D.h., dass zum Erreichen dieses Fehlers, zwei RW-Einheiten vom SUT als defekt erkannt worden sein müssen.

Im tatsächlichen operativen Betrieb des SUT würde dieser Zustand nach (einem evtl. zeitlich versetzten) Ausfall zweier RW-Einheiten erreicht werden und sich somit direkt aus dem Systemverhalten ergeben. Um das Auftreten

von Doppelfehlern dennoch testen zu können, werden diese ebenso wie die anderen Fehler (und analog dem Erdsensor Beispiel aus 6.1) nach der Initialisierung (künstlich in einem Schritt) provoziert.

Abbildung 25 stellt die beiden verschiedenen DEQF-Fälle mit ihren entsprechenden Folgesituationen dar.

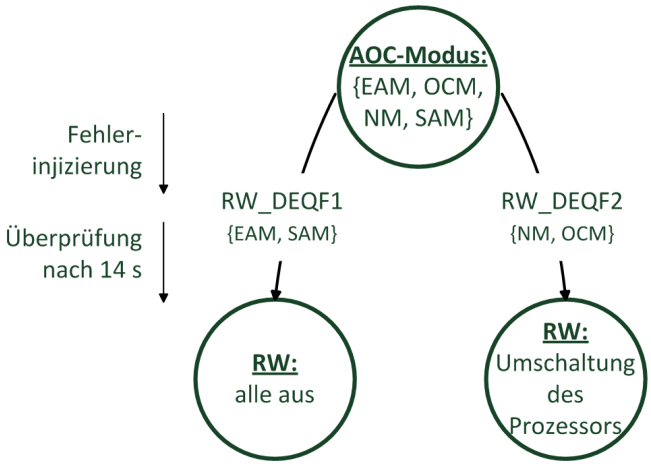

Abbildung 25: RW_DEQF1-2: Je nachdem welcher AOC-Modus aktiv ist, führt der DEQF zu einer unterschiedlichen Fehlerbehandlung.

Schritt 4: Erstellung des Testmodells in SpecExplorer

Die Implementierung des Testmodells in SpecExplorer verläuft analog zur Testmodellerstellung für den Erdsensor. Als erstes werden die benötigten Funktionen zur Initialisierung, der Fehlereinspeisung und zur Überprüfung der Fehlerbehebung festgelegt (s. Tabelle 14).

Funktionsname	**Funktionalität**
init(aocMode)	Initialisierung mit AOC-Modus
fail_TempOverHL(rwNum)	Fehlererzeugung für RWx_1 Überschreitung des oberen Limits
fail_TempBelowLL(rwNum)	Fehlererzeugung für RWx_1 Unterschreitung des niederen Limits
fail_PowerStatus(rwNum)	Fehlererzeugung für RWx_2
fail_CurrOverHL(rwNum)	Fehlererzeugung für RWx_3 Überschreitung des oberen Limits
fail_CurrBelowLL(rwNum)	Fehlererzeugung für RWx_3 Unterschreitung des niederen Limits
fail_FrictionOverHL(rwNum)	Fehlererzeugung für RWx_4 Überschreitung des oberen Limits
fail_FrictionBelowLL(rwNum)	Fehlererzeugung für RWx_4 Unterschreitung des niederen Limits
fail_SpeedJump(rwNum)	Fehlererzeugung für RWx_5
fail_SpeedOverHL (rwNum, prepare)	Fehlererzeugung für RWx_6 Überschreitung des oberen Limits prepare: benötigte funktionale Einheit
fail_SpeedBelowLL (rwNum, prepare)	Fehlererzeugung für RWx_6 Unterschreitung des niederen Limits prepare: benötigte funktionale Einheit
fail_DEQF(rwNum1, rwNum2)	Überprüfung ob Rekonfiguration auf redundanten ES erfolgreich war
check_States()	Überprüfung ob Rekonfiguration erfolgreich war

Tabelle 14: Die Funktionen für das Testmodell der Stabilisierungsschwungräder

Auf Grund der zeichenorientierten Socket-Kommunikation zwischen Spec-Explorer und Befehlsinterpreter wird auch hier einfachheitshalber für alle Parameter derselbe Parametertyp String verwendet. In Listing 12 werden zu den jeweiligen Funktionen die entsprechenden Parameterräume definiert.

Listing 12: Definition der Parameterräume

```
machine Program() : Main
{
  construct model program from CombinationConfig
}


config CombinationConfig : Main //extended config for
    combinations
{
  action static void Adapter.init(string aocMode)
    where {.
          Condition.In(aocMode, "EAM", "OCM", "NM", "
             SAM");
    };

  action static void Adapter.fail_TempOverHL(string
      rwID)
    where {.
      Condition.In(rwID, "RW1", "RW2", "RW3", "RW4");
    };

  action static void Adapter.fail_SpeedOverHL(string
      rwID, string prepare)
         where {.
      Condition.In(rwID, "RW1", "RW2", "RW3", "RW4");
      Condition.In(prepare, "F1", "F2", "F3", "F4", "
          F5");
    };

  ...
}
```

Im Anschluss zur Definition der Parameterräume werden die benötigten Szenarien implementiert.

Die Szenarien zu den Abbildungen 24 und 25 sind in Listing 13 dargestellt. Wie zu sehen ist, wird jedem Pfad aus den beiden Abbildungen ein eigenes Szenario zugewiesen (Ausnahme DEQF: Separierung in den Regeln). Es sind alle in Tabelle 14 enthaltenen Fehlerinjizierungen möglich, gefolgt von sich jeweils daran anschließenden Statusüberprüfungen.

Listing 13: Definition der Szenarien für die Fehlerinjektion.

```
machine Fail_RWx_123(): Main
{
  (fail_TempOverHL | fail_TempBelowLL |
     fail_PowerStatus | fail_CurrentOverHL |
     fail_CurrentBelowLL); check_States
}

machine Fail_RWx_45(): Main
{
  (fail_FrictionOverHL | fail_FrictionBelowLL |
     fail_SpeedJump); check_States
}

machine Fail_RWx_6(): Main
{
  (fail_SpeedOverHL | fail_SpeedBelowLL); check_States
}

machine Fail_RW_DEQF(): Main
{
  fail_DEQF; check_States
}
```

Die Erstellung des gesamten Testmodells (bislang ohne Einschränkungen durch Regeln) erfolgt über die in Listing 14 gezeigte Implementierung. Durch die Hinzunahme von *Program* werden die zuvor festgelegten Parameterräume miteinbezogen.

Listing 14: Erstellung des gesamten Testmodells

```
machine TotalScenario() : Main
{
  init; (Fail_RWx_123|Fail_RWx_45|Fail_RWx_6|
     Fail_RW_DEQF)
}

machine TotalModel() : Main
{
  TotalScenario || Program
}
```

Da das Testmodell noch nicht exakt den Anforderungen aus der FMEA entspricht, müssen zur Vervollständigung Regeln in das von SpecExplorer

automatisch generierte *ModelProgram* eingebunden werden.

In Listing 15 ist die Regel gezeigt, mit Hilfe welcher der Fehlerfall RWx_4 im AOC-Modus OCM unterbunden wird.

Listing 15: Ausschnitt der Regeln zum Testmodell

```
static string intern_aocMode;

[Rule(Action = "init(aocMode)")]
static void init(string aocMode)
{
  intern_aocMode = aocMode;
}

[Rule(Action = "fail_FrictionOverHL(rwNum)")]
static void fail_FrictionOverHL(string rwNum)
{
  Condition.IsFalse(intern_aocMode == "OCM");
}
```

Weiterhin wird in den Regeln definiert, dass beim DEQF nicht die gleichen kombinatorischen Fälle im Modell doppelt behandelt werden (z.B. *fail_DEQF("RW1","RW4")* und *fail_DEQF("RW4","RW1")*).

Die Implementierung des Testmodells für die Stabilisierungsschwungräder ist an dieser Stelle beendet, so dass anhand des SpecExplorers das entsprechende Modell graphisch dargestellt werden kann. Dieses enthält 25 Zustände und 228 Zustandsübergänge. Auf Grund der enormen Größe (durch die vielen Zustandsübergänge), wird das generierte Modell hier nicht abgebildet.

Schritt 5: Generierung der Testfälle

Nachdem das Testmodell vervollständigt wurde, kann SpecExplorer aus dem Modell alle Testfälle generieren. Dazu wird wie zuvor beim Erdsensor die ShortTests Strategie angewendet. Durch SpecExplorers Generierung werden insgesamt 208 Testfälle erstellt, die das hierzu erstellte Testmodell abdecken und somit alle abgeleiteten Anforderungen aus der FMEA einbeziehen.

Schritt 6: Definition spezifischer SUT-Steuerungsbefehle

Zur Kommunikation zwischen den Testläufen in SpecExplorer und dem zu testenden System wird analog der Vorgehensweise zum Erdsensor-Beispiel ein Befehlsinterpreter benötigt, der zu den Steuerungsbefehlen die entsprechend generischen Methoden aus der Testbibliothek aufruft. Sowohl der Befehlsinterpreter als auch die Testbibliothek müssen eigenständig implemen-

tiert werden, so dass die notwendigen Aufrufe auf dem zu testenden System ausgeführt werden können.

Die Steuerungsbefehle entsprechen dabei den Funktionen aus dem Testmodell (Übersicht in Tabelle 14), welche mit deren Parametern angereichert sind.

Zur Übertragung der Steuerungsbefehle an den Befehlsinterpreter wird auf Seite des SpecExplorers ein Adapter hinzugefügt. Dieser sendet bei seinem Aufruf durch die Testläufe die entsprechenden Befehle über Sockets an den Befehlsinterpreter. Parallel dazu werden die Statusvariablen des Testmodells im Adapter mitgeführt, mit deren Hilfe die zurückgesendeten Systemstati des SUT, welche das Systemverhalten repräsentieren, überprüft werden können.

Schritt 7: Ausführen der Testfälle auf SUT

Die Ausführung der Testfälle erfolgt analog der Beschreibung zum Erdsensor-Beispiel unter Anwendung der hierzu erstellten Testimplementierungen (Code zu den Testfällen, Befehlsinterpreter, Testbibliothek).

Bei der Durchführung aller Testläufe hat sich herausgestellt, dass sich das System stets konform zu den Anforderungen der FMEA verhält. Die als Referenz mitgeführten Statusvariablen im Adapter waren demnach in allen 208 Testläufen identisch zu denen im SUT.

6.3.2 Erweiterung des FMEA-Modells mit SpecExplorer

Der Argumentation aus Abschnitt 6.1.2 (Erweiterung des Erdsensor FMEA-Modells) folgend wird auch das FMEA-Testmodell zu den Stabilisierungsschwungrädern erweitert. Nachdem ein Fehler in das SUT eingespeist und daraufhin autonom behoben worden ist, werden nun direkt anschließend daran weitere Fehlerinjizierungen und -behebungen möglich.

Durch diese Erweiterung wird die zuvor explizit durchgeführte Injektion des DEQF nicht mehr benötigt, da sich dieser automatisch aus aufeinanderfolgenden Fehlern ergibt. Ebenso werden jegliche kombinatorischen Fehlerfälle eingeschlossen. Dadurch liegt das Hauptaugenmerk nicht mehr nur auf der Korrektheit einzelner Funktionalitäten, sondern es wird verstärkt auch das Systemverhalten beim Zusammenspiel verschiedener Fehlersituationen überprüft.

Um dies zu erreichen müssen zunächst die Szenarien neu zusammengesetzt

werden. Die einzelnen Fehlerinjektionen und die anschließenden Überprüfungen sind durch die in Listing 16 gezeigten Szenarien definiert.

Listing 16: Szenarien für die Fehlerinjizierungen und Überprüfungen

```
machine Fail_RWx_123(): Main
{
  (fail_TempOverHL | fail_TempBelowLL |
     fail_PowerStatus | fail_CurrentOverHL |
     fail_CurrentBelowLL);
   check_States
}

machine Fail_RWx_4(): Main
{
  (fail_FrictionOverHL | fail_FrictionBelowLL);
   check_States
}

machine Fail_RWx_5(): Main
{
  fail_SpeedJump; check_States
}

machine Fail_RWx_6(): Main
{
  (fail_SpeedOverHL | fail_SpeedBelowLL);
   check_States
}
```

Die Zusammensetzung dieser Szenarien zu einem Gesamtszenario stellt sich hierbei als wesentlich komplexer dar, als es bei der Erweiterung des Erdsensors der Fall war. Die unterschiedlichen Fehlerbehandlungen und Berücksichtigungen der Statusvariablen führen zu einem wesentlich verstrickteren Modell.

Deshalb wurde dieses Modell vor der eigentlichen Implementierung zunächst graphisch auf Papier modelliert, um alle möglichen Fehlerfolgen leichter identifizieren zu können. Anschließend wurde das Modell in die SpecExplorer Szenarien überführt, so dass aus diesen schließlich das Gesamtmodell erstellt werden konnte (Erweiterung der Szenarien: Listing 17).

Listing 17: Szenarien zur Erstellung des erweiterten Modells

```
machine FirstFailScenario(): Main
{
  Fail_RWx_123;
  Fail_RWx_6?;
  (Fail_RWx_123|Fail_RWx_4|Fail_RWx_5)
}

machine SecondFailScenario(): Main
{
  Fail_RWx_4;
  (((Fail_RWx_123|Fail_RWx_6); Fail_RWx_6?; (
     Fail_RWx_123|Fail_RWx_5)) |
   (Fail_RWx_123|Fail_RWx_5))
}

machine ThirdFailScenario(): Main
{
  Fail_RWx_5;
  (((Fail_RWx_123|Fail_RWx_6); Fail_RWx_6?; (
     Fail_RWx_123|Fail_RWx_4|Fail_RWx_5)) |
   (Fail_RWx_123|Fail_RWx_4|Fail_RWx_5))
}

machine TotalScenario() : Main
{
  init;
  (Fail_RWx_6?; (FirstFailScenario |
     SecondFailScenario | ThirdFailScenario);
     check_States)*
}

machine TotalModel() : Main
{
  TotalScenario || Program
}
```

Das erstellte Gesamtmodell verfügt insgesamt über 389 Zustände und 1700 Zustandsübergänge. Daraus ließen sich (mittels der ShortTests Strategie von SpecExplorer) 1312 Testfälle generieren.

Nach den entsprechenden, mit relativ geringem Aufwand verbundenen Anpassungen in den Regeln, am Adapter, dem Befehlsinterpreter und der Testbibliothek, konnten alle 1312 Testläufe erfolgreich am SUT ausgeführt werden. Aus diesem Ergebnis ist ersichtlich, dass das getestete SUT das gewünschte, aus FMEA und wenigem zusätzlichen Systemwissen hergeleitete Verhalten aufweist, so wie es im Testmodell vorgegeben worden ist.

Die Generierung, Verknüpfung und Ausführung der Testläufe zeigt, dass die Realisierung dieses zweiten Beispiels unter Verwendung der FMEA/MBT-Methode ebenfalls erfolgreich war. Dies bedeutet abschließend, dass auch die Implementierung der FDIR-Funktionalität für die Stabilisierungsschwungräder der Galileo Satelliten erfolgreich getestet und deren Korrektheit bezüglich der aus der FMEA abgeleiteten Anforderungen nachgewiesen wurde.

7 Zusammenfassung

In dieser Arbeit wurde am Beispiel des Galileo Satellitennavigationssystems gezeigt, dass basierend auf verfügbaren Ergebnissen der FMEA und mittels des modellbasierten Testansatzes (MBT) automatisiert Testfälle und Testcode generiert werden können. Die dabei gewonnenen Erkenntnisse und Ergebnisse werden in Abschnitt 7.1 aufgeführt und anschließend in Abschnitt 7.2 bewertet. Dort wird zudem auf die Praxistauglichkeit des verwendeten FMEA/MBT-Ansatzes eingegangen. Zum Abschluss dieser Arbeit erfolgt ein Ausblick über denkbare Möglichkeiten zur Weiterentwicklung der erarbeiten FMEA/MBT-Methode.

Im Folgenden wird der Gesamtablauf der Studie anhand der einzelnen bearbeiteten Themenbereiche kurz zusammengefasst:

Sichtung gängiger Testansätze

Zu Beginn der Arbeit wurden in Kapitel 2 die gängigsten, aus der Literatur bekannten, Ansätze zum Testen von IT-Systemen präsentiert. Nach Erläuterung des manuellen Testansatzes, wurde auf unterschiedliche Arten von automatisierten Testansätzen eingegangen.

Anhand einer Gegenüberstellung wurde dargestellt, wie sich der modellbasierte Testansatz von anderen (manuellen und/oder automatisierten) Ansätzen unterscheidet. In Abschnitt 2.4 wurden dazu einige relevante Merkmale zur Übersicht gegeben. Aus dem direkten Vergleich ergab sich, dass der MBT-Ansatz am besten geeignet ist, um die Ziele dieser Arbeit zu erreichen.

Zudem wurden in Kapitel 2 gebräuchliche Notationen vorgestellt, mittels derer die Modellierung im MBT erfolgen kann. Dabei wurde insbesondere auf die zustands- und transitionsbasierte Notation eingegangen, da diese beiden Arten in der Literatur am häufigsten bei Verwendung des MBT eingesetzt werden.

Beschreibung der Fehleranalyse mittels FMEA/FMECA/FTA

Bevor in Kapitel 3 die Spezifizierung der FMEA/MBT-Methode im Detail erfolgte, wurden zunächst die Fehleranalysemethoden FMEA, FMECA und FTA allgemein erläutert. Dabei wurde insbesondere auf das Prinzip, die Herangehensweise und den Zweck der FMEA eingegangen (s. Abschnitt 3.1), da diese ein wesentliches Hilfsmittel zur Durchführung dieser Arbeit darstellt.

Entwicklung der FMEA/MBT-Methode

Da die in dieser Studie zu entwickelnde FMEA/MBT-Methode auf das Projekt Galileo angewendet werden sollte, wurde vorab zur Abgrenzung auf die ursprünglich zum Test der Galileo Satelliten angewandte Vorgehensweise eingegangen. Im Zuge dieser Beschreibung wurde hervorgehoben, wo die Schwachstellen der zuvor angewandten Methode liegen. Die festgestellten Nachteile des Testprozesses (z.B. großer zeitlicher Aufwand, manuelle Implementierung ähnlicher Testfälle) galt es grundsätzlich zu optimieren (vgl. Abschnitt 3.3).

Basierend auf diesen Überlegungen und den bereits vorgestellten Prinzipien des MBT und der FMEA erfolgte anschließend die Entwicklung einer neuen Herangehensweise. Diese integriert die Nutzung der FMEA-Ergebnisse sowie die Vorgehensweise des modellbasierten Testansatzes. Die konzipierte Testmethode wurde als FMEA/MBT-Methode vorgestellt (vgl. Abschnitt 3.4). Mit ihr wurde ein grundsätzliches Verfahren entworfen und im späteren Verlauf auf das Projekt Galileo angewendet und verfeinert.

Auswahl eines MBT-Werkzeuges

Zur späteren Umsetzung der FMEA/MBT-Methode musste als Vorarbeit ein passendes MBT-Tool gefunden werden. Dazu wurden in Kapitel 4 unterschiedliche MBT-Tools vorgestellt und verglichen. Aus diesen Tools wurden die Werkzeuge SpecExplorer (vgl. Abschnitt 4.3.1) und Fokus!MBT (vgl. Abschnitt 4.3.4) zur weiteren Verwendung ausgewählt.

Beide Werkzeuge unterstützen das modellbasierte Testen - SpecExplorer durch textuelle und Fokus!MBT anhand einer graphischen Modellierung. Durch Betrachtung der von diesen Werkzeugen unterstützten, unterschiedlichen Umsetzungsmöglichkeiten des modellbasierten Testansatzes, war es möglich verschiedene Herangehensweisen zur Modellierung kennenzulernen und zu einer objektiven Beurteilung der Eignung der FMEA/MBT-Methode zu gelangen.

Integration der FMEA/MBT-Methode in Galileo

Für den praktischen Teil der Arbeit wurde zu Beginn die Integration des FMEA/MBT-Verfahrens in das Projekt Galileo erläutert (vgl. Kapitel 5). Nach Einführung in die Architektur der Soft- und Hardware der Galileo Satelliten, wurde das in den Galileo Satelliten verwendete FDIR-Konzept vorgestellt. Auf Basis dieser Informationen wurde im Anschluss die FMEA der Galileo Satelliten präsentiert und deren Bezug zum FDIR-Konzept der Satelliten erläutert. Dabei wurden diejenigen Informationen spezifiziert, welche

von Relevanz für die Anwendung der FMEA/MBT-Methode sind. Als weiterer Schwerpunkt wurde zur Integration der Methode die Anbindung des MBT-Tools SpecExplorer an die Galileo Testumgebung beschrieben (vgl. Abschnitte 5.5 und 5.6).

Beispielhafte Anwendung in Galileo

In Kapitel 6 wurde die praktische Anwendung der FMEA/MBT-Methode unter Verwendung der MBT-Werkzeuge SpecExplorer und Fokus!MBT demonstriert. Dazu wurde die Vorgehensweise exemplarisch an zwei Satellitenteilsystemen, nämlich dem Erdsensor (SpecExplorer und Fokus!MBT) und den Stabilisierungsschwungrädern (SpecExplorer) durchgeführt. Dabei konnte zunächst jeweils nachgewiesen werden, dass die automatisierte Testfallgenerierung anhand der FMEA/MBT-Methode und unter Beschränkung auf Einzelfehlerbetrachtungen aus der FMEA durchführbar ist.

Im weiteren Verlauf wurden die erstellten FMEA-Testmodelle erweitert. Zuvor spezifizierten diese ausschließlich das Systemverhalten für einzelne Fehlerursachen. Anhand der Erweiterung wurde mögliches Systemverhalten in Bezug auf das Zusammenspiel von Fehlerzuständen während der gesamten Betriebslaufzeit mit eingeschlossen. Auf diese Weise treten die Fehlerzustände verknüpft und in beliebiger Reihenfolge auf, so dass beim Testen jegliche Fehlerkombinationen eingeschlossen sind und beispielsweise Doppelfehler auf natürliche Weise entstehen. Dadurch bildet das Testmodell das tatsächliche Systemverhalten besser ab als bei strikter Beschränkung auf die Einzelfehlerbetrachtungen der FMEA. Die vorgenommenen Erweiterungen in diesen Testmodellen resultieren aus tiefer gehenden Ableitungen zu den Ergebnissen der FMEA sowie einigen Systemkenntnissen.

Die aus der Anwendung der FMEA/MBT-Methode resultierten Ergebnisse werden im folgenden Abschnitt näher beschrieben.

7.1 Ergebnisse

Die Anwendung der entwickelten FMEA/MBT-Methode wurde an der Galileo Satellitensoftware durchgeführt. Am ersten Beispiel (Erdsensor) wurde mit Unterstützung der beiden MBT-Werkzeuge SpecExplorer und Fokus!MBT gearbeitet. Beim zweiten Beispiel (Stabilisierungsschwungräder) wurde das Verfahren nur mit SpecExplorer umgesetzt.

7.1.1 Ergebnisse: Modellierung und Testfallgenerierung

Zur praktischen Anwendung der Methode wurden zu beiden Satellitenteilsystemen (vgl. Kapitel 6) jeweils Testmodelle erstellt, welche den Systeman-

forderungen aus der Fehlerauswirkungsanalyse (FMEA) folgen. Die modellierten Testspezifikationen konnten bei beiden MBT-Werkzeugen jeweils in Form eines einzelnen, integrierten Testmodells dargestellt werden.

Zur Übersicht sind in Tabelle 15 (je Testmodell und -fälle) die unterschiedlichen Anzahlen an Zuständen und Zustandsübergängen angegeben.

SpecExplorer		
Erdsensor	**FMEA**	**FMEA (erweitert)**
Testmodell: Zustände	49	61
Testmodell: Zustandsübergänge	82	174
Testfälle	46	114
Testfälle: Zustände	219	867
Testfälle: Zustandsübergänge	173	753
Stabilisierungsschwungräder	**FMEA**	**FMEA (erweitert)**
Testmodell: Zustände	25	389
Testmodell: Zustandsübergänge	228	1700
Testfälle	208	1312
Testfälle: Zustände	832	11200
Testfälle: Zustandsübergänge	624	9888
Fokus!MBT		
Erdsensor	**FMEA**	**FMEA (erweitert)**
Testmodell: Zustände	-	49
Testmodell: Zustandsübergänge	-	165
Testfälle	-	117
Testfälle: Zustände	-	879
Testfälle: Zustandsübergänge	-	762

Tabelle 15: Übersicht zu den erstellten Testmodellen und daraus generierten Testfällen

Grundsätzlich gilt, dass bei gleichen Systemanforderungen je nach Herangehensweise unterschiedliche Modelle gebildet werden können. Dabei hängt die Qualität des Modells mit den Fähigkeiten und Erfahrungen des Testers zusammen. Neben der funktionalen Korrektheit des Modells bezieht sich dessen Qualität dabei auch auf die Übersichtlichkeit und den später zu investierenden Wartungsaufwand. Die Übersichtlichkeit betrifft sowohl die Bezeichnungen als auch die Minimierung des Testmodells auf die notwendigen Zustände und Zustandsübergänge, so dass dieses leicht verständlich bleibt und die Einarbeitungszeit für den Anwender minimal ist. Zudem sollten die abzubildenden Systemanforderungen möglichst sinnvoll in das Testmodell integriert werden.

Wie hierbei zu sehen ist, ergeben sich in den jeweiligen Testmodellen unterschiedliche Verhältnisse zwischen den Anzahlen von Zuständen und Zustandsübergängen. Innerhalb der nicht-erweiterten FMEA-Testmodelle liegt das Verhältnis beim Erdsensor bei ca. 1:2, wohingegen sich bei den Stabilisierungsschwungrädern ein deutlich anderes Verhältnis von ca. 1:9 ergab. Dies resultiert daher, dass bei den Zuständen im Modell für die Stabilisierungsräder vermehrt mit Parametern gearbeitet wurde, so dass verhältnismäßig weniger Testmodellzustände entstanden, dafür aber mehr Zustandsübergänge.

Während der Bearbeitung der Anwendungsbeispiele wurden unterschiedliche Vorgehensweisen zur Integration der Systemanforderungen in das Modell erprobt. Dies führte zu der Erkenntnis, dass eine geringere Zustandsanzahl zur Übersichtlichkeit eines Modells beiträgt. Sollten dabei jedoch große Ansammlungen von Zustandsübergängen zwischen zwei Zuständen entstehen, so mindert dies wiederum die Übersichtlichkeit des entstehenden Modells (vgl. das erweiterte FMEA-Modell für die Stabilisierungsschwungräder). Es gilt also, soweit möglich, bei der Modellierung eine gute Balance zwischen der Anzahl an Zuständen und Zustandsübergängen zu finden.

Aufwand

Vor dem sinnvollen Einsatz der MBT-Werkzeuge SpecExplorer und Fokus!MBT war es nötig sich in das jeweilige Tool einzuarbeiten. Dazu wurde nachvollzogen, wie die Modellierung vorzunehmen ist und welche Möglichkeiten die Werkzeuge dazu anbieten.

Bei der Bearbeitung des ersten Beispiels (Erdsensor) ergaben sich zunächst unterschiedliche Probleme bei der Modellierung. Dazu gehörte u.a. die Fragestellung nach einer sinnvollen Möglichkeit zur Modellierung bestimmter Testanforderungen (z.B. zeitliche Verläufe) oder die Frage nach einer geeigneten Strukturierung in Teilszenarien. Auf Grund dessen war bei diesem ersten Beispiel die Einarbeitung noch nicht abgeschlossen. Daher ist für dieses Beispiel schwer abzugrenzen, wie lange die tatsächliche Erstellungszeit des Modells dauerte. Die Gesamtdauer der Einarbeitung und Umsetzung des ersten Beispiels hat hierbei rund einen Monat in Anspruch genommen.

Die Durchführung des zweiten Beispiels (Stabilisierungsschwungräder) fand im Gegensatz dazu bereits unter Nutzung der Erfahrungen aus dem Erdsensorbeispiel statt, d.h. ohne weiteren Einarbeitungsbedarf. In diesem Fall konnte die Testmodellierung, -generierung und Implementierung der Anbindung (Adapter, Befehlsinterpreter, Testbibliothek) innerhalb einer Woche realisiert werden. Zum Vergleich: Bei der ursprünglich im Galileo Projekt

angewandten Testvorgehensweise (vgl. Abschnitt 3.3) wurde zur Implementierung der Tests zu den Stabilisierungsschwungrädern rund dreimal so viel Zeit benötigt.

Hierbei ist jedoch zu berücksichtigen, dass die Implementierung der Testbibliothek für den FMEA/MBT-Ansatz dadurch vereinfacht wurde, dass aus den bereits bestehenden (vormals manuell erstellten) Tests die notwendigen Teilimplementierungen (z.B. um die Satellitensimulation in den korrekten Initialzustand zu versetzen) übernommen werden konnten. Daher ist davon auszugehen, dass für diesen Aspekt in einem neuen Projekt zusätzlicher Aufwand zu investieren wäre, aber dennoch etwa die Hälfte der Zeit eingespart werden könnte.

Anzahl der Tests

Abgesehen von der Zeiteinsparung, werden mit Hilfe des FMEA/MBT-Ansatzes auch deutlich mehr Testfälle generiert als bei der ursprünglichen manuellen Testimplementierung. Ein Vergleich zwischen diesen ursprünglichen Tests und den FMEA/MBT-Tests kann auf Grund der gleichartigen zugrundeliegenden Spezifikationen vorgenommen werden. Die ursprünglichen Tests basieren dabei vollständig auf der FDIR-Spezifikation (s. Abschnitt 5.2). Die neuen FMEA/MBT-Tests folgen dagegen den abgeleiteten Spezifikationen aus der FMEA, welche wiederum in direktem Zusammenhang zum FDIR stehen (vgl. Abschnitt 5.4.3). Auf Grund dessen werden die jeweils resultierenden Anzahlen an Testfälle in Tabelle 16 gegenübergestellt.

	Vorher	**SpecExplorer**	**Fokus!MBT**
Erdsensor	13	46 (114)	- (117)
Stabilisierungsschwungräder	17	208 (1312)	-

Tabelle 16: Anzahl der Tests (die geklammerten Werte sind die Anzahlen der Testfälle auf Basis der erweiterten Anforderungen aus der FMEA)

In der Tabelle 16 sind in Spalte *Vorher* die Anzahlen der ursprünglichen Tests angegeben, sowie in den Spalten *SpecExplorer* und *Fokus!MBT* die Anzahlen der FMEA/MBT-Tests. Die Steigerung der Anzahlen zwischen den vorherigen und den generierten Tests sind hierbei deutlich zu erkennen. Für das Beispiel der Stabilisierungsschwungräder konnten im Vergleich zu den vorherigen 17 Tests mittels der FMEA/MBT-Methode 1312 Tests generiert werden (was einer Steigerung um das 77-fache entspricht).

Der Erwartung entsprechend wurden zum erweiterten Erdsensor Beispiel

unabhängig von den benutzten MBT-Werkzeugen (SpecExplorer und Fokus!MBT) fast identische Testfälle generiert. Das Heranziehen von identischen Anforderungen für das Systemverhalten führt (trotz unterschiedlichem Aufbau) zu einer inhaltlichen Übereinstimmung zwischen den beiden Modellen. Daher ergeben sich aus den beiden Modellen etwa die gleichen Anzahlen an Testfälle (vgl. 114 bzw. 117 Testfälle).

Der Vergleich der für die beiden Teilsysteme resultierenden Testfälle verdeutlicht den Größenunterschied der zugrundeliegenden Modelle. Dieser ergibt sich insbesondere durch das Vorhandensein von vier physikalischen Einheiten bei den Stabilisierungsschwungrädern, wohingegen der Erdsensor nur zwei physikalische Einheiten aufweist.

Laufzeiten

Die Laufzeiten der originalen Testfälle unterscheiden sich beträchtlich von denen der generierten Tests. So werden im Falle des Erdsensors zur Ausführung der 13 originalen Tests nur 8 Minuten benötigt. Dagegen erfordern die generierten (wesentlich zahlreicheren) Tests zum Erdsensor die ca. fünf- bis zehnfache Laufzeit.

Die Laufzeit der 1312 generierten Testläufe des erweiterten Stabilisierungsschwungräder Modells ist im Vergleich noch erheblich größer. Diese benötigen (auf dem gleichen Rechner) über einen Tag, wohingegen die 17 originalen Tests nach rund 20 Minuten bereits ausgeführt sind.

Hierbei ergeben sich die beträchtlich längeren Laufzeiten insbesondere durch die automatisierte Generierung von Tests zu Fehlerkombinationen. Darin spiegelt sich also wieder, dass sowohl alle physikalischen (redundanten) Einheiten berücksichtigt als auch alle kombinatorischen Möglichkeiten (z.B. unterschiedliche Fehlerreihenfolgen) ausgereizt werden.

7.1.2 Das Gesamtergebnis

Durch die Bearbeitung der zu Beginn des Kapitels zusammengefassten Themenbereiche ließen sich die in Abschnitt 1.4 genannten Aufgaben zu dieser Studie umsetzen. Als Hauptziel wurde dabei nachgewiesen, dass ausgehend von der FMEA automatisiert Testfälle/-code anhand des modellbasierten Testansatzes generiert und ausgeführt werden können.

Darüber hinaus hat sich gezeigt, dass es mit relativ geringem Aufwand möglich ist, die zunächst strikt auf der FMEA basierende Modellierung zu erweitern, um dadurch ein umfangreicheres Spektrum an Tests zu erzielen.

Im Vergleich zum bisher eingesetzten Testansatz erlaubt die hier vorgestellte Methode die Generierung einer erheblich größeren Anzahl an Tests bei gleichzeitig geringerem zeitlichen Aufwand. Zu berücksichtigen ist hierbei allerdings, dass für die Anwendung der Methode ein nicht zu unterschätzender Einarbeitungsaufwand anfällt.

Ein weiterer Vorteil der Methode ist, dass sie die Durchführung von Wartungsarbeiten begünstigt. Bei Modifizierungen am SUT müssen zur Aktualisierung der entsprechenden Tests lediglich die entsprechenden Modellteile oder die betroffenen Stellen der Testfunktionalitäten der Testbibliothek angepasst werden. Da es sich um eine abstrakte und generische Testmethode handelt, ist der dafür zu investierende Aufwand häufig wesentlich geringer als die direkte Überarbeitung von manuell erstellten Testskripten.

Insgesamt wurde hiermit ein Testverfahren vorgestellt, welches sich (unter Berücksichtigung der im nachfolgenden Abschnitt erläuterten Gesichtspunkte) als effizient herausgestellt hat.

7.2 Kritische Bewertung

Wie zuvor beschrieben, wurde in dieser Arbeit mit der FMEA/MBT-Methode ein effizientes Testverfahren präsentiert. Im Folgenden werden dessen Stärken und Schwächen im Detail diskutiert, um letztlich die Praxistauglichkeit des Verfahrens bewerten zu können. Dabei wird auf folgende Faktoren eingegangen:

1. Ausführlichkeit der generierten Tests
2. Reines FMEA-Modell vs. erweitertes FMEA-Modell
3. Qualität des Testsystems
4. Faktor Zeit
5. Fähigkeiten des Testers
6. Benötigte Mittel
7. Management und Dokumentation der Tests
8. Anwendbarkeit in der Praxis

Ausführlichkeit der generierten Tests

Bei der Anwendung des vorgestellten Testverfahrens auf die beiden Teilsysteme der Galileo Satelliten konnte gezeigt werden, dass eine große Anzahl von Tests automatisiert erzielt werden kann. Um dieselbe Anzahl an Tests manuell zu implementieren, müsste ein immenser Aufwand betrieben werden. Zur Verdeutlichung: Zu den Stabilisierungsschwungrädern wurden 17 Tests implementiert, wohingegen 1312 Tests automatisiert bei geringerem (halber/drittel) Aufwand mittels FMEA/MBT generiert werden konnten.

Die zahlreicheren Tests der FMEA/MBT-Methode umfassen zudem einen inhaltlich größeren Bereich. Durch die effizientere Testerstellung wird es möglich auch redundant erscheinende Tests, wie beispielsweise das Testen von Fehlerbehandlungen an jeder einzelnen der identischen physikalischen Einheiten, durchzuführen. Zuvor galt aufgrund der gegebenen zeitlichen Rahmenbedingungen oftmals das Testen an einer (als repräsentativ betrachteten) physikalischen Einheit als ausreichend. Anhand solcher exemplarischen Tests wurde zusammen mit der Annahme einer gleichartigen bzw. identischen Implementierung gefolgert, dass die jeweilige Funktionalität insgesamt korrekt arbeitet. Da allerdings das Prinzip des Black-Box-Testens[4] angewendet wird, ist das explizite Testen aller Fälle die sicherste Vorgehensweise, um das System tatsächlich vollständig und ohne Hilfsannahmen validieren zu können.

Insgesamt kann die FMEA/MBT-Methode mittels der (erweiterten) Anforderungen aus der FMEA jegliches geforderte Systemverhalten überprüfen, welches als Reaktion auf potentielle Fehlerzustände erwartet wird. Diese Einschränkung des Verfahrens auf die FMEA verringert jedoch die Bandbreite der generierten Tests. Wie in Abschnitt 5.4.3 erläutert, werden mit dieser Methode so nur etwa 30% aller Tests erreicht, welche zur vollständigen Validierung der On-Board Software der Galileo Satelliten notwendig wären.

Dies bedeutet, dass das FMEA/MBT-Verfahren zunächst nicht als grundsätzliches Testvorgehen zum Test des gesamten Galileo Satellitensystems betrachtet werden kann. Allerdings muss die entwickelte Methode nicht zwangsweise auf Basis der FMEA durchgeführt werden. Jegliche Anforderungen könnten herangezogen werden, so dass mit dieser Methode auch die verbliebenen 70% Tests umgesetzt werden könnten.

[4] Beim Black-Box-Testen wird ein System auf Basis von Anforderungen ohne Einsicht in die Implementierung getestet.

Reines FMEA-Modell vs. erweitertes FMEA-Modell

Wie bereits erwähnt wurde, deckt das reine FMEA-Testmodell im Gegensatz zum erweiterten FMEA-Modell ausschließlich die direkten Systemanforderungen aus der FMEA ab. Dadurch wird nicht das tatsächliche Systemverhalten im operativen Betrieb abgebildet, sondern das Testen wird auf die (isolierte) Betrachtung von Einzelfunktionalitäten reduziert. Auf diese Weise werden alle geforderten Funktionalitäten ohne weitere Umwege überprüft. Die resultierende Testmenge bleibt dabei überschaubar.

Im Vergleich dazu können mit Hilfe des erweiterten FMEA-Modells jegliche Fehlersituationen des Systems in einer Art und Weise getestet werden, wie diese im Laufe der Betriebszeit auch in der Realität eintreten können. Mit dem Vorteil einer erhöhten Ausführlichkeit des Tests kann so das gesamte Verhaltensspektrum des SUT in Bezug auf die FMEA-Anforderungen überprüft werden.

Das Erstellen der erweiterten Tests verursacht im Vergleich zu den rein FMEA-basierten Tests einen kaum größeren Aufwand. Dadurch wird es möglich insbesondere auch Kombinationen von Fehlerzuständen einzubeziehen, welche entsprechend dem tatsächlichen Systemverhalten auftreten können.

Qualität des Testsystems

Die Qualität eines Testsystems zeichnet sich durch seine Wartbarkeit aus. Das auf Basis der FMEA/MBT-Methode realisierte Testsystem erweist sich in Hinblick auf diesen Aspekt als besonders effizient. Im Falle von Modifizierungen am SUT sind nur kleinere Anpassungen am Modell oder den Implementierungen von Testbibliothek, Befehlsinterpreter und Adapter zu erwarten. Nach Abschluss dieser Modifizierungen sind die generierten Testfälle per Knopfdruck aktualisiert.

Das abstrakte Verfahren und die möglichst generische Modellierung und Implementierung erlauben die zügige Anpassung sowohl an Modifikationen des SUT als auch an zusätzliche, völlig neu im SUT implementierte Anforderungen.

Faktor Zeit

Bevor mit der FMEA/MBT-Herangehensweise gearbeitet werden kann, muss der ausführende Testingenieur das Verfahren beherrschen. Dazu wird als Kernkompetenz die Befähigung zum Erstellen von Modellen benötigt. In diesem Zusammenhang hat sich bei der Entwicklung des zweiten Anwendungsbeispiels (Stabilisierungsschwungräder) herausgestellt, dass die Erstel-

lung des Testsystems auf Grund der bereits im Anwendungsbeispiel für den Erdsensor erfolgten Einarbeitung wesentlich schneller durchgeführt werden konnte. Die Modellierung der Testanforderungen sowie die weiteren Implementierungen benötigten insgesamt nur eine Woche, um schließlich die daraus generierten 1312 Tests ausführen zu können. Zum Vergleich: Mit dem ursprünglich verwendeten Testansatz wurde die zwei- bis dreifache Zeit benötigt, um nur die wichtigsten Testfälle zum selben Teilsystem implementieren zu können.

Im Falle der erweiterten Testmodelle entstehen viele gleichartige (redundante) Testfälle. Bei Ausschöpfen dieser enormen Testmenge ist Vorsicht angebracht, da der dabei zu betrachtende Zustandsraum regelrecht explodieren kann. Daher ist während der Modellierung zu hinterfragen, an welchen Stellen tatsächlich alle resultierenden Testfälle notwendig und sinnvoll sind.

Das Entstehen solch einer immensen Anzahl von Tests führt insbesondere auch bei deren Ausführung zu Problemen, da die verfügbare Rechenleistung in der Regel begrenzt ist. Es sollte also berücksichtigt werden, ob im Verlauf des Testprozesses der zu investierende Ausführungsaufwand betrieben werden kann oder ob die Tests in ihrer Anzahl beschränkt werden müssen.

Fähigkeiten des Testers

Wichtiger Bestandteil bei Umsetzung dieser Methode ist die Fähigkeit zur Modellierung und Implementierung. Entsprechend den Fähigkeiten des Testingenieurs, fällt die Qualität des entstehenden Testsystems aus. Zur Maximierung der Qualität sollte möglichst viel Erfahrung in der Modellierung und Implementierung gesammelt werden. Zusätzlich werden tiefgehende Kenntnisse über das SUT und die verfügbare Testumgebung benötigt. Ohne diese ist der Aufbau der benötigten Testbibliothek nicht möglich. Weiterhin sind gute Systemkenntnisse auch nötig, um mit dem entsprechenden Hintergrundwissen erweiterte Testfälle bzw. Testmodelle aus der FMEA ableiten zu können.

Benötigte Mittel

Zum Extrahieren der Systemanforderungen werden zur Anwendung der Testmethode die Ergebnisse einer FMEA verlangt. Die Erstellung der FMEA ist aufwendig und muss zum Zwecke der Aktualität bei der Entwicklung stets gewartet werden. Im optimalen Fall ist die Anwendung der FMEA für das Projekt bereits eingeplant.

Des Weiteren ist die Verfügbarkeit eines Werkzeugs für die Modellierung von großer Relevanz. Bei Benutzung von kommerziellen MBT-Werkzeugen kann

der Erwerb der Lizenzen sehr teuer ausfallen. Bevor darin investiert wird, kann die Tauglichkeit allerdings oftmals anhand von Testlizenzen überprüft werden. Alternativ kann auf kostenlose MBT-Tools zurückgegriffen werden, welche für viele Einsatzzwecke bereits ausreichenden Funktionsumfang zur Verfügung stellen (vgl. SpecExplorer oder NModel).

Wichtig bei der Werkzeugwahl ist es zu beachten, dass die bevorzugte Art der Modellierung unterstützt wird. Im Falle des SpecExplorers wurden die Testszenarien textuell implementiert, wohingegen bei Fokus!MBT die Modellierung graphisch erfolgt. Grundsätzlich besteht eine Vielzahl an Modellierungsarten. Da die Umsetzung von den subjektiven Präferenzen abhängt und jeweils gut zu beherrschen sein sollte, muss die Wahl jeweils individuell getroffen werden.

Während dieser Arbeit hat sich gezeigt, dass bei der Modellierung mittels SpecExplorer intuitiv vorgegangen werden kann und die Erstellung von Testmodellen relativ einfach zu bewerkstelligen ist. Insbesondere nach erfolgter Einarbeitung und etwas Übung kann mit diesem Werkzeug effizient gearbeitet werden. Weiterhin spricht der kostenfreie Zugang für dieses MBT-Tool, sowie die zahlreichen Beiträge im Internet mit Bezug zur Nutzung des SpecExplorers.

Bei Anwendung von Fokus!MBT für die Modellierung zeigte sich, dass die von diesem Werkzeug unterstützte graphische Modellierung die Erstellung des Modells wesentlich überschaubarer gestaltet. Dennoch ist zu beachten, dass trotz graphischer Modellierung auch hier die Regeln und der Adapter textuell implementiert werden müssen. Da die zu bewältigende Hauptaufgabe jedoch in der Testmodellierung liegt, ist die Verwendung eines graphischen Tools durchaus sehr vorteilhaft. Durch den im Vergleich zu SpecExplorer weitaus größeren Funktionsumfang fällt der Einarbeitungsaufwand bei diesem Werkzeug etwas höher aus, erlaubt aber nach einer gewissen Übungsphase und durch den graphischen Modellierungsansatz ebenfalls eine effiziente Nutzung. Bei der Verwendung eines nicht direkt frei verfügbaren Tools wie diesem ist grundsätzlich zu berücksichtigen, ob der Erwerb der Lizenz für das jeweilige durchzuführende Projekt zwischen dem Tool-Anbieter und dem Anwender erfolgreich abgestimmt werden kann.

Management und Dokumentation der Tests

In Abschnitt 4.2 wurden die Anforderungen an MBT-Werkzeuge im Hinblick auf die Erstellung der Testdokumentation erläutert. Während der Bearbeitung der Anwendungsbeispiele zur FMEA/MBT-Methode (vgl. Kapitel 6) wurde dieser Aspekt des Testens nicht weiter beleuchtet. Deshalb kann an dieser Stelle keine tiefgehende und abschließende Betrachtung der dazu von

den MBT-Werkzeugen gebotenen Unterstützung erfolgen.

Dennoch war in diesem Zusammenhang zu erkennen, dass zum Zweck der Beschreibung der Testfälle (d.h. Erstellung eines Testplans oder einer Testspezifikation) gewisse Werkzeugfunktionalitäten zur Verfügung stehen. So können das gesamte Testmodell bzw. (zur Förderung der Übersichtlichkeit) repräsentative Auszüge davon Auskunft über die entstehenden Testfälle geben. Ebenso können die tatsächlich generierten Testfälle (s. Beispiel in Abbildung 13) zum Überblick über die generierten Testinhalte beitragen. In beiden Fällen handelt es sich um graphische Visualisierungen von Testinhalt bzw. Testablauf. Für eine textuelle Beschreibung dieser Testinhalte und Testabläufe (vgl. Abschnitt 4.2) könnten entsprechende Texte zu den jeweiligen Modellkomponenten hinzugefügt werden. Anhand dieser ließe sich parallel zu den Testläufen auch eine zusammenfassende, textuelle Dokumentationen des Testablaufs generieren. Dieser Ansatz wird aktuell von den Herstellern des Fokus!MBT verfolgt und umgesetzt. Bei SpecExplorer konnten keine entsprechenden Ansätze zur Unterstützung der Testdokumentation gefunden werden.

Im Hinblick auf den für die Testdokumentation ebenfalls wichtigen Nachweis der vollständigen Anforderungsüberdeckung bieten beide MBT-Werkzeuge Möglichkeiten, um Anforderungen mit entsprechenden Zustandsübergängen im Modell zu verknüpfen. Somit lässt sich bestimmen wie viele Anforderungen insgesamt und welche Anforderungen an welchen Stellen in den Testfällen abgedeckt werden. Anhand dieser Informationen wird die Nachverfolgung und Dokumentation der Anforderungsvalidierung ermöglicht.

Des Weiteren müssen die Testdurchführungen und Testergebnisse angemessen protokolliert werden. Dabei muss erkenntlich sein welche Testfälle bzw. welche Anforderungen (nicht) erfolgreich getestet wurden. Diese Funktionalität wird sowohl von SpecExplorer als auch von Fokus!MBT unterstützt. Ebenso sollen Testlogs über die jeweiligen Testverläufe Auskunft geben. Dazu könnten die jeweiligen Aktionen mit entsprechenden Informationen angereichert werden, so dass aus dem Log die ausgeführten Testschritte nachvollziehbar sind. Solche Anreicherungen könnten an die entsprechenden Stellen im Testmodell oder bei Benutzung einer Testbibliothek (s. Anwendungsbeispiel) zu den jeweiligen Testfunktionalitäten hinzugefügt werden.

Ferner ermöglicht die FMEA/MBT-Methode auf Grund der nachvollziehbaren und reproduzierbaren Testergebnisse die Durchführung von Regressionstests. Dadurch lässt sich während der Entwicklungsphase eines Systems kontinuierlich überprüfen, ob sich in bereits früher validierten Systemteilen Fehler eingeschlichen haben. Auf diese Art lässt sich das System, ausgehend von einer stabilen Basis, kontinuierlich verbessern, da anhand der Tester-

gebnisse das Aufspüren etwaiger Abweichungen vom erwarteten Testverlauf einfach zu bewerkstelligen ist.

Anwendbarkeit in der Praxis

Nachdem gezeigt werden konnte, dass die Anwendung der FMEA/MBT-Methode prinzipiell möglich und sinnvoll ist, ist nun von Interesse, ob sich diese Methode auch als praxistauglich erweist. Um dies zu beurteilen, werden die zuvor in diesem Abschnitt genannten Faktoren betrachtet.

Um festzustellen, ob die FMEA/MBT-Methode in einem bestimmten Projekt anwendbar ist, muss zunächst abgeklärt werden, ob die notwendigen Voraussetzungen gegeben sind. Dies betrifft zum einen die Verfügbarkeit aktueller Ergebnisse der FMEA und zum anderen das Vorhandensein eines entsprechenden MBT-Tools (über den gesamten Projektverlauf). Ohne diese kann das Verfahren nicht eingesetzt werden, da diese die Basis des gesamten Testvorgehens bilden.

Können diese Basisvoraussetzungen gewährleistet werden, so ist zu berücksichtigen, dass eine gewisse Einarbeitungsphase notwendig ist, damit das Verfahren von jeweiligen Testingenieuren umgesetzt werden kann. Der dafür zu investierende Aufwand ist nicht zu unterschätzen, bleibt aber insgesamt überschaubar.

Um frühzeitig bereits während der Entwicklungstätigkeit Systemfehler identifizieren zu können, ist die parallele Durchführung von Test- und Systementwicklung grundsätzlich sinnvoll. Die FMEA/MBT-Methode unterstützt diese Vorgehensweise, zumal die FMEA normalerweise bereits in frühen Entwicklungsphasen zur Verfügung steht. Da zudem bei diesem Testansatz auch die Wartung der entstehenden Tests bzw. Testmodelle gut handhabbar ist (siehe oben), ist der frühzeitige Einsatz des Verfahrens problemlos möglich.

Allerdings kann sich die Schwierigkeit ergeben, dass beim erweiterten FMEA-Testmodell der resultierende Zustandsraum förmlich explodiert und somit die Ausführung der generierten Tests auf Grund begrenzter Rechenleistung nicht mehr in einem sinnvollen Zeitrahmen möglich ist. Es muss deshalb während des Projektverlaufs wiederholt überprüft und abgewägt werden, ob die Testmenge von den verfügbaren Rechnern noch bewältigt werden kann oder reduziert werden muss. Alternativ könnte auch das rein FMEA basierte Testmodell angewandt werden.

Des Weiteren ist zu beachten, dass mit dieser Methode ausschließlich die (erweiterten) abgeleiteten Anforderungen aus der FMEA getestet werden. Somit wird hierbei ausschließlich das Systemverhalten getestet, das auf po-

tentielle Fehlerzustände folgt. Hierzu ist fraglich, ob man zum Testen dieser Fälle eine gesonderte Testverfahrensweise nutzen möchte. Da allerdings das angewandte Verfahren nicht notwendigerweise auf die Nutzung der FMEA beschränkt ist, könnten die verbliebenen Tests ebenso anhand des modellbasierten Testansatzes realisiert werden. Dazu könnten die FMEA-Testmodelle als Grundlage fungieren und während des Projektfortschritts anhand beliebiger zusätzlicher Anforderungen erweitert werden.

Da die FMEA/MBT-Methode neben den zuvor genannten Punkten den Vorteil bietet ausführliche Tests bei relativ geringem Zeitaufwand zu erhalten (vgl. oben), ist das vorgestellte Testverfahren für die Praxis als grundsätzlich sinnvoll zu bewerten. Allerdings ist zu beachten, dass Herausforderungen bezüglich der Zustandsraumexplosion und der Dokumentation bestehen, welche entsprechend gehandhabt werden müssen.

In Tabelle 17 sind die zuvor beschriebenen Vor- und Nachteile der FMEA/MBT-Methode nochmals im Überblick zusammengefasst.

Vorteile	**Nachteile**
Sehr ausführliche Tests	Gefahr der Zustandsraumexplosion (riesige Testmenge führt zu immensen Laufzeiten, benötigte Rechenleistung)
Testsystem sehr gut wartbar	Dokumentationslösung erforderlich (für Testsystem und -ergebnisse)
Akzeptabler Entwicklungsaufwand	Einarbeitung notwendig
Kostenfreie MBT-Tools verfügbar	Kommerzielle Tools erfordern Lizenzen
Automatisierte Ausführung	
Regressionstesten möglich	
Methode muss nicht auf FMEA basieren	
Modelle sind durch graphische Darstellung gut einsehbar	

Tabelle 17: Auflistung der Vor- und Nachteile zur FMEA/MBT-Methode

Zum Abschluss dieser Arbeit wird im letzten Abschnitt ein Ausblick auf denkbare Erweiterungsmöglichkeiten der vorgestellten FMEA/MBT-Methode gegeben.

7.3 Ausblick

Wie bereits in vorhergehenden Abschnitten erwähnt wurde, kann sich bei Verwendung des modellbasierten Testansatzes das Problem der Zustandsraumexplosion und damit verbunden die Schwierigkeit der rechentechnischen Bewältigung aller Testläufe ergeben. Ein denkbarer Lösungsansatz dazu betrifft die Verwendung eines zufallsgestützten Verfahrens, welches je Testausführung nur einen Teil der gesamten Testmenge auswählt. Beispielsweise könnte für die in Kapitel 6 betrachteten Teilsysteme je Testsitzung jeweils nur ein repräsentativer AOC-Modus zufällig ausgewählt werden, so dass nur ein Drittel/Viertel aller potentiell möglichen Tests ausgeführt werden müsste. Mittels dieser zufälligen Wahl sollten dennoch nach einigen Testsitzungen alle AOC-Modi einmal zur Anwendung gekommen sein, so dass über einen gestreckten Zeitraum alle Testfälle durchgeführt werden. Bei dieser Herangehensweise besteht allerdings das Problem, dass die Vergleichbarkeit der Tests abnimmt, was die Durchführung von Regressionstests erschwert.

Im Hinblick auf die Dokumentation der Testergebnisse wäre weiterhin eine zusätzliche Kennzeichnung (z.B. Einfärbung) der erfolgreich durchlaufenen Zustandsübergänge des Testmodells vorstellbar. Zudem könnte ein sogenannter *Treewalker* bei der Ausführung der Testläufe die aktuell betrachtete Stelle des Testmodells kennzeichnen, um den Verlauf der Testdurchführung simultan mitverfolgen zu können.

Wie sich bei der Anwendung der FMEA/MBT-Methode auf die beiden Galileo Teilsystemen (s. Kapitel 6) herausgestellt hat, bestanden die Testmodelle aus vielen, sich ähnelnden Teilkonstrukten. Die Verwendung von redundanten Einheiten, sowie die hier konkret benötigte Fehlereinspeisung oder die diversen Initialisierungen des Systems führen zu ähnlichen Strukturen im Testmodell. Im Hinblick auf diese und weitere Parallelen könnte die Verwendung von Metamodellen innerhalb des MBT-Werkzeugs zu einer Reduzierung des Modellierungsaufwands führen.

Ein weiterer Aspekt dessen Untersuchung lohnenswert erscheint, betrifft den gemeinsamen Einsatz mehrerer Modelle. So könnten beispielsweise Teilmodelle in größere Modelle integriert werden oder generell das Zusammenspiel verschiedener Teilsysteme untersucht werden. Dazu müssten Schnittstellen entwickelt werden, um Interaktionen zwischen diesen Modellen ermöglichen zu können.

Ferner bietet die Kommunikationsschnittstelle zwischen dem MBT-Tool und der spezifischen Testumgebung Möglichkeiten für zukünftige Erweiterungen. Diese Schnittstelle wurde in dieser Arbeit mittels TCP/IP-Sockets und einem einfachen Client/Server Modell implementiert. Dieser Ansatz erlaubt

es die Testumgebung (in dieser Arbeit die Galileo Testumgebung, vgl. Abbildung 5) vom MBT-Werkzeug zu trennen und auf zentralen Rechnern zur Verfügung zu stellen. Dadurch könnten die generierten Testläufe entfernt auf leistungsfähigen Servermaschinen ausgeführt werden. In diesem Zusammenhang wäre es interessant zu untersuchen, ob und wie eine Parallelisierung der Testausführung möglich ist, um evtl. zur Milderung des Problems der Zustandsexplosion beitragen zu können.

Literatur

[Aci05] Fatma Aciksöz. U2TP, UML Testing Profile Specification. Softwaretechnik, Entwicklung verteilter eingebetteter Systeme, TU-Berlin, 2005.

[Bak08] Baker, Paul and Dai, Zhen Ru and Schieferdecker, Ina and Williams, Clay. *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2008.

[Con12] Gavin Robbins Consulting. FMEA. www.fmea.co.uk, 2012. Letzer Aufruf: Dez 2012.

[Eri99] Clifton Ericson. Fault Tree Analysis, A History, 1999.

[ESA07] ESA. *Die ersten Galileo Satelliten - Galileo In-Orbit Validation Element*. ESA Publications Division, 2007.

[ESA12] ESA. What is Galileo? http://www.esa.int/esaNA/galileo.html, 2012. Letzer Aufruf: Dez 2012.

[FOK11] Fraunhofer FOKUS. Fokus!MBT. http://www.fokus.fraunhofer.de/en/motion/ueber_motion/technologien/fokusmbt/index.html, 2011. Letzer Aufruf: Dez 2012.

[Ham10] Vít Hampl. FMEA and FTA, 2010.

[Hay04] Linda Hayes. *The Automated Testing Handbook*. Software Testing Inst, Mrz 2004.

[Hei09] Heise. Microsofts Spec Explorer 2010 hilft beim modellbasierten Testen von Software. http://www.heise.de/developer/meldung/Microsofts-Spec-Explorer-2010-hilft-beim-modellbasierten-Testen-von-Software-843574.html, Okt 2009. Letzer Aufruf: Dez 2012.

[Hoh06] Volker Hohlwegler. TTCN-3, Testing and Test Control Notation. Seminar Software-Entwicklungsmethoden, 2006.

[Hom03] Werner Hommes. Werkzeuge zur Testautomatisierung. Fachhochschule Trier, Jul 2003.

[IST07] ISTQB. Certified Tester, Advanced Level Syllabus, Kapitel 3.10. German Testing Board e.V., 2007.

[ITW12] ITWissen. Galileo. http://www.itwissen.info/definition/lexikon/Galileo-Galileo.html, 2012. Letzer Aufruf: Dez 2012.

[JVCS07] Jonathan Jacky, Margus Veanes, Colin Campell, and Wolfram Schulte. *Model-Based Software Testing and Analysis with C#*. Cambridge University Press, 2007.

[Kla05] Christiane Klapdohr. Modellbasiertes Testen. Universität Paderborn, 2005.

[Koc12] Martin Koch. Ihre herausforderungen im bereich modellbasiertes testen. http://www.iese.fraunhofer.de/de/competencies/model-based_testing/challenges_mbt.html, 2012. Letzer Aufruf: Dez 2012.

[KS07] Andreas Kohlhase and Reinhard Schlitt. *Thermalkontrolle. In: Handbuch der Raumfahrttechnik.* Hanser Verlag, 2007.

[Lei12] Bernd Leitenberger. Aufbau eines Satelliten. http://www.bernd-leitenberger.de/aufbau-der-satelliten.shtml, 2012. Letzer Aufruf: Dez 2012.

[Mic12] Microsoft. NModel. http://nmodel.codeplex.com, 2012. Letzer Aufruf: Dez 2012.

[Moi99] Adam Moik. Strukturierte Erstellung von formalen Sicherheitsmodellen für Automatisierungssysteme mit Sicherheitsverantwortung. Paper, 1999.

[MRG11] Ben Swarup Medikonda, P. Seetha Ramaiah, and Anu A. Gokhale. FMEA and Fault Tree based Software Safety Analysis of a Railroad Crossing Critical System. Andhra University, Mai 2011.

[Nef03] Dominik Neff. Automatisiertes Testen graphischer Benutzeroberflächen. http://www.bea-projects.de/files/BlueNotes/BlueNote%20-%20GUI-Tests.pdf, 2003. Letzer Aufruf: Dez 2012.

[Pra12] Gold Practice. Model Based Testing. https://goldpractice.thecsiac.com/practices/mbt/, 2012. Letzer Aufruf: Dez 2012.

[Res12] Microsoft Research. Model-based Testing with SpecExplorer. http://research.microsoft.com/en-us/projects/specexplorer, 2012. Letzer Aufruf: Dez 2012.

[Sch05] Ina Schieferdecker. Das UML Testing Profile, Erweitertes Abstrakt. Faunhofer FOKUS, Berlin, 2005.

[Sch08] Fabian Scholz. Testen von Benutzeroberflächen. Westfälische Wilhelms-Universität Münster, 2008.

[Sch10] Schmitt, Robert. Qualitaetsmanagement in den frühen phasen, fokus abweichung. Vorlesung Qualitaetsmanagement, RWTH Aachen, 2010.

[Sch11] Ina Schieferdecker. Basiswissen Softwaretest, Certified Tester. Vorlesung Grundlagen des Softwaretestens, Freie Universität Berlin, 2011.

[Seg10] Galileo IOV Space Segment. FDIR Requirements Specification Document. GAL-REQ-ASTD-SA-R-0140, Astrium GmbH, 2010.

[Seg11a] Galileo IOV Space Segment. FMECA. GAL-AN-ASTD-SS-A-0021, Astrium GmbH, 2011.

[Seg11b] Galileo IOV Space Segment. Satellite FDIR Concept. GAL-TN-ASTD-SA-Z-0033, Astrium GmbH, 2011.

[Seg11c] Galileo IOV Space Segment. Space Segment RAMS Report. GAL-RP-ASTD-SS-R-0003, Astrium GmbH, 2011.

[SMJU10] Monalisa Sarma, P.V.R. Murthy, Sylvia Jell, and Andreas Ulrich. Model-Based Testing in Industry: A Case Study with Two MBT Tools, 2010.

[Sof12a] SmartBear Software. Benefits of Keyword Testing for Automated Testing. http://support.smartbear.com/articles/testcomplete/benefits-of-keyword-testing-for-automated-testing/, 2012. Letzer Aufruf: Dez 2012.

[Sof12b] SoftwareTesting.ro. Manuelles vs. automatisiertes Testen. http://www.softwaretesting.ro/Deutsch/Files/ManualVsAutomation/Software%20Testing%20Manual%20vs%20Automated.html, 2012. Letzer Aufruf: Dez 2012.

[Sub09] Galileo IOV Avionics Subsystem. FMECA. GAL-AN-ASTD-AV-A-0032, Astrium GmbH, 2009.

[Tes12] Testmanagement. Testautomatisierung. http://www.testoptimierung.de/testautomatisierung.html, 2012. Letzer Aufruf: Dez 2012.

[UL07] Mark Utting and Bruno Legearn. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2007.

[Ver12] Verifysoft. Automated Test Design mit Conformiq Tool Suite. http://www.verifysoft.com/de_conformiq_automatic_test_generation.html, Apr 2012. Letzer Aufruf: Dez 2012.

[Ves05] Bill Vesely. Fault Tree Analysis (FTA), Concepts and Applications. NASA, 2005.

[Wen12] Marc-Florian Wendland. Fokus!MBT, The Test Modeling Environment, Technical and Methodological Insights. Fraunhofer FOKUS, Berlin, Mrz 2012.

[Zec03] Jens Zech. TTCN-3, Testing and Test Control Notation. TU-Berlin, 2003.