Abschlussbericht des Projekts SiLEST

(Software in the Loop for Embedded Software Test)

in der BMBF-Forschungsoffensive

"Software Engineering 2006"

Dr. Olaf Maibaum, Dr. Anita Herrmann, Holger Schumann, Axel Berres; Deutsches Zentrum für Luft- und Raumfahrt e.V.

Roland Serway, Sven Rebeschieß; Ingenieurgesellschaft Auto und Verkehr GmbH

Prof.-Dr. Clemens Gühmann, Dr. Thomas Liebezeit, Uzmee Bazarsuren; Technische Universität Berlin

Dr. Sergio Montenegro; Fraunhofer FIRST

Jens Geppert, Ioan Ionescu; webdynamix GmbH

Das diesem Bericht zugrunde liegende Projekt wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01ISC12A gefördert. Die Verantwortung für den Inhalt der Veröffentlichung liegt bei den Autoren.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Dokument ist der Schlussbericht des Verbundvorhabens SiLEST (Software-in-the-Loop for Embedded Software Test). Das Vorhaben wurde vom 1.3.2004 bis 30.4.2007 durch das BMBF im Rahmen der Forschungsoffensive *Software Engineering 2006* unter dem Förderkennzeichen 01ISC12A gefördert. Im Verbundprojekt arbeiteten die folgenden Kooperationspartner zusammen:

- Deutsches Zentrum für Luft- und Raumfahrt e.V., Simulations- und Softwaretechnik, Braunschweig (DLR)
- Fraunhofer Gesellschaft zur Förderung der Angewandten Forschung e.V., Institut für Rechnerarchitektur und Softwaretechnik FIRST, Berlin (FhG)
- Ingenieurgesellschaft Auto und Verkehr GmbH, Berlin (IAV)
- webdynamix GmbH, Cottbus (webdynamix)

Die IAV vergab einen Teil ihrer Umfänge an die Technische Universität Berlin, Fachgebiet Elektronische Mess- und Diagnosetechnik, Prof. Gühmann (TUB).

Der Schlussbericht beschreibt die Aufgabenstellung, die Planung, das Umfeld sowie den Ablauf von SiLEST. Des Weiteren werden die erzielten Ergebnisse ausführlich beschrieben sowie Einsatzempfehlungen für den in SiLEST entwickelten Testprozess gegeben. Schließlich werden der Nutzen und die Verwertbarkeit der Ergebnisse ausgeführt.

1.1 Aufgabenstellung

An den Entwicklungsprozess von Softwaresystemen, die in ein technisches Umfeld eingebettet sind, werden hohe Anforderungen an Korrektheit, Sicherheit und Zuverlässigkeit gestellt. Verifikation und Validation derartiger Systeme umfassen bis zu 30-50% des gesamten Entwicklungsaufwands. Oftmals können beide in der gegenwärtigen Praxis des System-Engineerings jedoch erst beim Vorliegen der entsprechenden Hardware durchgeführt werden. Dadurch sind der Verkürzung der Entwicklungszeiten durch zu geringe Parallelisierbarkeit der Entwicklungsprozesse für Hard- und Software Grenzen gesetzt.

Bedingt durch die Anforderungen an die Software hinsichtlich der Einbettung in ein technisches System wird die Verifikation der Software durch die üblicherweise minimal ausgestatteten Software-Schnittstellen erschwert. Eine weitere Besonderheit beim Test solcher Systeme besteht darin, dass die Ausgänge eingebetteter Systeme eine Rückkopplung auf ihre eigenen Eingänge ausüben. Die Software kann deshalb nicht losgelöst von ihrer technischen Umgebung getestet werden, sondern muss in dem Regelkreis untersucht werden, in dem sie eingebettet ist. Von besonderem Interesse ist in diesem Zusammenhang die Untersuchung der Auswirkungen von Sensor- und / oder Aktuator-Fehlerverhalten auf das Gesamtsystem.

Durch die Integration der zu testenden Software in eine Nachbildung des umgebenden technischen Systems als eine realistische und ausführbare Software-Simulation (Software-in-the-Loop, SiL) sollen die Softwaretests wesentlich früher, kostengünstiger sowie umfassender und zudem wiederholbar, automatisierbar sowie rückverfolgbar durchgeführt werden können.

Das Ziel des Projekts SiLEST ist es, Steuerungssoftware komplexer Anlagen und Geräte testen zu können, noch bevor die Ziel-Hardware zur Verfügung steht. Die originale Steuerungssoftware soll später ohne Änderungen auf ihrer Ziel-Hardware arbeiten. Auch für das Testen von gefährlichen oder riskanten Manövern ist es sinnvoll, einen

Simulator einzusetzen. Die benötigte Umgebung wird möglichst so detailliert simuliert, dass die Steuerungssoftware keinen Unterschied bemerken kann, ob sie sich in einer Umgebung mit realen oder virtuell simulierten Komponenten befindet. Die simulierten Sensoren und Aktuatoren sollen neben ihrem Nominalverhalten auch ein Fehlerverhalten aufweisen können, um derartige Effekte umfassend berücksichtigen zu können.

Für dieses Ziel sind Simulationsverfahren und -werkzeuge einzusetzen, diese anhand praktischer Beispiele zu analysieren und Werkzeuge für noch fehlende Verfahrensschritte zu entwickeln. Anhand von zwei typischen Systemen aus dem Automobilbau und der Raumfahrt sollen die Verfahren und Werkzeuge hinsichtlich ihrer Praktikabilität und der Einsatzgrenzen untersucht werden. Aufbauend auf diesen Ergebnissen ist ein kompletter Prozess für die Verifikation, den Test und die Validation eingebetteter Software unter Einbeziehung der Anforderungen an die Software beeinflusste Systemsicherheit und Zuverlässigkeit[1] zu entwickeln.

Die Vorbereitung der Tests, d. h. die systematische Definition von Testdaten und Testszenarien erfordert bereits bei der Anwendung der gängigen Testverfahren einen hohen Ressourcenanteil im Testprozess. Dieser Anteil wird durch die Anwendung von Simulationen im Test nicht verringert, wenn die Testmethodik losgelöst von den während der Systementwicklung eingesetzten Entwicklungsmethoden und -werkzeugen sowie den vorhandenen Daten betrachtet wird. Der SiL-Testprozess sollte daher von den in den Phasen des Entwicklungsprozesses erarbeiteten Informationen und dem Datenformat, in dem diese Informationen gespeichert werden, ausgehen. Ebenso sollten Daten durchgängig im gesamten Entwicklungsprozess genutzt werden können, wie beispielsweise Testfälle.

Durch die Analyse der auf dem Gebiet der Simulation laufenden Standardisierungsarbeiten und der Schnittstellen von allgemein angewendeten Simulationswerkzeugen sind die zum Aufbau einer SiL-Simulationsumgebung erforderlichen Schnittstellen und Modulbibliotheken so zu gestalten, dass sie wiederverwendungsfähig und branchenspezifisch austauschbar sind.

Dieser SiL-Testprozess ist innerhalb des Projekts in einem Werkzeug-Demonstrator zu implementieren und an praxisnahen Anwendungsbeispielen aus dem Automobilbereich und der Kleinsatellitentechnik zu erproben.

Es ist ein essentielles Ziel des Projekts, dass die Ergebnisse (Anwendungsbereiche der Verfahren, Testprozess, Demonstrator, Simulationsmodulbibliotheken) auf eingebettete Software von Produkten anderer Branchen übertragbar und anwendbar sind.

1.2 Projektplanung

Das Projekt SiLEST gliedert sich in fünf Hauptarbeitspakete. Diese sind:

- Definition des Verfahrensmodells für den SiL-Test (AP 2000)
- Modellierung der Simulationen für die betrachteten Anwendungsfälle (AP 3000)
- Anwendungserprobung des in AP 2000 definierten Verfahrensmodells (AP 4000)
- Erstellung eines Demonstrators für die Testprozess-Unterstützung (AP 5000)
- Erprobung des Demonstrators aus AP 5000 (AP 6000)

Das projektbegleitende AP 1000 umfasst die Tätigkeiten zum Management, der Dokumentation und der Außendarstellung des Projekts.

[1] Der Begriff "Zuverlässigkeit" wird hier inhaltlich wie "Verlässlichkeit" / Dependability benutzt.

Die Hauptarbeitspakete sind jeweils in drei bis sechs weitere Arbeitspakete unterteilt. Ergebnis jedes Arbeitspakets sind – neben der entwickelten Software – zwischen ein und sechs Abschlussdokumente.

1.3 Stand der Technik

1.3.1 Ist-Zustand bei den Projektpartnern vor Projektbeginn

1.3.1.1DLR

Das DLR setzt für die Entwicklung von On-Board-Software, Kommandierungssoftware von Raumfahrtmissionen, Manipulationssoftware von Weltraum-Robotern, Pilotenassistenzsystemen in der Luftfahrt u.a. schon langjährig HiL-Simulationssysteme ein. Mit der Etablierung der Verkehrsforschung im DLR werden HiL-Simulationen auch für Steuerungssoftware der Verkehrs- und Fahrzeugtechnik eingesetzt werden (z. B. für aktive Fahrerassistenzsysteme).

Bei den meisten dieser Systeme besteht das Problem, dass diese Simulationssysteme nicht die wirklichen Umgebungsbedingungen während des Betriebes widerspiegeln, da z. B. im Weltraum kein Schwerkrafteinfluss vorhanden ist oder problematische Betriebssituationen durch den möglichen entstehenden Schaden nicht realistisch testbar sind. Mit HiL-Simulationen ist in solchen Fällen die notwendige Vertrauenswürdigkeit in die Systemkorrektheit und -zuverlässigkeit sowie in die Software beeinflusste Systemsicherheit nicht vollständig herstellbar. Die SiL-Simulation soll hierfür künftig als effektives und ergänzendes Testverfahren zur HiL-Simulation eingesetzt werden.

Die Entwicklung von sicherheitskritischer eingebetteter Software im Rahmen neuartiger innovativer Produktkonzepte in der Raumfahrt, Luftfahrt und der Verkehrstechnik wird im DLR künftig weiter zunehmen, wobei der Zeit-, Kosten- und Qualitätsdruck auch bei diesen wissenschaftlichen Entwicklungen zum Einsatz effektiverer System- und Software-Engineering-Methoden zwingt.

Der Einsatz der Simulation für die Entwicklung innovativer wissenschaftlicher Systeme und Software ist auch eine industriell hochrelevante Thematik insbesondere hinsichtlich der Gewährleistung der Systemsicherheit beim Einsatz von nicht für die Raumfahrt qualifizierten Komponenten für On-Board-Systeme der kommerziellen Raumfahrt.

Für die Verbesserung der Verkehrssicherheit im Straßen- und Schienenverkehr werden im DLR neue, innovative Konzepte und prototypische Lösungen für aktive Fahrerassistenzfunktionen entwickelt. Die hochkritische eingebettete Echtzeit-Steuersoftware dieser Systeme erfordert bereits in den frühen Phasen ihrer Entwicklung ein umfassendes Safety- und Dependability-Management. Die mit SiLEST möglichen Simulationen werden hierzu einen effektiven Beitrag leisten können.

1.3.1.2IAV GmbH

Der Softwaretest sicherheitskritischer Systeme wird zukünftig aufgrund des zunehmenden Einsatzes so genannter X-by-Wire Steuerungen (z. B. elektronische Bremse oder Lenkung) an Bedeutung gewinnen.

Bisher wurden die Softwaretests für sicherheitskritische Systeme im Fahrzeug und an HiL-Prüfständen durchgeführt. Durch den Einsatz der SiL-Simulation können viele Tests unabhängig von kostenintensiven HiL-Testsystemen angeboten und durchgeführt werden. Dies bedeutet für die IAV GmbH eine höhere Flexibilität, so dass neue Kundenkreise erschlossen werden können. Durch die niedrigeren Kosten ist darüber

hinaus zu erwarten, dass die Akzeptanz der Simulation zum Softwaretest steigt und damit auch insgesamt die Auftragszahlen zunehmen.

Des Weiteren wird erwartet, dass sich der Einsatz von SiL-Methoden auf den gesamten Bereich der Funktionsentwicklung von Steuergerätesoftware übertragen lässt und somit gerade für Modultests verstärkt eingesetzt werden kann. Daraus würde sich ein beträchtliches Potenzial an einzusparender Zeit ergeben, was folglich kostenreduzierend wirken würde.

1.3.2 Werkzeugevaluation zu Projektbeginn

Zu Projektbeginn wurde eine ausführliche Evaluation der auf dem Markt befindlichen Software-Werkzeuge durchgeführt. Dabei wurden folgende Kategorien als relevant für das SiLEST-Projekt identifiziert: Werkzeuge zur Datenverwaltung, Entwicklungs-Werkzeuge, Steuerungs-Werkzeuge, Analyse- und Visualisierungs-Werkzeuge und sonstige Werkzeuge. Diese lassen sich wie folgt weiter untergliedern:

- Werkzeuge zur Datenverwaltung: Anforderungs-Management-Werkzeuge, Testfall-Management-Werkzeuge, Konfigurations-Management-Werkzeuge und Dokumentations-Management-Werkzeuge
- Entwicklungs-Werkzeuge: Systemmodellierung (UML), Testfall-Generierung, Modellierungs-Sprachen und Editoren / Simulatoren sowie Text-Editoren und Dokumentations-Werkzeuge
- Steuerungs-Werkzeuge: Hardware-Steuerung, Simulator-Steuerung und Prozess-Steuerung bzw. -Management
- Analyse- und Visualisierungs-Werkzeuge: Monitore zur Überwachung und Werkzeuge zur Test-Analyse / -Auswertung
- Sonstige Werkzeuge: XML-Editoren

Kurz zusammengefasst wurde festgestellt, dass ein Bedarf an einer Prozess-Steuerung für den im SiLEST-Projekt zu entwickelnden Testprozess bestand. Insbesondere fehlte eine Testfallbeschreibungssprache, welche alle Anforderungen an den Testprozess erfüllte. Zudem existierte kein Tool, um Testfälle in dem gewählten Format ohne XML- oder Programmierkenntnisse einzugeben. Für alle anderen Aufgaben waren kommerzielle bzw. teilweise auch freie Standard-Tools verfügbar.

Deshalb wurde beschlossen, im Rahmen von SiLEST keine komplette Testumgebung zu erstellen. Vielmehr sollte bereits weit verbreitete und ausgereifte Standard-Software über universelle Schnittstellen an eine neue Prozess-Steuerung angebunden werden. Um die Testautomatisierung auch Test-Ingenieuren ohne Programmierkenntnisse zugänglich zu machen, legten die Projektbeteiligten fest, einen Testfalleditor zur komfortablen Eingabe von XML-Testfällen zu programmieren.

1.3.3 Fremde Entwicklungen zeitgleich zum Projekt

Während der Laufzeit des Projekts wurden folgende Tools und Produkte beobachtet:

Es kamen zwei kommerzielle Tools auf den Markt, die das automatisierte Testen im Bereich der Entwicklung von Motorsteuerfunktionen unterstützen:

"ECU-Test" wurde an der TU Dresden entwickelt. Mit diesem Werkzeug konnten IAV-Kollegen in einem Kundenprojekt erste Erfahrungen sammeln. Die Stärke dieses Programms liegt bei der automatisierten Testdurchführung über Kommando-Folgen, die an Standard-Messprogramme geschickt werden. Testfälle werden dort also – im Gegensatz zu SiLEST – nicht über Eingangs- und Ausgangssignalverläufe, sondern über

Steuersequenzen definiert. ECU-Test besitzt keinen automatischen Bewerter. Es bietet auch keine Anbindung an Matlab und Simulink. Das Tool unterstützt eine breite Palette von Standard-Protokollen und Bus-Systemen.

"MTest" wird von der Firma dSPACE vertrieben. Es stammt ursprünglich von der DaimlerChrysler-Forschung Berlin. Mit diesem Programm existieren keine Erfahrungen bei den Projektpartnern.

Lange Zeit nur innerhalb des Daimler-Konzerns wurde die Testumgebung "Time Partition Testing TPT" von der DaimlerChrysler-Forschung Berlin eingesetzt.[2] Die Daimler-Ausgründung PIKETEC GmbH strebt eine freie Vermarktung des Tools ab dem Sommer 2007 an, so dass erst dann ein Vergleich mit dem SiLEST-Testprozess möglich ist.

Zeitgleich zu SiLEST, aber mit deutlich späterem Entwicklungshorizont entstand der IEEE-Standard "ATML (Automatic Test Markup Language)" zur Beschreibung von automatischen Testsystemen. Der ATML-Homepage[3] ist zu entnehmen, dass nach aktueller Planung ATML im Herbst 2007 als Standard verabschiedet werden soll.

Die Definition eines derartigen Sprachstandards durch eine renommierte Institution wie das IEEE lässt eine zukünftige Verbreitung der Sprache erahnen. Allerdings ist die Festlegung eines Standards allein noch keine Garantie dafür, dass dieser in der Praxis auch tatsächlich angenommen wird. Auch gibt es für neue Standards anfangs keine oder nur eine geringe Unterstützung durch andere Tools.

Für das SiLEST-Projekt kam der neue IEEE-Standard in jedem Falle zu spät. Überlegungen, die eigene Testfallbeschreibungssprache während der Projektlaufzeit noch grundlegend zu verändern und an den (damals noch nicht abschließend definierten) Standard anzupassen, wurden verworfen, da sie zu aufwendig und risikobehaftet erschienen.

Intensiv untersucht wurde die Testbeschreibungssprache "TTCN3 (Testing and Test Control Notation)". TTCN3 wurde bislang hauptsächlich zum Testen von Kommunikationsnetzen eingesetzt. Die Sprache hat ihre Stärken im Bereich der Beschreibung zeitdiskreter und ereignisbasierter Signale. Dagegen besitzt es Defizite bei der Beschreibung analoger Signale und im Bereich der harten Echtzeitfähigkeit. Analoge Signale werden dagegen von der Testfall-Beschreibungssprache von SiLEST schwerpunktmäßig unterstützt, wohingegen hier ereignisbasierte Beschreibungen nur eingeschränkt möglich sind.

Um die Stärken beider Konzepte zu vereinen, fand zwischen der IAV und der Firma testing_tech, die TTCN3 mit entwickelt hat und deren Implementierung jetzt kommerziell vermarktet, ein Gedankenaustausch statt. Der dabei für das Zusammenführen der unterschiedlichen Konzepte identifizierte Aufwand überstieg jedoch nach Einschätzung der Gesprächsteilnehmer den Rahmen des laufenden SiLEST-Projekts. Mittlerweile versucht testing_tech alleine, TTCN3 zum Testen zeitkontinuierlicher Systeme weiterzuentwickeln. Unserer Einschätzung nach wird es jedoch noch einige Zeit dauern, bis eine fertige Umsetzung verfügbar ist.

[2] Beschreibung TPT-Verfahren: http://www.systematic-testing.com/functional_testing/tpt.php

[3] ATML-Homepage: http://grouper.ieee.org/groups/scc20/tii/atml_status.htm

Mit TestML steht eine Beschreibungssprache für Testfälle im XML-Format zur Verfügung. Diese Beschreibungssprache ermöglicht es jedoch nur, analoge Signalverläufe zu definieren und zu kaskadieren. Die Mächtigkeit und Unterstützung der automatischen Testfallauswertung des in SiLEST definierten XML-Testfall bietet TestML jedoch nicht.

2 Testprozess

Dieser Abschnitt beschreibt den Prozess, welcher für die Durchführung eines SiL-Softwaretests notwendig ist. Die diesem Abschnitt zugrunde liegenden Prozessablaufdiagramme werden in Abschnitt D dargestellt. Eine Auflistung der verwendeten Symbole zur Prozessbeschreibung ist in Abbildung D-1 zu finden.

Abbildung D-2 zeigt den groben Ablauf des SiL-Softwaretests. Der SiL-Testansatz unterscheidet sich nicht grundlegend von anderen Testansätzen.

Grundlage für den Test ist die Definition von Nutzungsszenarien, welche aus Anwendersicht zu definieren sind bzw. vom Anwender zu definieren sind. Für den Automobilbereich sind beispielsweise Nutzungsszenarien Fahrzyklen, welche später durch den Fahrer vorgenommen werden. Für ein Satellitensystem sind dies Kommandosequenzen, welche beim späteren Betrieb ausgeführt werden.

Der Testleiter bestimmt zunächst, welche Tests durchgeführt werden sollen. Dieser Schritt wird im Detail in Abschnitt 2.1 beschrieben. Sofern nicht alle erforderlichen Simulationskomponenten und Testfälle für die ausgewählten Tests vorhanden sind, müssen die fehlenden Simulationskomponenten und Testfälle durch den Testingenieur bzw. Simulationsspezialisten erstellt werden. Dieser Prozessschritt wird im Abschnitt 2.2 näher beschrieben. Vor der Verwendung der neu erstellten Simulationskomponenten und Testfälle müssen diese durch den Testleiter freigegeben werden.

Nach der Freigabe der Testfälle kann der Testleiter Testfälle zu einer Testsuite zusammenstellen und zur Ausführung anweisen. Die Testfälle gelangen dann im Prozessschritt Testdurchführung, der im Abschnitt 2.3 behandelt wird, zur Ausführung. Nach der Testdurchführung erfolgt eine Testauswertung, welche in Abschnitt 2.4 beschrieben ist.

Sofern alle Testfälle durchführbar waren, kann der Testleiter im letzten Prozessschritt einen Testbericht erstellen und den Prozess des SiL-Softwaretests abschließen. Sofern Testfälle nicht ausführbar waren, sind diese Testfälle erneut zur Testdurchführung anzuweisen. Alle Ergebnisse und erstellten Testdokumente im Testprozess werden versioniert auf dem Testdokumentenserver festgehalten.

2.1 Testauswahl

Der Prozessschritt der Testauswahl wird durch den Testleiter durchgeführt. Eine Darstellung der Testauswahl lässt sich Abbildung D-3 entnehmen. Bei den Tests ist zwischen Tests für den Nominalbetrieb und Tests von einem gestörten Betrieb des zu testenden Systems zu unterscheiden.

Für den Fall, dass Nutzungsszenarien im Nominalbetrieb getestet werden sollen, werden aus den aus Anwendersicht erstellten Nutzungsszenarien die für das Testziel relevanten Nutzungsszenarien ausgewählt und in den Testanforderungen auf dem Testdokumentenserver abgelegt.

Falls in den Tests auch ein gestörter Betrieb betrachten werden soll, so sind als erstes die Nutzungsszenarien auszuwählen, in denen Störungen auftreten sollen. Für jedes dieser ausgewählten Nutzungsszenarien sind auf Basis einer FMEA die kritischen Komponenten zu bestimmen, welche durch einen Ausfall oder einen fehlerhaften Betrieb eine kritische Situation auslösen können, sofern die Software diese Störungen nicht korrekt verarbeitet. Als Auswahlkriterium kann hierzu beispielsweise die Risikobewertung der FMEA verwendet werden. Als nächstes ist für die kritischen Komponenten das Störverhalten zu definieren. Dabei können für eine kritische Komponente auch mehrere

Störverhalten betrachtet werden. Als Grundlage für die Auswahl kann für diesen Prozessschritt ebenfalls die Risikobewertung der FMEA herangezogen werden.

Die Kombinationen aus Nutzungsszenarien, kritischen Komponenten und deren Ausfallverhalten werden als Testanforderungen in der Testdokumentation festgehalten und auf dem Testdokumentenserver abgelegt. Der Prozessschritt der Testauswahl endet mit der erfolgten Ablage der Testanforderungen.

2.2 Testerstellung

Beim Prozessschritt der Testerstellung ist zu unterscheiden zwischen der Erstellung der Simulation und dem Erstellen von Testfällen. Eine Darstellung dieses Prozessschritts ist in Abbildung D-4 zu finden. Die Arbeiten für die Erstellung von Testfällen und der Erstellung der Simulation können parallel laufen, sofern die Schnittstellen der verwendeten Simulationsmodule im Voraus bekannt sind.

Als Grundlage für die Erstellung der Simulation dienen Simulationsmodule, aus denen die Simulation zusammengesetzt wird. Ein Simulationsmodul kann zum Beispiel die Simulation eines bestimmten Sensortyps sein, welcher von dem zu testenden System verwendet wird. Simulationsmodule werden in ein Simulationsmodell-Repository revisionssicher gespeichert. Sofern nicht alle Simulationsmodule für die Simulation bereit stehen, sind zunächst diese Simulationsmodule zu erstellen, danach kann dann die Simulation erstellt werden.

Die Erstellung von Simulationsmodulen ist in Abschnitt 2.2.1 und die Erstellung der Simulation in Abschnitt 2.2.2 näher beschrieben. Das Erstellen von Testfällen wird in Abschnitt 2.2.3 vorgestellt.

2.2.1 Erstellen von Simulationsmodulen

Das Erstellen von Simulationsmodulen wird von einem Simulationsentwickler durchgeführt. Eine grafische Darstellung des Prozesses zur Erstellung von Simulationsmodulen ist in Abbildung D-5 enthalten.

Der Prozess startet mit der Auswahl eines fehlenden Simulationsmodells, welches für die Erfüllung der Testanforderungen in der Testdokumentation noch modelliert werden muss. Zur Reduktion der Kosten für einen Test sollten Simulationsmodule wiederverwendbar gestaltet werden. Das heißt, dass zu einem Simulationsmodul noch weitere Informationen über ein Simulationsmodell gegeben werden, wie beispielsweise eine detaillierte Schnittstellenbeschreibung, die auf das Modell angewandten Tests, in welchen Projekten das Modell verwendet wurde usw. Ein Simulationsmodell unterscheidet sich von einem Simulationsmodul dadurch, dass es lediglich ausführbarer Code ist. Ein Simulationsmodul besteht aus ausführbarem Code und weiteren Informationen über das Simulationsmodell.

Nach der Auswahl eines fehlenden Simulationsmodells sucht der Simulationsentwickler zunächst nach einem Basismodell oder einem ähnlichem Simulationsmodell in dem Simulationsmodell-Repository, welches als Grundlage für die Modellierung verwendet werden kann. Befindet sich ein entsprechendes Modell im Simulationsmodell-Repository, so kann dies Modell entsprechend erweitert werden, damit es die geforderten Funktionalitäten bietet. Lässt sich kein passendes Basismodell finden, so ist ein neues Modell mit den geforderten Funktionalitäten zu erstellen.

Anschließend kann das Simulationsmodell entsprechend den Anforderungen parametriert werden, um es danach hinsichtlich seiner Korrektheit zu testen. Als Referenz für den Test werden physikalische Zusammenhänge überprüft oder das reale Verhalten eines Sensors oder Aktuators, sofern es sich um ein Simulationsmodell eines

Sensors oder Aktuators handelt. Schlägt ein Test des Simulationsmodells fehl, so ist die Fehlerursache zu lokalisieren und das Simulationsmodell zu korrigieren.

Waren alle Tests des Simulationsmodells korrekt, so wird das Simulationsmodell als Simulationsmodul an den Testleiter übergeben, welcher es noch einmal formal überprüft und bei Korrektheit freigibt. Mit der Freigabe eines Simulationsmoduls erfolgt die Übernahme des Simulationsmoduls in das Simulationsmodell-Repository.

Die Dokumentation und Vorgehensweise bei der Erstellung der Simulationsmodule muss den Anforderungen genügen, welche gemäß dem Kritikalitätslevel an das Endprodukt gestellt werden. Dies ist beispielsweise zwingend notwendig bei einer Entwicklung im Luftfahrtbereich nach der DO178B.

Der Prozessschritt wird solange wiederholt, wie es noch weitere fehlende Simulationsmodelle gibt, welche zur Erfüllung der Testanforderungen benötigt werden.

2.2.2 Erstellen der Simulation

Der Prozess der Erstellung der Simulation für einen SiL-Test ist in Abbildung D-6 abgebildet. Er startet mit der Auswahl der Simulationsmodule. Die Auswahl geschieht auf Basis der Testanforderungen, einem Modell (z. B. SysML [3]) der Hardware des technischen Systems und der im Simulationsmodell-Repository verfügbaren Simulationsmodule. Die Modelle der ausgewählten Simulationsmodule werden gemäß dem Informationsfluss in dem System miteinander verknüpft. Abschließend wird die Simulation getestet. Verhält sie sich korrekt, so wird die Simulation durch den Testleiter für die Durchführung der Tests frei gegeben. Wurden in der Simulation Fehler festgestellt, so ist die Simulation noch einmal durch den Simulationsentwickler zu überarbeiten. Die Vorgehensweise für die Freigabe der Testumgebung richtet sich nach den geltenden Entwicklungsnormen für den Anwendungsbereich und erfüllt die Anforderungen hinsichtlich der Kritikalität der zu testenden Software.

2.2.3 Erstellen von Testfällen

Neben der Simulation der Umgebung der zu testenden Software sind die notwendigen Testfälle zu erstellen. Testfälle setzen sich zusammen aus einem Betriebsszenario, wie beispielsweise dem Fahrerverhalten, und falls ein Ausfallverhalten getestet werden soll, einem oder mehreren Auslösern von Ausfällen. Grundlage für die Erstellung von Testfällen sind die ausgewählten Testszenarien, welche getestet werden sollen. Eine grafische Darstellung des Prozesses für die Erstellung eines Testfalls ist in Abbildung D-7 gegeben.

Als erstes überprüft der Testingenieur, ob ein Testfall im Testfall-Repository vorhanden ist, welcher ein ähnliches Testszenario wie gefordert beschreibt. Ist dies der Fall, so lässt sich der Testfall durch ein einfaches Duplizieren und Abändern des ähnlichen Testfalls erreichen.

Sofern kein ähnlicher Testfall im Testfall-Repository vorhanden ist, ist zunächst das Bedienermodell zu erstellen. Das heißt, die Aktivitäten des Benutzers werden durch ein Skript, ein Simulationsmodell oder durch aufgezeichnete Benutzeraktivitäten in eine für den Computer ausführbare Form gebracht. Dies geschieht in der Regel durch die Angabe von analogen Eingangsverläufen oder die Angabe von Kommandos an das zu testende System.

Sofern ein Testfall auch eine Fehlersimulation enthalten soll, ist die Benutzeraktivität durch Auslöser für die Fehlerfälle zu ergänzen. Ein entsprechender Auslöser wird beschrieben durch den Zeitpunkt und eine Schnittstelle, um die Fehlersimulation in der

Simulation der technischen Umgebung zu aktivieren. In den SiLEST-Werkzeugen steht zu diesem Zweck ein standardisierter Fehlertrigger im Testfallformat bereit.

Nachdem das Bedienermodell und die Fehlerereignisse definiert worden sind, ist das erwartete Verhalten des Systems zu definieren. Das Verhalten lässt sich zum einen durch eine ausführbare Spezifikation testen, mit der das Sollverhalten beschrieben wird, zum anderen aber auch durch Definition von Werteverläufen von beobachtbaren Werten in der Simulation oder in dem zu testenden System. Es sollte mit akzeptablen Toleranzen überprüft werden, da ansonsten bei einer automatisierten Testdurchführung häufig mit dem Testurteil *Failed* gerechnet werden muss, ohne dass wirklich ein Fehler in der zu testenden Software zu Grunde liegt.

Nach der Definition des erwarteten Verhaltens lässt sich noch ein Grenzverhalten definieren. Das Grenzverhalten lässt sich in der Regel aus der Spezifikation des zu testenden Systems gewinnen und ist unabhängig von einem speziellen Bedienerverhalten. Solch ein Grenzverhalten kann zum Beispiel die Obergrenze einer Beschleunigung sein, welche im Rahmen des Betriebs nicht überschritten werden darf.

Nachdem ein Testfall komplett definiert wurde, wird er durch den Testingenieur in das Testfall-Repository eingecheckt und steht für die Testausführung bereit.

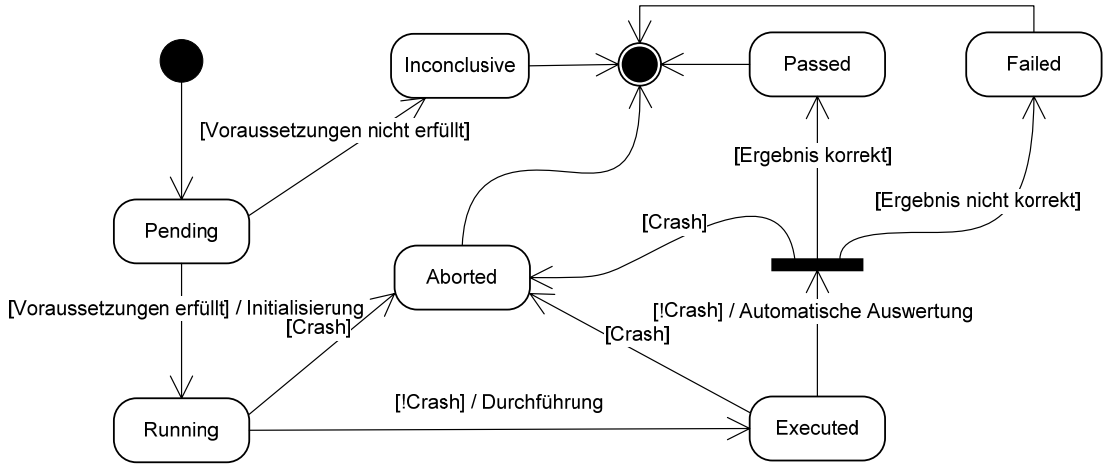

Abbildung 2-1 Zustandsdiagramm der Testfallbewertung

2.3 Testdurchführung

Die Testdurchführung des SiL-Softwaretests soll automatisch in einer Testumgebung erfolgen. Der Ablauf dieser Testdurchführung wird in Abbildung D-8 gezeigt. Abbildung 2-1 zeigt das Zustandsdiagramm der Testfallbewertung und damit, welchen Teststatus ein Testfall während der Durchführung annehmen kann.

Die Testdurchführung beginnt mit dem Start der Simulationsumgebung. Nach dem Start der Simulationsumgebung laufen alle für den Test erforderlichen Systeme und Prozesse und befinden sich in einem initialisierten Anfangszustand. Sofern sich nicht alle erforderlichen Systeme und Prozesse in einen definierten Zustand zurücksetzen lassen, muss die Simulationsumgebung für jeden Testfall neu gestartet werden.

Alle Testfälle, welche durchgeführt werden sollen, werden vor dem Start auf den Teststatus *Pending* gesetzt. Der Testmanager sucht sich unter den Testfällen einen Testfall aus, welcher den Teststatus *Pending* besitzt und initialisiert die Testumgebung und die zu testende Software gemäß dem Testfall. Dabei werden vom Testmanager die möglichen Abhängigkeiten zwischen den Testfällen berücksichtigt. Der Testfall erhält

nach erfolgreicher Initialisierung den Teststatus *Running*. Werden die im Testfall spezifizierten Voraussetzungen durch die Simulation und das zu testende System nicht erfüllt, dann erhält der Testfall das Testergebnis *Inconclusive*. Kommt es während der Initialisierung des Systems zu einer Störung, so erhält der Testfall das Testergebnis *Aborted*.

Unmittelbar nach der Initialisierung startet der Testmanager die Testausführung, welche nach dem Erreichen einer definierten Simulationszeit oder bei einem schweren Fehler der zu testenden Software beendet wird. Nach der erfolgreichen Testausführung erhält der Testfall den Teststatus *Executed*. Dies heißt, der Testfall wurde ausgeführt, es wurde aber noch kein Ergebnis zugewiesen. Im Fall einer fehlerhaften Durchführung durch einen aufgetretenen Crash erhält der Testfall das Testergebnis *Aborted*.

Die anschließende Testauswertung legt die Testergebnisse in einem Massenspeicher ab und vergleicht das zu erwartende Verhalten mit dem gezeigten Verhalten. Differieren beide Verhalten zu stark voneinander oder wurde das erwartete Ergebnis nicht erreicht, so erhält der Testfall das Testergebnis *Failed.* Ebenso erhält der Testfall das Testergebnis *Failed*, wenn ein spezifiziertes Grenzverhalten durch das gezeigte Verhalten verletzt wird. Entspricht das gezeigte Verhalten dem erwarteten Verhalten und hält das Grenzverhalten ein, so ist das Ergebnis korrekt, und der Testfall erhält das Testergebnis *Passed*. Sollte die Auswertung nicht fehlerfrei durchlaufen, sondern einen Crash produzieren, so erhält der Testfall den Zustand *Aborted*.

Nach der Testauswertung führt der Testmanager eine Deinitialisierung der Simulation und der zu testenden Software durch. Nach der Deinitialisierung befinden sich die Simulation und das zu testende System in einem definierten Zustand. Sie sind in dem definierten Zustand bereit für die Ausführung des nächsten Testfalls mit dem Teststatus *Pending*. Ist kein weiterer Testfall mit dem Teststatus *Pending* vorhanden, so wird die Simulationsumgebung geschlossen. Mit dem Schließen der Simulationsumgebung ist der Prozessschritt der Testdurchführung abgeschlossen.

2.4 Testauswertung

Der abschließende Prozessschritt der Testauswertung durch den Testingenieur, dessen Ablauf in Abbildung D-9 grafisch dargestellt wird, ist der letzte Prozessschritt des SiL-Softwaretests. In diesem Schritt werden alle Testergebnisse der automatisierten Testdurchführung durch den Testingenieur anhand der Zustände der Testfälle überprüft.

Aus den Testfällen mit dem Testergebnis *Passed* werden stichprobenartig einige Ergebnisse überprüft, um Testfälle aufzudecken, die trotz fehlerhaften Verhaltens zu einen fehlerfreiem Testergebnis geführt haben. Für den Fall, dass die Vergabe des Testurteils *Passed* nicht korrekt ist, wird dem Testfall ein neues Ergebnis zugewiesen. Die Größe der Stichprobe sollte entsprechend den Erfahrungen mit der verwendeten Simulationsumgebung und den Testfällen gewählt werden. Bei Regressionstests kann dieses auch entfallen. Bei neuen Testfällen oder einer komplett neuen Simulation sollte eine größere Stichprobe gezogen werden.

Besitzt ein Testfall das Testergebnis *Failed*, so werden zunächst die Ähnlichkeitsfunktion und die Prädikate des Testfalls überprüft. Wenn diese keinen Fehler enthalten, so ist ein Fehlerreport an den Softwareentwickler zu erstellen. Ist eine Ähnlichkeitsfunktion oder ein Prädikat fehlerhaft, so ist der Fehler in dem Testfall zu beheben und der Test erneut für einen Test anzuweisen.

Falls der Testfall das Testergebnis *Executed* besitzt, ist die automatische Auswertung zu überprüfen, und es ist ein Fehlerreport mit der Fehlerursache an den Testingenieur zu erstellen.

Besitzt ein Testfall das Testergebnis *Inconclusive*, so wurde der Testfall nicht durchgeführt, weil aufgrund der überprüften Vorbedingungen für den Testfall keine sinnvollen Ergebnisse zu erwarten waren. Dies ist der Fall, wenn die im Testfall angegebenen Prädikate der Vorbedingungen nicht erfüllt sind oder ein vorausgehender abhängiger Testfall nicht mit dem Testergebnis *Passed* abgeschlossen werden konnte.

Hat ein Testfall den Teststatus *Running* oder *Aborted*, so wurde der Testfall nicht zu Ende ausgeführt, oder es gab einen Abbruch des Testlaufs durch einen Absturz in der Simulation oder dem zu testenden System. Die Ursache für diesen Abbruch ist zunächst zu untersuchen. Wurde der Abbruch durch die zu testende Software hervorgerufen, so ist ein Fehlerreport an den Softwareentwickler zu generieren. Sollte der Fehler in der Simulationsumgebung begründet liegen, so ist ein Fehlerreport an den Simulationsentwickler zu generieren. Sollte die Fehlerursache nicht auf Seiten der Simulation und der zu testenden Software zu finden sein, so ist ein Fehlerreport an den Testleiter zu generieren, der über weitere Maßnahmen zur Behandlung des Fehlers entscheiden muss.

Sollte ein Testfall noch den Teststatus *Pending* besitzen, so sind die Ursachen hierfür festzustellen und zu beheben. Für den Testfall ist daraufhin ein neuer Testlauf anzuweisen. Dieser Teststatus sollte aber bei einer voll funktionsfähigen Testautomatisierung nicht auftreten.

Alle in dem Prozessschritt erstellten Testreporte werden zur Dokumentation des Testablaufs unter ein Versionsmanagement gestellt. Die in der Rolle des Testingenieurs an der Testausführung und Testauswertung beteiligten Personen müssen nicht identisch sein.

3 Umsetzung des Testprozesses

3.1 Allgemeine Testumgebung

Die Testumgebung setzt sich aus drei allgemeinen Bestandteilen zusammen (vgl. Abbildung 3-1).

3.1.1 Testverwaltung

Die Testverwaltung stellt den Zugang zu den einzelnen Tests und ihren Ergebnissen bereit. Sie verwaltet dazu alle zugehörigen Testfälle und -resultate. Damit lässt sich jederzeit nachvollziehen, welche Tests mit welchem Ergebnis durchgeführt wurden. Gleichfalls gehört die Anbindung an das Requirement-Management zu den Aufgaben der Testverwaltung.

3.1.2 Versionsverwaltung

Die Versionsverwaltung übernimmt die Versionierung aller im Rahmen des Tests verwendeten Dateien, um eine umfassende Reproduzierbarkeit zu gewährleisten.

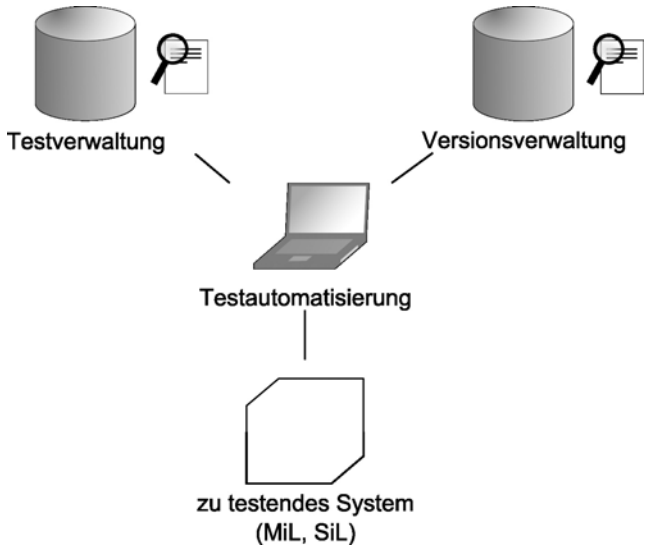

Abbildung 3-1 Testumgebung

3.1.3 Testautomatisierung mit dem zu testenden System

Für die automatisierte Durchführung von Tests auf dem zu testenden System sorgt eine in SiLEST neu entwickelte Anwendung. Die Anwendung heißt Testablaufsteuerung und unterstützt aktuell:

- als Simulator: Matlab / Simulink (The Mathworks),
- als Testverwaltung: TestDirector / Quality Center (HP), Testmaster (Open Source), verzeichnisorientierte lokale Ablage,
- als Testverwaltung: StarTeam (Borland), Subversion (Open Source), verzeichnisorientierte lokale Ablage.

Die Anbindung dieser Werkzeuge wird über eine Plugin-Schnittstelle realisiert, welche lokal oder über SOAP die Anforderung aus der SiLEST-Testautomatisierung auf die API der Werkzeuge umsetzen. Damit wird eine hohe Flexibilität erreicht, um Werkzeuge von Drittanbietern in den Testprozess einzubinden.

3.2 XML-Formate

Die SiLEST-Testumgebung arbeitet mit Testfall-Dateien (Testfällen), die unabhängig von einer Testmethode (z. B. SiL) definiert sind. Damit wird eine Wiederverwendbarkeit der Testfälle für andere Testmethoden erreicht, was z. B. eine direkte Vergleichbarkeit von SiL- und HiL-Testergebnissen ermöglicht. Um die Tests mit den zu testenden Komponenten (z. B. in der SiL-Variante) durchführen zu können, wird der Aufbau aller zu testenden Komponenten in der Testkonfiguration beschrieben, und es wird ein Mapping des Testfalls auf diese Beschreibung vorgegeben. Die dazu notwendige Abstraktion zwischen Testfall und Testsystem wird über die Verwendung von Labeln erreicht, die in einer Namensraum-Datei definiert werden. Die Datei wird vom Testfalleditor für die benutzerfreundliche und fehlerrobuste Eingabe der Testfälle und der Testkonfiguration verwendet. Die bei der Durchführung der Testfälle erzielten Resultate werden in einem für die Weiterverarbeitung geeigneten Testreport-Format gesichert.

Alle Dateien sind XML-Dateien, und es existiert jeweils ein zugehöriges XML-Schema, das den Aufbau und die Inhalte spezifiziert sowie umfangreich dokumentiert. Die Vorteile eines XML-Datenformats liegen vor allem in der klaren, offenen Strukturierung der Daten, was auf einfache Weise die Wiederverwendbarkeit über entsprechende Transformationen ermöglicht. Des Weiteren garantiert ein Schema eine einfache Validierung der Dateistruktur.

Im Folgenden wird der wesentliche Aufbau der Dateien beschrieben, um einen Eindruck der zugehörigen Leistungsfähigkeit zu vermitteln.

3.2.1 Namensraum-Datei

Die Namensraum-Datei definiert für ein Projekt (d. h. für eine zu testende Funktion) die zu verwendenden Label. Label sind dabei eine abstrakte Bezeichnung für Ein- bzw. Ausgänge sowie Parameter von Testsystemkomponenten. Beispielsweise können für den Test eines potentiellen Regelungsalgorithmus folgende Label definiert werden: Vorgabesignal (soll), erzieltes Verhalten (ist) und ein Parameter (K).

Mit Hilfe dieser Label lassen sich die Testfälle unabhängig vom Testsystem definieren, was eine Wiederverwertung bzw. unterschiedliche Testmethoden ermöglicht. So ist während der Testfalldefinition im Beispielfall egal, ob die Sollvorgabe an Pin 21 eines HiL-Systems oder an den Eingang Regler_Input eines Simulink-Modells gelangen muss.

Ein Namensraum sichert dabei die in ihm definierte Menge an Labeln der Testablaufsteuerung zu.

3.2.2 Testkonfiguration

Die Testkonfiguration besteht aus einer abstrakten Beschreibung des Testsystems und der Zuordnung der Ein- und Ausgänge sowie der Parameter zu den definierten allgemeinen Testfall-Labeln. In der Beschreibung des Testsystems sind die einzelnen Komponenten des Testsystems so abgebildet, dass die Testablaufsteuerung alle nötigen Informationen erhält. Im zweiten Teil findet ein Mapping der Komponenten-Eingänge auf die im Testfall definierten Testsignale und der Komponenten-Ausgänge auf die definierten Signale / Ports der Tests statt.

Die Testkonfiguration enthält zudem formale Informationen, wie die aktuelle Version oder den Autor. Außerdem muss der Typ der Testmethode angegeben werden. Definieren lassen sich Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL) und Hardware-in-the-Loop (HiL). Ein weiterer wesentlicher Bestandteil sind Kommentare, die für den automatisch generierten PDF-Testreport zur Dokumentation verwendet werden.

3.2.2.1Definition der Komponenten des Testsystems

Der erste Teil der Testkonfiguration besteht aus der Definition der Komponenten des zu testenden Systems. In der aktuellen Version der Testkonfiguration werden alle Komponenten in Form von "Modellen" definiert.

Ein Modell liefert eine (aus Sicht der Testablaufsteuerung) vollständige Beschreibung der entsprechenden Komponente des zu testenden Systems. Nur bei einer vollständigen Definition ist eine einfache Weiterverwendung erreichbar. Das zu testende System kann sich dabei aus ein oder mehreren Bestandteilen zusammensetzen.

Für ein Modell lassen sich Informationen wie Name, benötigtes Plugin, Betriebsart wie beispielsweise Master / Slave-Betrieb usw. festlegen. Des Weiteren werden die Eingänge (Verbindungen in das Modell hinein), Ausgänge (Verbindungen aus dem Modell heraus) und Parameter (Werte, die dem Modell gesetzt werden können) definiert. Ebenfalls werden alle zugehörigen Dateien aufgelistet. Das ist im Normalfall mindestens die Datei, die das Modell beinhaltet (z. B. Matlab-MDL-Datei oder Binärdatei für ein Steuergerät).

3.2.2.2Definition des Mappings

Das Mapping verknüpft die Ein- und Ausgänge sowie Parameter des Testsystems mit den im Testfall verwendeten Labeln. Die Label sind dabei in ein oder mehreren Namensraum-Dateien definiert.

Durch diese Vorgehensweise lassen sich Testfälle unabhängig von einem konkreten Testsystem entwickeln und zudem einfach wiederverwenden.

Das Mapping besteht aus einzelnen Verknüpfungen. Jeder Ein- oder Ausgang einer Testsystemkomponente kann entweder gegen einen anderen Aus- bzw. Eingang einer anderen Komponente oder gegen ein bzw. mehrere Label aus dem Testfall bzw. den Testfällen gemappt werden.

Im ersten Fall muss der Kopplungsmanager der Testablaufsteuerung während der Simulation den Datenaustausch zwischen den Testsystemkomponenten sicherstellen. Anderenfalls nutzt die Testablaufsteuerung die Ein- bzw. Ausgänge, indem sie diesen (wenn vom Testfall definiert) Signale vorgibt oder diese aus ihnen liest. Es ist dabei möglich, einen Ein- oder Ausgang gegen mehrere Label aus verschiedenen Namensräumen zu mappen. Damit lassen sich Testfälle aus verschiedenen Quellen (die unterschiedliche Namensräume benutzen) in einer Testkonfiguration verknüpfen. Dies ist wichtig vor dem Hintergrund einer Wiederverwendung von Testfällen.

Zudem können die Parameter einer Testsystemkomponente gegen ein oder mehrere (aus verschiedenen Namensräumen) Label gemappt werden.

3.2.3 Testfälle

Das Testfallformat ermöglicht die Spezifikation von Tests. Dabei handelt es sich um funktionale Anforderungen an die zu untersuchende Funktion, die hauptsächlich durch ein analoges Signalverhalten geprägt ist.

Ein Testfall setzt sich aus verschiedenen Bestandteilen zusammen. Zuerst werden formale Informationen, wie die textuelle Beschreibung des Testfalls, der Autor, die Schemaversion u.ä. vermerkt. Dem folgt eine Definition der testspezifischen Parameter (Abweichungen von der Grundparametrierung) von System und Testumgebung.

Der nächste Teil nimmt die Definition von analogen Eingangsverläufen oder Kommandosequenzen vor. Um eine automatische Auswertung des Testfalls zu ermöglichen, wird im letzten Teil das erwartete Ergebnis definiert. Dieses setzt sich aus binären Einzelresultaten zusammen, die über logische Verknüpfungen zu einem Gesamt-Testresultat verbunden werden.

Alle Elemente bieten die Möglichkeit einer umfangreichen Kommentierung. Dies ist bedeutsam, da die Kommentare beim automatischen Erstellen der PDF-Testreporte verwendet werden.

3.2.3.1Definition von Konfiguration und Kalibrierung

Es ist entweder möglich, dass ein Testfall von einem anderen Testfall abhängt, oder der Testfall kann eine Kalibrierung und Konfiguration definieren.

Soll ein Testfall von einem anderen abhängen, so muss dies im abhängigen Testfall angegeben werden. Eine Abhängigkeit bedeutet, dass der Testfall nur ausgeführt wird, wenn der Testfall, von dem er abhängt, erfolgreich getestet wurde. Andernfalls wird ein Resultat vergeben, das kennzeichnet, dass der Testfall nicht ausgeführt werden konnte. Die Abhängigkeit wird über die Vorgabe des Testfallnamens definiert.

Nicht abhängige Testfälle können eine Kalibrierung sowie Konfigurationsinformationen (d. h. Größen, die keine Parameter sind) vorgeben. Im SiLEST-Prozess wird prinzipiell davon ausgegangen, dass die Komponente eine Grundparametrierung aufweist. Der Testfall definiert die Abweichungen von dieser Grundparametrierung. Damit ist nur eine minimale Anzahl an Definitionen in jedem Testfall nötig. Prinzipiell besteht jedoch auch die Möglichkeit, von dieser Vorgehensweise abzuweichen. Als Parameter sind skalare Größen, Vektoren und Matrizen zulässig.

3.2.3.2Definition der Signalvorgaben

Signalvorgaben lassen sich durch die sehr leistungsfähige Signaldefinition einfach spezifizieren. Diese unterstützt als Basisbestandteile konstante Werte, Linien, Rampen, Sprünge, Zeit-Werte-Vektoren und Funktionen. Die Basisbestandteile können stückweise zur Definition komplexer Signale verwendet werden, wobei immer für eine vollständige Definition gesorgt wird.

Alternativ zu den beschriebenen analogen Signalverläufen können diskrete Kommandos definiert werden. Diese werden zum vorgegebenen Testzeitpunkt an das Testsystem übergeben und müssen von einem speziellen externen Compiler in ein Eingangssignal übersetzt werden. Dies können beispielsweise Satellitenkommandos sein, welche in der Form `"XMT[<time tag>: <Command name>; <parameter liste>];"` definiert werden und in den Bytecode des Satelliten-Upstream übersetzt werden müssen.

3.2.3.3Definition von Fehlerverhalten

Im Testfall können verschiedene Arten von Fehlerverhalten für Signale definiert werden. Zweck der Definition ist die Untersuchung von Signalstörungen oder Alterungserscheinungen von Sensoren oder Aktuatoren. Bei analogem Fehlerverhalten handelt es sich um eine Beeinflussung des Verhaltens von analogen Signalen. Um allgemeine Fehlerbilder nachbilden zu können, wurden drei prinzipielle Beeinflus-

sungsarten identifiziert (ersetzend, additiv, modifizierend). Diese können auch parallel zum selben Zeitpunkt aktiv sein. Sie besitzen jedoch eine Priorisierung, die u.U. das erzielte Fehlerverhalten beeinflusst.

Beim additiven Fehlerverhalten wird dem analogen Signal ein additiver Anteil (Signal oder Rauschen) hinzugefügt. Durch modifizierendes Fehlerverhalten wird das Signal in Abhängigkeit von sich selbst verändert. Die Veränderung erfolgt durch eine Multiplikation mit dem vorgegebenen Signal. Beim ersetzenden Fehlerverhalten wird das Signal durch das vorgegebene Signal ersetzt. Dies hat höhere Priorität als das additive oder modifizierende Verhalten. Alternativ zu den Einzelbeschreibungen kann auch eine allgemeine Transformation in Form einer Funktion angegeben werden.

Neben dem analogen ist auch die Definition von digitalem und zeitlichem Fehlerverhalten möglich.

3.2.3.4Definition des erwarteten Verhaltens

Grundlage für die automatisierte Auswertung der Tests ist eine Beschreibung des erwarteten Verhaltens des zu testenden Systems. Dies betrifft einen oder mehrere Ausgänge des Testsystems. Die Auswertung erfolgt über einzelne Tests, die miteinander logisch (AND, OR, NOT) verknüpft sind und so das Gesamttestresultat bilden. Die Verknüpfung kann beliebig hierarchisch gestaffelt werden.

Dabei wird immer für jeweils einen Ausgang ein zulässiges Signalverhalten spezifiziert. Jeder Test definiert ein binäres Testresultat auf Basis der erzielten Signalverläufe aus der Testumgebung.

Als Testmethoden stehen Schranken- und Bändertests zur Verfügung. Der Schrankentest definiert für das zu untersuchende Signal eine obere und eine untere Schranke, die vom Signal nicht verletzt werden dürfen. Es können jeweils eine oder auch beide Schranken definiert werden.

Der Test mit Bändern definiert eine Basisfunktion, um die ein Band (in Form einer Epsilonfunktion) gelegt wird. Das Signal darf sich nur innerhalb des Bandes aufhalten. Es wird zur Definition des Bandes eine Basis- und eine Epsilonfunktion erwartet.

Für die Definition der Auswertesignale wurde das Testfallformat um eine implizite, zeitunabhängige Signaldefinition erweitert. Das XML-Format ermöglicht nun eine Beschreibung von Zustandsautomaten bzw. getriggerten Tests. Es werden Trigger verwendet, die einen endlichen Automaten darstellen, der vom Startzustand nur über eine einzige Zustandsabfolge in seinen Folgezustand gelangt. Konkret bedeutet das, dass vom Trigger ein Signal ausgegeben wird, bis eine von ihm überwachte Bedingung erfüllt ist. Es tritt ein Zustandswechsel ein, indem die neue Signaldefinition ausgegeben wird. Um mehr Flexibilität zu erreichen, können Reihentrigger gebildet werden, indem mehrere Trigger hintereinander geschaltet werden. Damit wird eine feste Zustandsabfolge definiert. Tritt eine der überwachten Bedingungen nicht ein, kann der Endzustand nicht erreicht werden. Dies gilt auch, wenn die Bedingungen in der falschen Reihenfolge auftreten, und hat zur Folge, dass der Test fehlschlägt.

3.2.4 Testreport

Ein Testreport enthält jeweils das Resultat der Durchführung eines Testfalls. Der Testreport vermerkt allgemeine Informationen zum Test, wie beispielsweise, in welcher Version der Testfall durchgeführt wurde und wann dies geschah. Des Weiteren wird für das Testsystem mit seinen Komponenten vermerkt, welche Versionen zum Einsatz kamen. Dazu liefern die Komponenten eine definierte Selbstauskunft.

Der XML-Testreport enthält das Testresultat sowie die Einzelresultate aus der Durchführung der Tests, die im Testfall definiert sind. Zudem werden wieder formale Informationen (z. B. Datum) vermerkt. Der zugehörige Testfall und die Testkonfiguration werden über Verweise (Links) angebunden. Des Weiteren enthält der Testreport Versionsinformationen der einzelnen Testkomponenten und aller beteiligten Dateien, um eine Reproduzierbarkeit der Tests zu erzielen.

Der (XML-)Testreport wird automatisch für jeden Testfall, der von der Testablaufsteuerung durchgeführt wird, erzeugt. Bei Bedarf generiert die Testablaufsteuerung aus ihm und der Testkonfiguration sowie dem Testfall eine PDF-Version.

3.3 Testablaufsteuerung

Die Testablaufsteuerung ist das zentrale Element der automatisierten Testdurchführung. Nach dem Starten konfiguriert sie die Testumgebung, führt alle Testfälle einer Testsuite ohne weitere Eingriffe aus und stellt die Testergebnisse für die weitere Bearbeitung zur Verfügung. Dabei ist die Testablaufsteuerung flexibel gestaltet, um sich den Rahmenbedingungen in einem Unternehmen und den Erfordernissen aus der Anwendungsdomäne anzupassen. Diese Flexibilität wird durch ein Plugin-Konzept und ein Skript zur Steuerung des Testablaufs erreicht.

In den folgenden Unterabschnitten werden die Details der Testablaufsteuerung dargestellt. Dies beginnt mit der Vorstellung des Aufbaus, dem Konzept hinter den Plugins und damit, welche Plugins im Projekt realisiert wurden. Danach folgt ein kurzer Einblick in die Systemkonfiguration, um dann den durch das Skript definierten Testablauf zu erläutern.

3.3.1 Aufbau

Der Aufbau der Testumgebung lässt sich in drei Schichten aufteilen. Dies sind:

- die Prozessverwaltung,
- die Steuerungsschicht
- und die Testplattform (Testbed).

Abbildung 3-2 zeigt den prinzipiellen Aufbau der Testablaufsteuerung.

Die Prozessverwaltung enthält je eine Komponente für das Versionsmanagement beliebiger Dateien (VCS) sowie für das Testfall-Management (TCM). Das Versionsmanagement übernimmt die Versionierung aller im Rahmen der Tests verwendeten Dateien, um eine umfassende Reproduzierbarkeit zu gewährleisten. Das Testfallmanagement stellt den Zugang zu den einzelnen Tests und ihren Ergebnissen bereit. Es verwaltet alle zugehörigen Testfälle, -resultate und -reports in einer Datenbank. Damit lässt sich jederzeit nachvollziehen, welcher Anwender zu welchem Zeitpunkt was mit welchem Ergebnis getestet hat.

Die Steuerungsschicht beinhaltet sowohl die zentrale Komponente zur Testablaufsteuerung (TSC), als auch jeweils eine Komponente zur Auswertung, zur Testreporterstellung und zur Kopplung von Testplattform-Komponenten. Die Testablaufsteuerung ist das zentrale Element, an dem sich alle anderen Komponenten andocken müssen. Sie enthält ein anpassbares Python-Skript, welches den Ablauf der Testausführung definiert. Die Komponente zur Auswertung überprüft gemäß dem Analyse-Teil eines Testfalls die aufgezeichneten Daten während eines Testlaufs und ermittelt das Testresultat. Die Komponente zur Reporterstellung fasst alle Informationen zur Ausführung eines einzelnen Testfalls zusammen und generiert aus diesen

Informationen ein Dokument zur Dokumentation der Testausführung. Die Komponente der Kopplung spricht die für die Testausführung benötigten Komponenten der Testplattform an und verbindet diese miteinander, sofern mehr als eine Komponente in der Testplattform enthalten ist.

Die Testplattform ermöglicht z.B. die Anbindung einer Simulationsumgebung und einer zu testenden Software über entsprechende Adapter.

Die Komponenten aller Schichten werden durch einen Plugin-Mechanismus zur Laufzeit an die TSC angebunden. Für jede Komponente gibt es eine Standard-Implementierung für die für den minimalen Betrieb, welche in Abbildung 3-2 rot gekennzeichnet sind. Sämtliche Plugins lassen sich aber auch durch spezifische Eigenentwicklungen ersetzen, welche in Abbildung 3-2 blau gekennzeichnet sind.

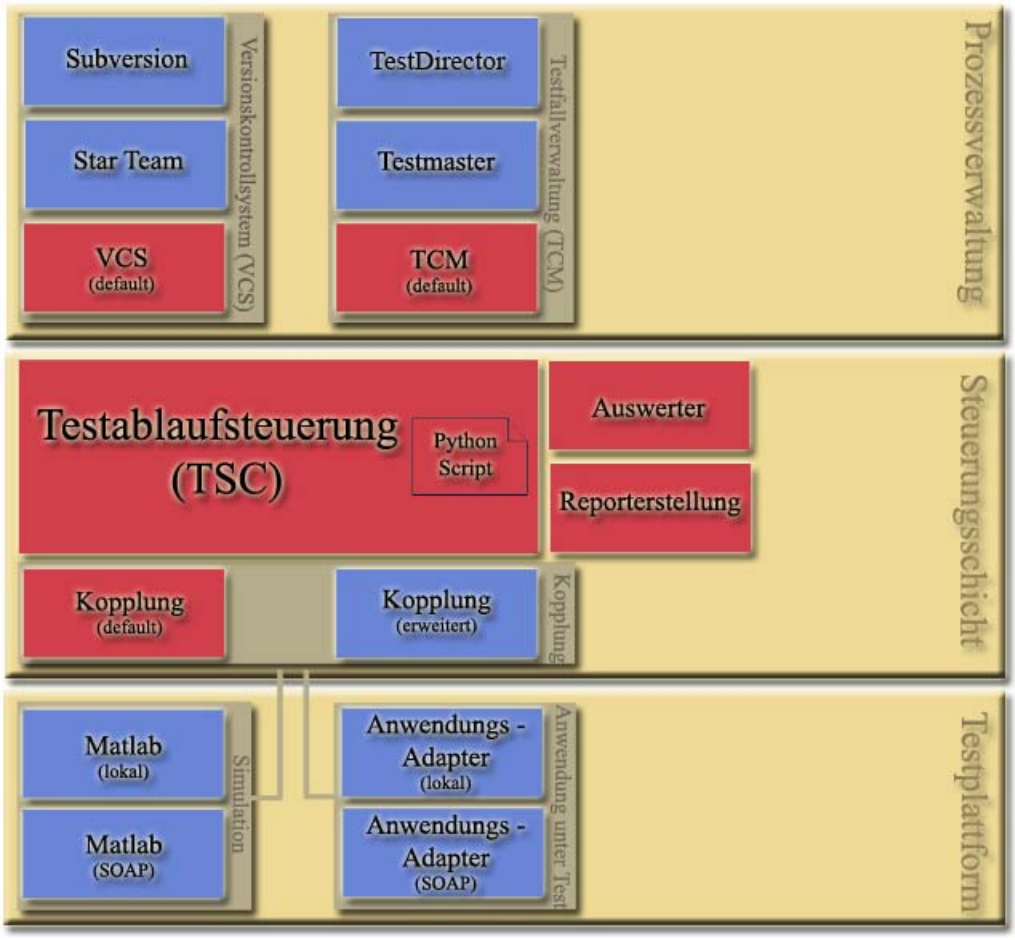

Abbildung 3-2 Struktur der Testablaufsteuerung

3.3.2 Plugin-Konzept

Alle Komponenten der Testablaufsteuerung werden als Plugin zur Laufzeit angebunden. Die Plugins müssen die für ihren Anwendungszweck üblichen Dienste der Testablaufsteuerung bereitstellen. Allen Plugins gemeinsam sind Dienste zum dynamischen An- und Abmelden von der Testablaufsteuerung sowie zur Konfiguration der Umgebung. Beispiele sind das Setzen des Arbeitsverzeichnisses oder von Parametern für das Plugin sowie die Abfrage von Laufzeitinformationen vom Plugin.

Für das Plugin zum Testfallmanagement existieren zusätzlich Dienste zur Abfrage von Testfällen und einer Testsuite. Des Weiteren muss dieses Plugin Dienste zum Melden des

Status eines Testfalls oder einer Testsuite sowie zur Mitteilung eines Testreports bereitstellen. Realisiert wurden drei Plugins für die Anbindung des Testmanagements, namentlich Plugins für TestDirector / Quality Center, Testmaster und ein lokales Dateisystem.

Ein Plugin zur Anbindung eines Versionsverwaltungssystems besitzt Methoden zum Setzen eines lokalen Directorys und eines Repositorys sowie zum Verwalten des lokalen Verzeichnisses und zum Abgleichen mit dem Repository. Realisiert wurde dieses Plugin für den Zugriff auf Subversion, StarTeam und ein lokales Dateisystem.

Die Plugins zur Anbindung von Systemkomponenten der Testplattform besitzen Dienste zum Initialisieren der Komponente, zum Lesen und Schreiben von Daten, zum Speichern und Laden von Zuständen sowie Methoden zum Starten, Stoppen und Auslesen von Ereignissen. Im Rahmen des Projekts wurden derartige Plugins zur Anbindung von Matlab sowie eines weiteren Rechnerboards realisiert, wobei die Plugins in zwei Realisierungen vorliegen. Eine Version dient dem lokalen Betrieb der Systemkomponente auf dem gleichen Rechner, auf dem auch die Testablaufsteuerung läuft. Die andere Version dient dem Betrieb auf einem anderen Rechnersystem über eine SOAP-Schnittstelle.

Zur Kopplung stehen zwei unterschiedliche Komponenten zur Verfügung. Beide Komponenten unterscheiden sich dadurch, dass die eine nur eine Systemkomponente an die TSC anbindet und die andere mehrere Systemkomponenten über das in Abschnitt 3.4 beschriebene Synchronisationsprotokoll koppelt. Die erste Komponente wird bei der Einbettung des Testobjekts in die Simulation verwendet, die zweite Komponente, falls das Testobjekt losgelöst von der Simulation betrieben wird. Die Komponenten zur Kopplung bieten Dienste zur Initialisierung, zum Starten und Stoppen des Testbeds und zum Setzen von Start- und Endzeit eines Simulationslaufs an. Des Weiteren kann ein Mapping zwischen den einzelnen Komponenten definiert werden.

Die Komponente Auswerter führt eine Auswertung eines Testlaufs durch. Hierzu besitzt die Komponente Dienste, um ihr alle wesentlichen Informationen zu einem Testlauf mitzuteilen, sowie den Dienst zur Durchführung der Auswertung eines Testlaufs und zum Erzeugen eines Testreports im XML-Format.

Die letzte an der Testablaufsteuerung beteiligte Komponente ist die zur Reporterstellung. Im Wesentlichen besitzt diese Komponente eine Schnittstelle zur Erzeugung eines Testreports in einem lesbaren Dokument aus dem Testreport im XML-Format, dem Testfall, der Testkonfiguration, der Systemkonfiguration sowie einer Liste von erzeugten Bildern, welche in den lesbaren Testreport eingebunden werden.

Jedes Plugin kann durch weitere Plugins ersetzt werden, welche die für ihren Zweck festgelegten Schnittstellen einhalten. Weitere Plugins zur Anpassung an eine Infrastruktur müssen in Java programmiert werden.

3.3.3 Systemkonfiguration

Die Aufgabe der Systemkonfiguration ist, die Konfiguration der Testablaufsteuerung zu beschreiben. Sie definiert beispielsweise die Plugins, welche für die Ausführung der aktuellen Testsuite verwendet werden sollen. Damit lässt sich eine Testsuite in verschiedenen Konfigurationen testen: Beispielsweise mit einem lokalen Versionsverwaltungssystem oder mit einer Anbindung an Subversion. Ebenso lässt sich zwischen einem lokal betriebenen Matlab oder einer Ausführung auf einem entfernten Rechner über eine SOAP-Schnittstelle umschalten.

Die Systemkonfiguration enthält Informationen über das Projekt, die einzusetzenden Komponenten der Testablaufsteuerung sowie die Services, welche genutzt werden

sollen. Die Informationen über das Projekt betreffen zum einen allgemeine Angaben wie Projektname und eine URL für weiterführende Informationen als auch den Namen des Kunden. Zum anderen werden die für die Testdurchführung verantwortlichen Personen mit ihren entsprechenden Rollen und Kontaktdaten festgelegt.

Die Informationen über die einzusetzenden Komponenten der Testablaufsteuerung enthalten den Namen des anzuwendenden Pythonskripts, die Angabe eines Archivnamens unter dem zur Laufzeit generierte Daten unter eine Versionsverwaltung gestellt werden können. Alle weiteren von der Testablaufsteuerung genutzten Komponenten werden beschrieben über den internen Namen, ihre Plugin-ID, Verbindungsinformationen für nutzbare Verbindungen, Dateiparameter und Pfadinformationen, Werteparameter und die für die einzelnen Komponenten Verweise auf die verantwortlichen Personen. Mit all diesen Informationen konfiguriert die Testablaufsteuerung nach ihrem Start die für die Testdurchführung einer Testsuite benötigten Komponenten. Auf diese Art und Weise ist es möglich, einzelne Testfälle sowohl in einer Umgebung für einen SiL-Test, als auch für alle anderen Testansätze zu verwenden, ohne einen Testfall anpassen zu müssen.

3.3.4 Testablauf

Der prinzipielle Testablauf ist in Abbildung 3-3 dargestellt. Der Testablauf wird durch ein Python-Skript in der Testablaufsteuerung festgelegt und lässt sich mit ausreichenden Kenntnissen über die Testablaufsteuerung und deren Plugins an die eigenen Bedürfnisse anpassen. Im Folgenden wird der im Projekt definierte Testablauf beschrieben.

Der Benutzer der Testumgebung weist die automatisch durchzuführenden Tests an und erhält nach ihrer Durchführung die Testresultate und -reporte angezeigt. Der Anstoß für die Abarbeitung von Testfällen kann aus der Testfallverwaltung heraus erfolgen. Der Speicherort für die Testfälle und die zugehörigen Resultate wird in der Systemkonfiguration festgelegt.

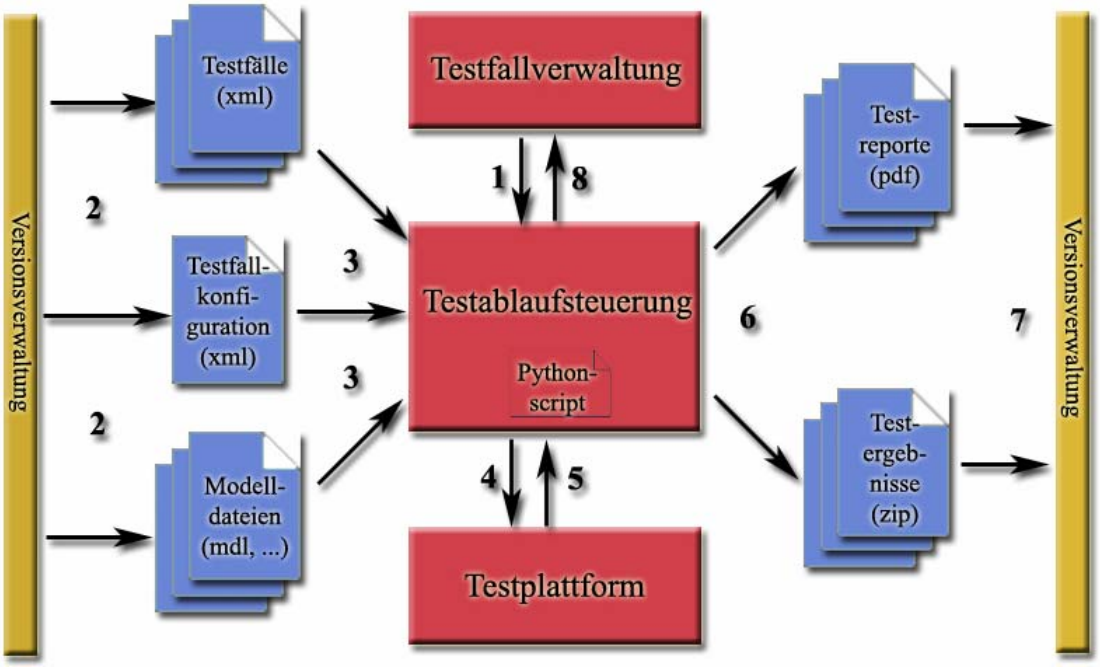

Abbildung 3-3 Prinzipieller Testablauf

Als Erstes holt sich die Testablaufsteuerung eine Liste mit den in einer Testsuite durchzuführenden Testfällen (1). Danach holt sich die Testablaufsteuerung gemäß den Startparametern die Testfall-Dateien, die Testkonfiguration und die Simulationsmodell-Dateien aus der Versionsverwaltung (2, 3) und übernimmt die automatische Abarbeitung der Testfälle auf der Testplattform, wofür sie die Testkonfiguration nutzt (4).

Anschließend ruft sie die Test- bzw. Simulations-Ergebnisse von der Testplattform ab (5) und erstellt für jeden Testfall einen XML-Testreport sowie Grafiken und das Testresultat. Über einen Konverter in der Komponente Reporterstellung werden die Testreporte und Grafiken in PDF-Testreporte gewandelt (6) und von der Testablaufsteuerung unter die Versionsverwaltung gestellt (7), wo die Ergebnisse aller Tests übersichtlich verwaltet werden und jederzeit verfügbar sind. Abschließend werden der Testfallverwaltung das Testergebnis und der Speicherort der Testreporte übermittelt (8).

3.4 Simulationskern / Betriebssystemanpassung

Zur Durchführung von SiL-Tests und PiL-Tests ist die zu testende Software mit einer Simulation der physikalischen Umgebung zu koppeln. Die Art der Kopplung kann auf unterschiedliche Arten erfolgen. Zum einen können die zu testende Modelle der Regelungsfunktionalität im SiL-Test oder der Softwaremodule im PiL-Test direkt in die Simulation der Umgebung eingebettet werden. Zum anderen können die Modelle und Softwaremodule separat von der Simulation der Umwelt betrieben werden. Der erste Ansatz wurde im Rahmen des SiLEST-Projekts für den Test des Motormanagementsystems verwendet, der zweite Ansatz zum Test des vollständig integrierten Lageregelungssystems. Der zweite Ansatz wird im Folgenden detaillierter betrachtet.

Die Kopplung zwischen der Simulation und der zu testenden Software in einer SiL-Simulation erfolgt nach dem Schichtenmodell im OS / HAL-Interface. Die Abbildung 3-4 zeigt das Schichtenmodell eines eingebetteten Systems mit seiner Umgebung. Von der Realisierung der Kopplung ist jedoch nicht nur das OS / HAL-Interface betroffen, sondern es werden auch die Schichten Controller Interface, Kommunikationsverbindung und Sensor- / Aktuator-Kommunikationsprotokoll berührt.

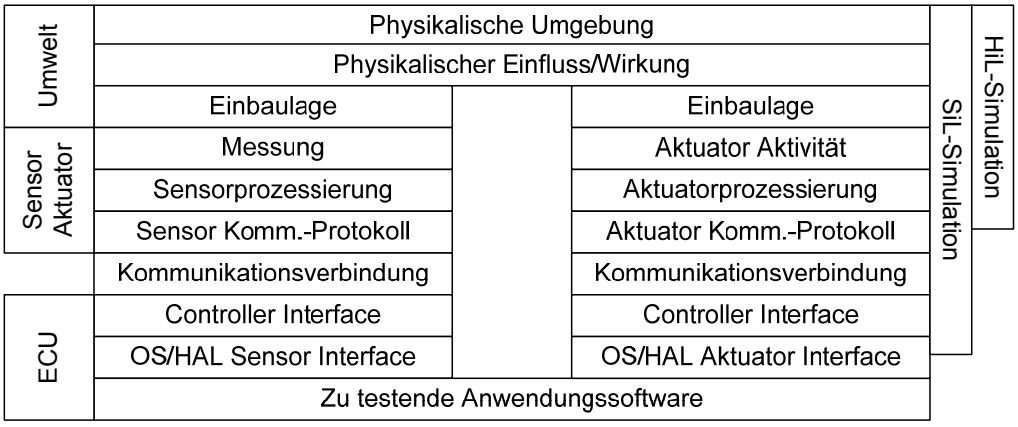

Abbildung 3-4 Schichtenaufbau eines Steuergeräts in seiner Umgebung

Abbildung 3-5 zeigt die von der Kopplung betroffenen Komponenten im Detail. Nicht betroffen von den Änderungen sind das Modellinterface und die Betriebssystem-Anwendungsschnittstelle (OS-API). Die Kopplung erfolgt durch das Ersetzen der Kommunikationsschnittstelle und der Interfacetreiber des Betriebssystems. Zur Kommunikation wird ein alternativer Kommunikationskanal genutzt. Zur Kompensation

des geänderten Zeitverhaltens müssen die Komponenten Scheduler und Timer der Simulation und der zu testenden Software ebenfalls zur Realisierung der Synchronisation zwischen Simulation und der zu testenden Software angepasst werden. Die Anpassungen zur Realisierung der Synchronisation werden weiter unten beschrieben.

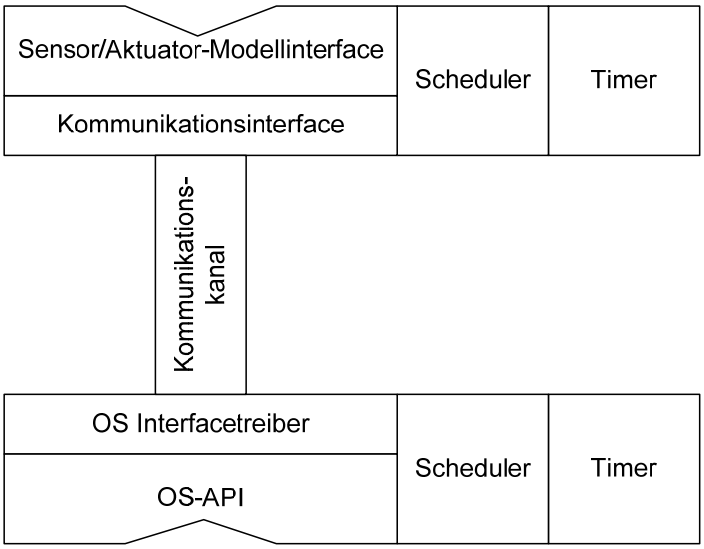

Abbildung 3-5 Kopplung zwischen Simulation und zu testender Software

Der simulationsseitige Scheduler wird dahingehend angepasst, dass er eine Synchronisation mit dem eingebetteten System durchführt. Das heißt, er tauscht die Informationen zur Synchronisation mit dem Scheduler des Steuergeräts aus, und beide führen eine Zeitanpassung durch. Die durch die Uhren angezeigten Zeiten entsprechen einer simulierten Zeit. Die Zeiten zur Synchronisation beider Systeme werden hierdurch für die Simulation und das zu testende Systeme unsichtbar. Dadurch entfällt die Forderung nach einer Echtzeitfähigkeit der Simulation. Das Testobjekt selber sieht die Zeit so, wie sie auch unter einer realen Umgebung sichtbar ist.

Der Synchronisation erfolgt nach dem Grundsatz, dass das Steuergerät die Zeit nur zum Zeitpunkt von Ein- und Ausgaben sieht. Die Zeit zwischen den Ein- und Ausgaben ist für die laufende Software nicht sichtbar. Unter Ein- und Ausgaben sind hier Aufrufe von Ein- und Ausgabefunktionen des Betriebssystems im Steuergerät oder das Senden von Nachrichten an das Steuergerät zu sehen. Intern wird dies dadurch umgesetzt, dass nach jeder Kommunikation mit der Umgebungssimulation im Steuergerät die Uhren gestellt werden. Diese Anpassung entspricht der Zeit, die verstreicht, wenn eine Kommunikationsfunktion über die unveränderte Schnittstelle aufgerufen wird.

Bei der Durchführung eines Testes arbeiten Steuergerät und Simulation abwechselnd in Zeitscheiben. Dabei sind zwei Zyklen zu unterscheiden, der Minorzyklus und der Majorzyklus. Der Minorzyklus stellt eine ununterbrechbare Zeiteinheit da, in der die Simulation arbeitet. Innerhalb des Minorzyklus kann keine Kommunikation zwischen Simulation und dem Steuergerät auftreten. Der Minorzyklus bestimmt unter anderem auch die Genauigkeit der Simulation, da die Simulation mit der Schrittweite des Minorzyklus ausgeführt wird. Der Majorzyklus stellt die Länge der Zeitscheibe dar und kann durch eine Kommunikation unterbrochen werden.

Von besonderem Interesse für die Synchronisation von Simulation und Steuergerät sind Kommunikationsereignisse zwischen dem Steuergerät und seiner Umgebung. Diese Kommunikationen können im Steuergerät durch Ein- oder Ausgabeoperationen, oder Ereignisse in der Simulation sein, welche eine Unterbrechung im Steuergerät auslösen.

Hinsichtlich des Auftretens von Ereignissen in einen Majorzyklus sind insgesamt vier Fälle zu unterscheiden.

- Es treten keine Ereignisse im Steuergerät und der Simulation auf.
- Es tritt ein I/O-Ereignis im Steuergerät und kein Ereignis in der Simulation auf.
- Es tritt kein Ereignis im Steuergerät und ein Ereignis in der Simulation auf.
- Es tritt ein Ereignis im Steuergerät und ein Ereignis in der Simulation auf.

Die vierte Möglichkeit ist dabei vom Protokoll her nur ein Sonderfall der zweiten Möglichkeit, bei der die berechnete Vorlaufzeit durch das Steuergerät nicht abgearbeitet wird, weil bereits vorher ein Ereignis im Steuergerät auftritt. Aus diesem Grund wird dieser Fall nicht gesondert durch das Synchronisationsprotokoll behandelt.

Im ersten Fall laufen Simulation und Steuergerätesoftware komplett ohne Ereignis einen Majorzyklus durch. Die zwischen Simulation und Steuergerät auszutauschenden Nachrichten beschränken sich auf eine run- und eine sync-Nachricht zum Anfang und Ende des Majorzyklus. Abbildung 3-6 zeigt das Sequenzdiagramm für die Abarbeitung des Majorzyklus.

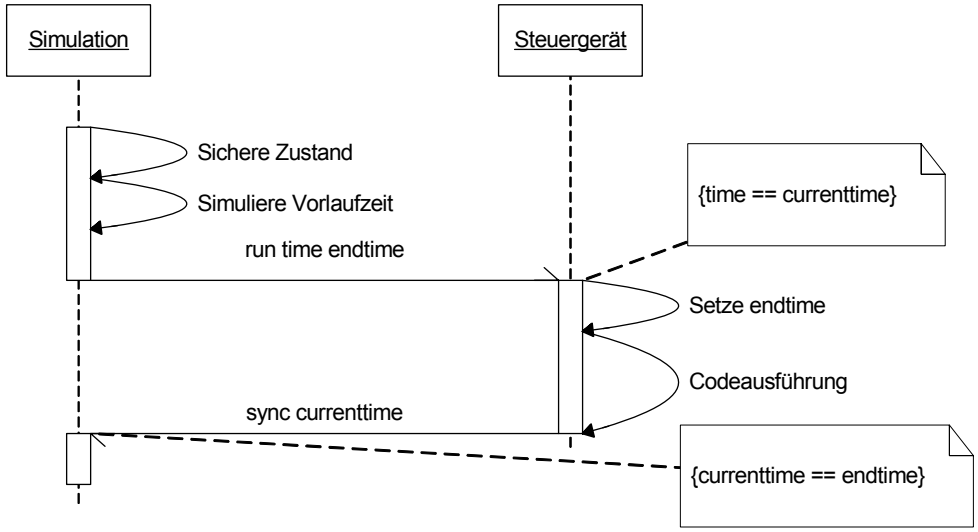

Abbildung 3-6 Sequenz ohne Ereignisse

Im zweiten Fall tritt im Gegensatz zum ersten Fall während der Berechnung des Majorzyklus in der Simulation ein Ereignis, welches dem Steuergerät durch eine Kommunikation mitgeteilt wird. Diese Kommunikation kann durch einen Interrupt oder aber durch das Senden einer Nachricht auf einem Kommunikationsbus erfolgen. In diesem Fall führt nach der Berechnung des Majorzyklus das Steuergerät keinen kompletten Majorzyklus durch, sondern wird zum Zeitpunkt der Kommunikation angehalten. Nach dem Eintreffen einer sync-Nachricht vom Steuergerät wird diesem die Kommunikation übermittelt und es darf anschließend mit den Berechnungen des Majorzyklus weiter fortfahren Abbildung 3-7 zeigt das Sequenzdiagramm für diesen Fall.

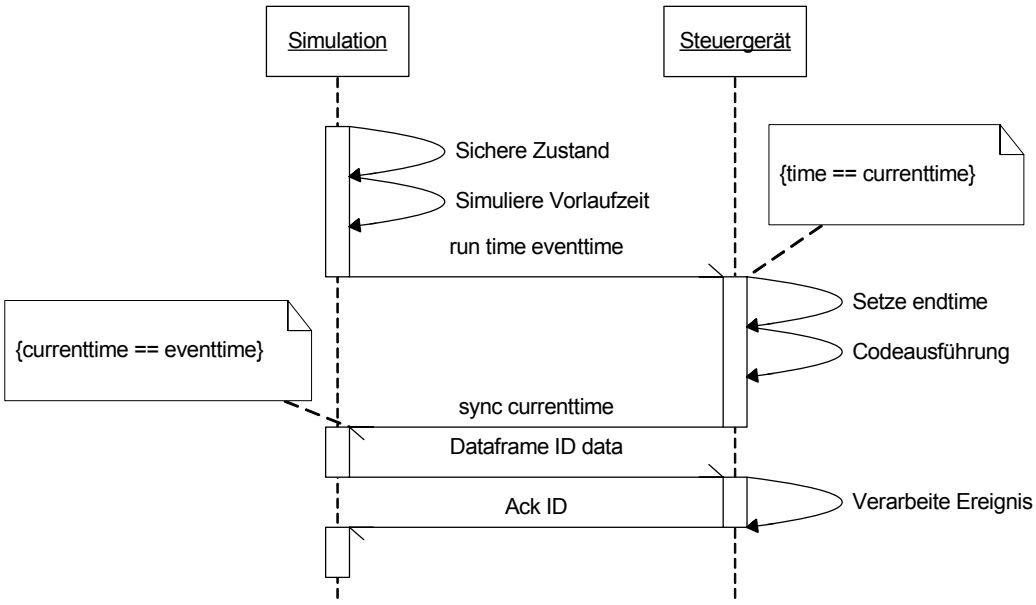

Abbildung 3-7 Sequenz mit einem Ereignis in der Simulation

Beim letzten zu berücksichtigenden Fall tritt im Steuergerät vor Ablauf für das Steuergerät vorgesehenen Rechenzeit ein Ein- / Ausgabeereignis auf. Der Ablauf unterscheidet sich bis zum Eintritt des Ein- / Ausgabeereignisses im Steuergerät nicht von den beiden bereits vorgestellten Fällen. In diesem Fall teilt das Steuergerät durch eine event-Nachricht anstelle einer sync-Nachricht die vorzeitige Berechnung mit. Die Simulation muss sich nach dem Empfang dieser Nachricht den Zustand der Umgebung zum Zeitpunkt dieses Ereignisses ermitteln. Danach erfolgt der Datenaustausch für das Lese- oder Schreibereignis zwischen Steuergerät und Simulation, welcher mit einer sync-Nachricht abgeschlossen wird. Abbildung 3-8 und Abbildung 3-9 zeigen die Sequenzdiagramme für Aus- und Eingabeereignisse im Steuergerät.

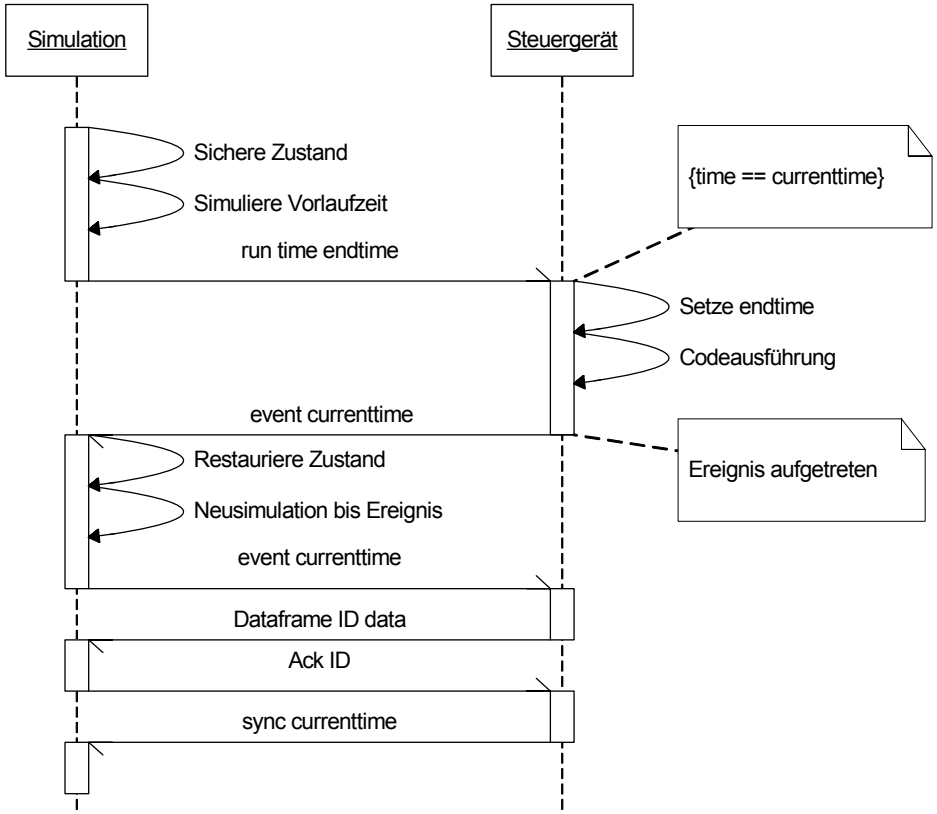

Abbildung 3-8 Sequenz mit einem schreibenden Ereignis im Steuergerät

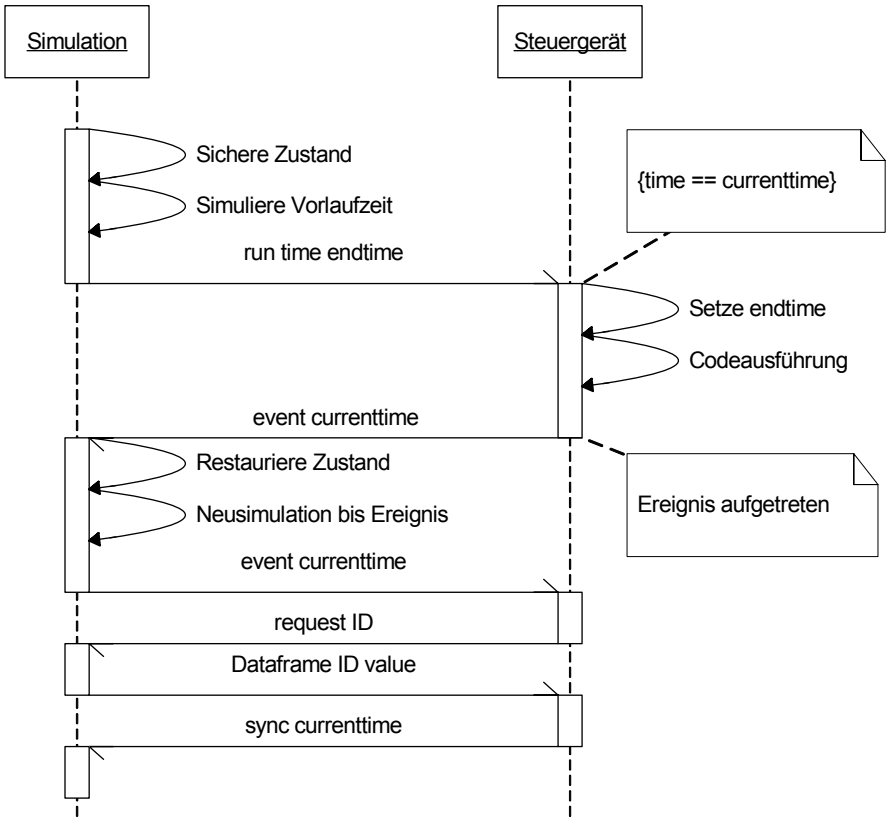

Abbildung 3-9 Sequenz mit einem lesenden Ereignis im Steuergerät

3.5 SALib

Die Testfälle des Software-in-the-Loop-Tests (SiL) untersuchen einerseits das Nominalverhalten der Software für gängige Eingangsverläufe und andererseits das Softwareverhalten bei Fehlern. Eingebettete elektronische Systeme und insbesondere Steuergeräte in Kraftfahrzeugen sind dadurch gekennzeichnet, dass sie über Sensoren und Aktuatoren mit ihrer Umwelt in Verbindung stehen. Von allen möglichen Fehlern, Anomalien, Beeinflussungen und Ausfällen konzentriert sich SiLEST auf die Analyse vom nominalen und fehlerhaften Verhalten des Steuergerätecodes infolge fehlerhafter Sensoren und Aktuatoren.

SALib (Sensors and Actuators Library) ist eine zur Simulation von Sensoren und Aktuatoren entwickelte Simulink-Modellbibliothek, die im Rahmen dieses Projekts entstanden ist. Die Modelle bilden verschiedene Fehlerfälle wie Kabelbruch, Kurzschluss gegen Masse, additives Messrauschen oder auch Alterungseffekte nach. Die wesentliche Eigenschaft dieser Modelle in Bezug auf den automatischen SiL-Testablauf besteht in der Möglichkeit der Fehlerinjektion mit entsprechenden Fehlerparametern und zu ausgewiesenen Zeitpunkten während der Simulationslaufzeit. Im Folgenden werden Aufbau und grundlegende Funktionalitäten der Bibliothek beschrieben.

3.5.1 Bibliotheksaufbau

Die Bibliothekshierarchie enthält drei Hauptgruppen SENSORS, ACTORS und FAULT MODEL, die je nach vorhandenen Sensoren und Aktuatoren in weitere Untergruppen eingeteilt werden (Abbildung 3-10). Es wird erwartet, dass die Simulink-Bibliothek stetig wachsen wird, so dass an dieser Stelle nur die jeweils aktuelle Hierarchie wiedergegeben werden kann.

Die Fehlermodelle aus der Untergruppe FAULT MODEL stellen den expliziten Teil aus den Sensor- und Aktuatormodellen dar und dienen der Nachbildung deren Fehlerverhaltens. Darüber hinaus können sie aber auch zur Modellierung von diversen Signalquellen oder Effekten außerhalb von Sensoren und Aktuatoren herangezogen werden.

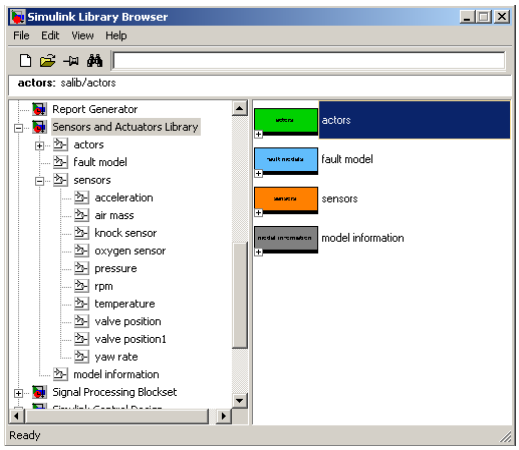

Abbildung 3-10 Simulink Bibliothek SALib

Blocktypen

Jeder Sensor oder Aktuator wird mittels eines allgemeinen Blocktyps ausgeführt. Spezielle Sensoren und Aktuatoren lassen sich durch entsprechende Kalibration dieser erhalten. Es können also verschiedene Sensoren mit dem gleichen Blocktyp realisiert werden (z. B. analoge Beschleunigungs- oder Drucksensoren) sowie ein und derselbe

Sensor vom verschiedenen Typ sein (z. B. analoger oder digitaler Temperatursensor). Aktuell sind in der Bibliothek die Blocktypen ANALOGSENSOR und DIGITALSENSOR vorhanden:

In Simulink ist ein Sensor oder Aktuator als ein maskiertes Subsystem realisiert, um lokale, strukturinterne Variablen definieren zu können Der Maskenname eines Blocks gibt den jeweiligen Blocktyp an. Der Blockname identifiziert die eigentliche Bezeichnung für die jeweilige Komponente, z. B. SHT7X.

Versionierung einzelner Blöcke

Simulink bietet zwar standardmäßig Versionierung einer Modelldatei (dazu gehört auch eine Library-MDL-Datei), allerdings gilt das nicht für die einzelnen Blöcke innerhalb einer Bibliothek. Um die Versionierung der letzteren dennoch zu ermöglichen, werden die Blocktyp-Modelldateien, die die Struktur der Blöcke beinhaltet, ausgelagert. Nach einer Änderung dieser Modelldatei wird sie standardmäßig versioniert, und gleichzeitig werden alle darauf basierenden zugehörigen Blöcke in der Bibliothek automatisch aktualisiert.

3.5.2 Funktionalitäten

Im Folgenden werden die grundlegenden Funktionalitäten der Sensor- und Aktuatormodelle dargestellt. Die Ausführungen sollen am Beispiel eines analogen Sensors erfolgen und behalten ihre Gültigkeit auch für einen analogen Aktuator.

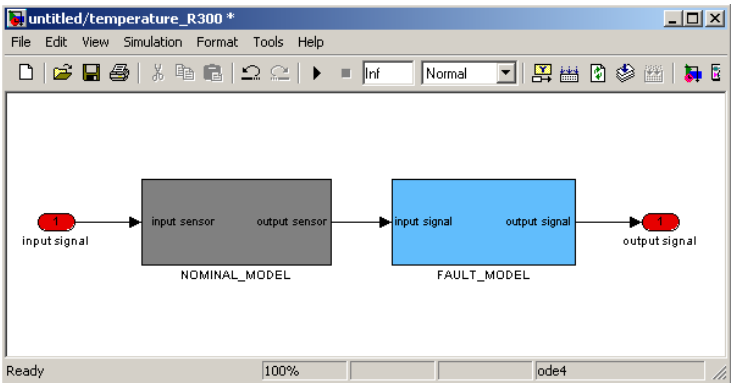

Abbildung 3-11 Analoges Sensormodell

3.5.2.1Modellierung des Nominalverhaltens

Das Modell eines analogen Sensors besteht aus den Teilmodellen, die das nominale und Fehlerverhalten eines Sensors nachbilden sollen (Abbildung 3-11).

Das nominale Verhalten eines Sensors ist im Allgemeinen durch das statische und das dynamische Verhalten gekennzeichnet. In den entsprechenden Simulink-Modellen lassen sich die typischen Sensorkenngrößen wie Empfindlichkeit, Messbereichsgrenzen, Grenzfrequenzen u.a. variabel parametrieren. Jeder Sensor kann in unterschiedlichen Varianten, beispielsweise in unterschiedlichen Empfindlichkeiten erhältlich sein. Erst eine Kalibrierung mit gemessenen bzw. vom Hersteller mitgelieferten Kennlinien und Kenngrößen zusammen mit einem Simulationscode ergibt einen bestimmten Sensor. Die Modellvariablen, die diese Kenngrößen enthalten, werden als lokale Variablen ausgeführt. Damit wird sichergestellt, dass es beim mehrfachen Einsatz ein und desselben Sensors in der Simulation zu keinem Namenskonflikt der Variablen kommt.

3.5.2.2Modellierung des Fehlerverhaltens

Die Fehlerarten von Sensoren lassen sich im Allgemeinen in folgende Kategorien einteilen: Elektrische Fehler, Fehlfunktionen der Sensoren selbst, Bordnetzstörungen, Alterung und Temperatureinfluss von Baugruppen und Signalstörungen. Bei der Modellierung der Fehlerfälle wurden je nach Wirkungsweise drei Fehlermodelltypen unterschieden, nämlich REPLACE, ADD und TRANSFORM. In der Tabelle 1 sind die Bedeutungen der drei Modelltyp-Bezeichnungen kurz erläutert.

Fehlermodell	**Bedeutung**
REPLACE	Das Fehlermodell ersetzt das Nominalmodell vollständig
ADD	Das Fehlersignal wird auf das Nominalsignal aufaddiert
TRANSFORM	Das Ausgangssignal des Nominalmodells wird durch das Fehlermodell transformiert

Tabelle 1 Fehlermodelltypen

Jeder der Fehlertypen erhält eine Identifikationsnummer, die bei der Fehlerinjektion während der Simulation benötigt wird. Die Fehlerparameter von jedem der einzelnen Fehlerfälle, z. B. der Faktor für einen Steigungsfehler oder die Amplitude eines additiven Messrauschens, werden zur Simulationslaufzeit an die Modellvariablen übergeben.

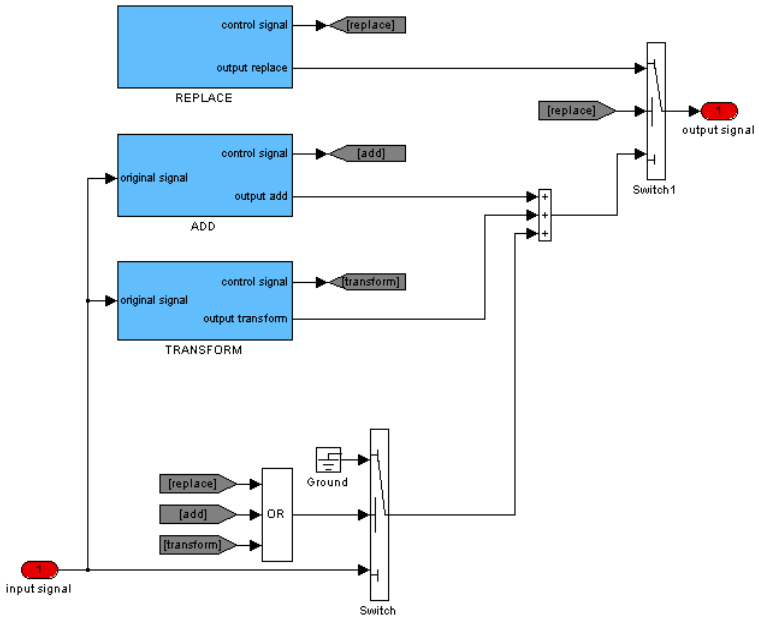

Abbildung 3-12 Gesamtstruktur des Fehlermodells

Des Weiteren wird jedes Fehlermodell durch geeignete Steuersignale aktiviert bzw. deaktiviert, so dass in einem Sensor bestimmte Fehler mit bestimmten Kenngrößen nur während eines ausgewiesenen Zeitintervalls auftreten können. Im Abschnitt 3.4.3.3 wird auf den Vorgang der Fehlerinjektion näher eingegangen. Die Abbildung 3-12 zeigt die Gesamtstruktur des implementierten Fehlermodells. In jedem der Fehlermodelltypen befinden sich weitere untergeordnete Fehlermodelle, die auf die eine oder andere Weise verknüpft sind:

REPLACE

- Signalgenerator (vgl. unten)

ADD

- Signalgenerator (vgl. unten)

TRANSFORM

- **DEAD ZONE** modelliert Totzeitverhalten
- **HYSTERESIS** modelliert Hysterese-Eigenschaften eines Sensors
- **GAIN** bewirkt Verstärkung bzw. Dämpfung des Sensorsignals
- **MATLAB-FCN** ermöglicht beliebige Transformationen mit Hilfe Matlab-eigener Funktionen

Der Signalgenerator stellt für die Realisierung verschiedener Fehlerarten einen Satz diverser Signalformen zur Verfügung, die in Tabelle 2 beschrieben sind:

CONST	Gleichsignal
RAMP	steigende bzw. fallende Rampe
SINE	Sinussignal variabler Amplitude, Frequenz und Phase
NOISE	Rauschen mit variabler Varianz
TRIANGLE	Dreiecksignal variabler Amplitude, Frequenz und Phase
SAWTOOTH	Sägezahnsignal variabler Amplitude, Frequenz und Phase
RECTANGLE	Rechtecksignal variabler Amplitude, Frequenz und Phase
SEQUENCE	Benutzerdefinierter Signalvektor

Tabelle 2 Signalformen des im Fehlermodell verwendeten Signalgenerators

Zu einem bestimmten Zeitpunkt lässt sich immer nur eine Signalform aufrufen. Durch entsprechende Parametersetzung lässt sich optional nur die positive Halbwelle eines Signals ausgeben bzw. ein Signal nur zu definierten Pulsfolgen generieren. In folgender Tabelle 3 sind einige Beispiele für die Modellierung diverser realer Fehlerarten mit den implementierten Fehlermodellen gegeben.

Fehlerfall	Fehlermodell
Kabelbruch	REPLACE / CONST
Kurzschluss gegen Masse bzw. Versorgung	REPLACE / CONST
Masseversatz	ADD / CONST
Übergangswiderstand	TRANSFORM / GAIN
Blockieren	REPLACE / CONST
Reibung	TRANSFORM / DEAD ZONE
Hysterese-Effekt	TRANSFORM / HYSTERESIS
Additives Rauschen	ADD / NOISE
Spannungseinbruch	ADD / SEQUENCE
Spannungsschwankung	ADD / SINE

Tabelle 3 Beispiele für die Modellierung diverser Fehlerarten

3.5.2.3Fehlerinjektion

Nachdem ein oder mehrere Sensoren aus der SALib-Bibliothek in das Umgebungsmodell eingebunden wurden, müssen im Vorfeld der Simulation die sog. Fehlermatrizen definiert und geladen werden. Die Decodier-Bausteine der Sensormodelle erzeugen schließlich daraus Steuersignale zur Aktivierung bzw. Deaktivierung der entsprechenden Fehlermodelle und weisen diesen die vorgegebenen Parameter zu. Das Konzept der Fehlerinjektion ist in der Abbildung 3-13 dargestellt.

Fehlermatrix

Für den automatischen SiL-Test wird aus dem jeweiligen Testfall eine Fehlermatrix abgeleitet. Diese Matrix ist auf eine fest vorgeschriebene Weise aufgebaut und beinhaltet in jeder Zeile den Fehlertyp, die Identifikationsnummer des Fehlerfalls, den Start- und Endzeitpunkt der Aktivierung des entsprechenden Fehlermodells und die entsprechenden Fehlerparameter. Die notwendigen Daten für die Parametrierung der Fehlermodelle sind in einer Fehlermatrix zusammengefasst.

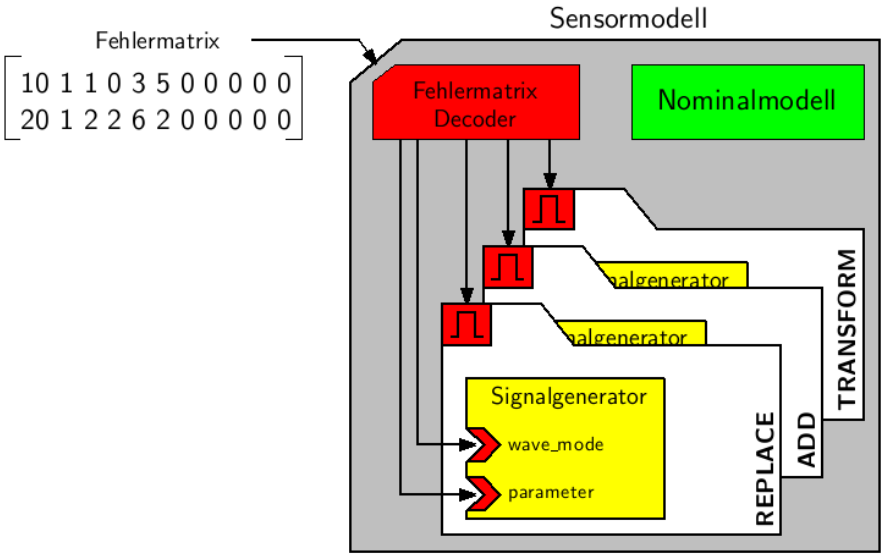

Abbildung 3-13 Prinzip der Fehlerinjektion

Aktivierung der Fehlerfälle

Nachdem das Sensormodell in das Simulationsmodell integriert und alle Dialogvariablen gesetzt wurden, kann das Simulationsmodell gestartet werden. Folgende Vorgänge laufen beim Starten der Simulation ab:

- Zu Beginn werden die Parameter des Nominalmodells geladen.
- Der Decoder entschlüsselt danach die Fehlermatrix und generiert entsprechende Steuer- und Parametersignale.
- Die Steuersignale fungieren dabei als Triggersignale für das Nominal- und die Fehlermodelle.

Die Decodier-Funktion für die Fehlermatrix wird als eine S-Function realisiert. Zur Integration in die Solversoftware, der Simulink-Engine, werden in einer S-Function verschiedene Schnittstellen mit genau definierten Bezeichnungen zur Verfügung gestellt. Im konkreten Falle einer C-Mex-Funktion definiert der Anwender eine Prozedur mit dem entsprechenden Namen (z. B. mdlOutput) und setzt dort den jeweiligen Programmcode ein. Der Decoder hat als Eingangsvariablen die Fehlermatrix und die Nummer des aktuellen Fehlertyps. Die Ausgangssignale entsprechen den Einträgen in der Fehlermatrix unter der Bedingung, dass der vorgegebene Fehlertyp zum gegebenen aktuellen Zeitpunkt aktiv ist.

Während der Simulation können die Sensormodelle folgende Zustände annehmen:

- Wenn kein Fehlerfall angefordert wird, werden alle Fehlermodelle deaktiviert.
- Wenn ein Fehler vom Typ REPLACE angefordert wird, werden alle anderen Fehlermodelle und das Nominalmodell deaktiviert (höchste Priorität).
- Alle Fehlermodelle vom Typ ADD und TRANSFORM können gleichzeitig auftreten.

Die Abbildung 3-14 zeigt ein Beispiel für das Auftreten eines doppelten Fehlers: Der Sensor bewirkt ein additives Rauschen und ein kurzzeitiges Unterbrechen der Signalmessung (z. B. aufgrund eines Kabelbruchs).

Für den automatischen SiL-Test des Steuergerätecodes lässt sich somit aus dieser Simulink-Bibliothek für die jeweiligen Steuergeräte- und Umgebungsmodelle ein Satz geeigneter Sensor- und Aktuator-Bausteine auswählen, die während der Simulationslaufzeit diverse Fehler generieren. Die eigens entwickelte Bibliothek kann aber auch als eigenständiges Tool für Modellierungsaufgaben von verschiedenartigen Signaleffekten eingesetzt und erweitert werden.

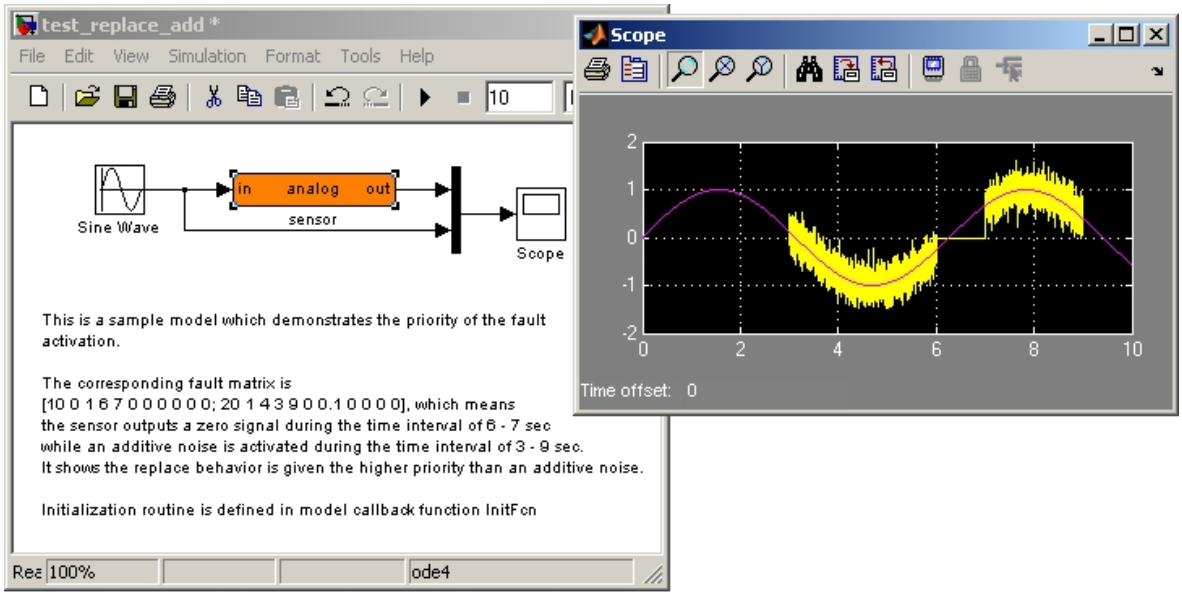

Abbildung 3-14 Modellierungsbeispiel für einen fehlerbehafteten Sensor

3.6 Unterstützungswerkzeuge / Testfalleditor

Der Testingenieur hat die Aufgabe, Testfälle zu erstellen und zu prüfen. Um eine hinreichende Testabdeckung zu erreichen, ist es des Öfteren notwendig, ganze Testfall-Suiten aufzubauen. Bei dieser Arbeit wird der Testingenieur durch einen im SiLEST-Projekt entwickelten Testfalleditor unterstützt.

Der Testfalleditor vereinfacht die Abwicklung der Testfallerstellung und -pflege, indem er die Bearbeitung der Konfigurationsdateien und der Testfalldaten übernimmt. Er ermöglicht über eine grafische Oberfläche die Eingabe verschiedener Testfallparameter und validiert die eingegebenen Daten gemäß dem XML-Schema für den entsprechenden Datentyp. Er stellt sicher, dass der festgelegte Namensraum für den Testfall in der Konfiguration nicht verletzt wird und erleichtert durch Vorschläge für Portnamen die Eingabe. Hierdurch wird eine hohe Wiederverwendbarkeit der Testfälle erreicht. In Abbildung 3-15 ist die grafische Oberfläche mit der kontextsensitiven Hilfe bei der Auswahl eines Portnamens im gegebenen Namensraum zu sehen.

Alle Daten werden in Form von XML-Dateien durch den Testfalleditor gespeichert, welche für die Testautomatisierung im Rahmen des SiLEST-Testprozesses Gebrauch finden. Eine Sammlung von Testfällen in einer Testsuite benötigt eine Testfallkonfiguration, welche ebenfalls durch den Testfalleditor erstellt wird.

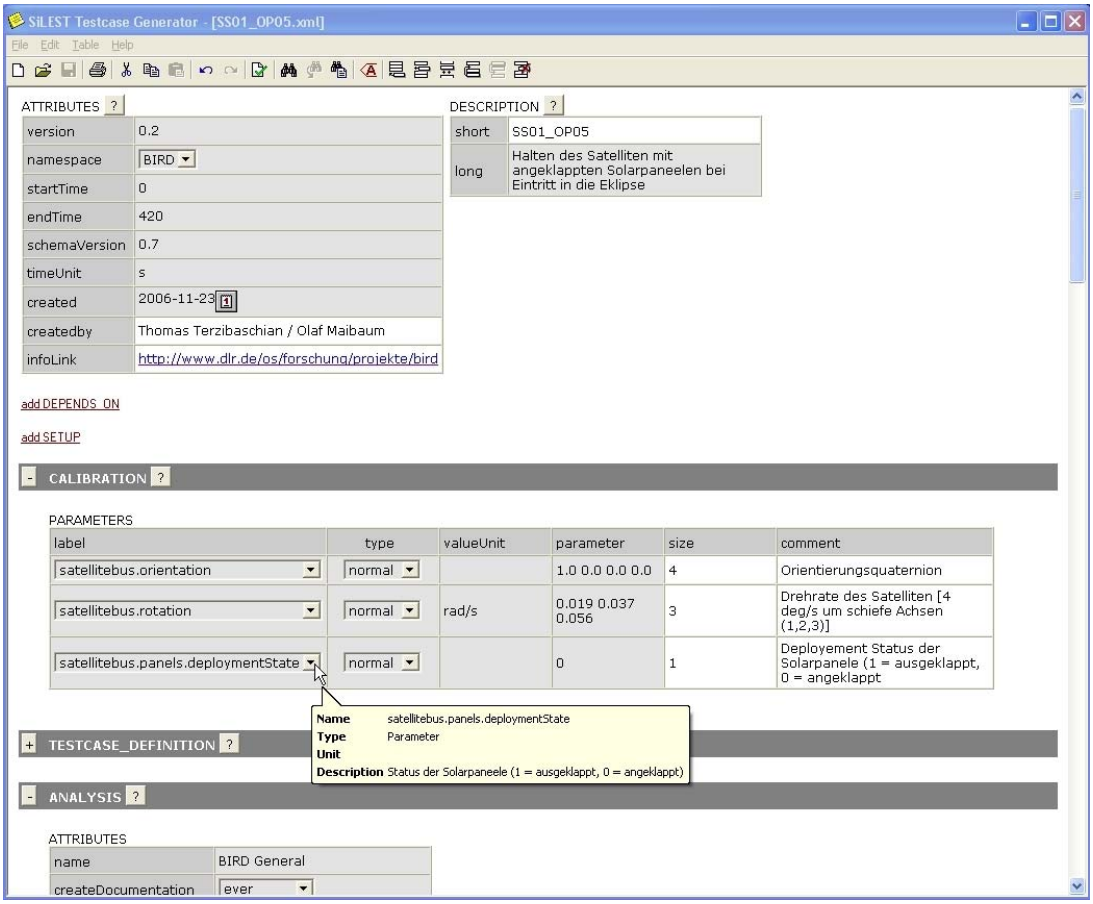

Abbildung 3-15 Testfalleditor

Die XML-Technologie ermöglicht ein flexibles und erweiterbares Arbeiten am Testfalleditor. Hierdurch wird eine Anpassbarkeit der verwendeten Formate an die für den Einsatz notwendigen Randbedingungen erreicht, welche sich für unterschiedliche Anwendungsdomänen unterscheiden.

Der Testfalleditor ist als .NET-Anwendung realisiert und verwendet als Basis das öffentlich nutzbaren Authentic Browser Plugin von Altova INC[4]. Zusätzlich kommen die Komponenten VXPLib[5] und NiceLine[6] zum Einsatz. Der Testfalleditor ist sehr flexibel gestaltet und lässt sich durch Plugins erweitern. So lassen sich beispielsweise anwendungsbezogene Editoren für die grafische Eingabe von Werteverläufen oder spezielle Werkzeuge zum Einlesen von Daten in den Editor integrieren. Des Weiteren besitzt der Editor eine kontextsensitive Hilfe, welche Informationen zu den Eingabefeldern des Testfallformats bereitstellt.

Die implementierten Funktionalitäten des Testfalleditors noch einmal im Überblick:

- Anlegen, Editieren und Speichern von Testfällen
- Anlegen, Editieren und Speichern von Konfigurationsdateien
- Historie der bereits erstellten Dateien

[4] www.altova.com

[5] www.tooltips.net

[6] www.codeproject.com

- Darstellung des XML-Formats in einer übersichtlichen Ansicht
- Einfaches Hinzufügen von zusammengesetzten Elementen
- Einfache Eingabe von Werten mit Formatvalidierung auf Basis des XML-Schemas
- Ein- und Ausblenden von einzelnen Abschnitten im Dokument
- Undo- / Redo-Funktionalität
- Drucken
- Allgemeine Hilfe
- Kontextsensitive Hilfe
- Plugin-Schnittstelle für erweiterte Eingabe

4 Testdurchführung

4.1 DLR

Das DLR nutzte im Rahmen des SiLEST-Projekts als Anwendungsszenario das Lageregelungssystem des Kleinsatelliten Bird. Es sollte durch die Durchführung der Tests und den Vergleich mit aufgezeichnetem Datenmaterial aus dem Betrieb des Kleinsatelliten die Eignung für den Test von Onboard-Regelungssoftware nachgewiesen werden.

4.1.1 Testfälle

Zur Definition der Testfälle wurden drei Nutzszenarien definiert, welche für den Satellitenstart und den operativen Betrieb regelmäßig auftreten. Diese sind im Einzelnen

- Booten eines rotierenden Satelliten
- Halten des Satelliten im Safemode
- Neuorientieren des Satelliten

Diese Nutzszenarien wurden mit der Konfiguration an- und ausgeklappter Solarpaneelen gemischt, wobei die erste Konfiguration nur in der Inbetriebnahmephase des Satelliten auftritt und daher ein besonders hohes Risiko darstellt.

Die Nutzszenarien sollten einmal mit störfreier und gestörter Sensorik und Aktuatorik ausgeführt werden. Als kritische Komponenten werden die Sonnensensoren, der Magnetfeldsensor, das Lasergyroskop und die Reaktionsräder angesehen. Diese Komponenten würden auch im Rahmen einer FMEA als kritisch eingestuft werden.

Für die Sonnensensoren werden die folgenden Störungen betrachtet, wobei einige im nominalen Betrieb des Satelliten regelmäßig auftreten.

- Verdunklung von Sensorplättchen (Eklipse)
- Unterschiedliche Ausrichtung von Sensorplättchen
- Rauschen der Sensoren
- Einfluss von Streulicht (Albedo)

Für die Magnetfeldsensoren werden die folgenden Störungen betrachtet, wobei einige auch im nominalen Betrieb des Satelliten auftreten können.

- Externes magnetisches Ereignis
- Rauschen des Sensors

Für das Lasergyroskop werden die folgenden Störungen betrachtet, wobei einige auch im nominalen Betrieb des Satelliten regelmäßig auftreten können.

- Verlust einer Nachricht
- Datenverfälschung in einer Nachricht
- Rauschen der Sensorwerte
- Stark springende oder gleich bleibende Sensorwerte

Für die Reaktionsräder werden folgende Störungen betrachtet, wobei einige auch im nominalen Betrieb des Satelliten selten auftreten können.

- Ein Reaktionsrad ist defekt und ausgeschaltet

- Verlust von Nachrichten
- Ein Rad klemmt

Aus den Szenarien und den Störfällen wurden insgesamt 60 Testfälle definiert mit störfreiem Betrieb, einem Gerät gestört und zwei Geräten gestört.

4.1.2 Aufbau der Testumgebung

Der prinzipielle Aufbau der SiLEST-Testumgebung wird in Abschnitt 3.1 beschrieben. An dieser Stelle wird die Konfiguration beschrieben, wie sie beim DLR zum Einsatz gekommen ist.

Als Testmanagement-Werkzeug wird das als Open Source verfügbare Werkzeug Testmaster verwendet. Zur Versionsverwaltung kommt Subversion zum Einsatz. Die zu testende Lageregelungssoftware kann auf einem Linux-System oder einem Prozessorboard, welches ähnlich dem im Satelliten verwendeten Hauptcomputer ist, ausgeführt werden. Die Lageregelungssoftware wird über das in Abschnitt 3.4 beschriebenen Protokoll mit der Simulation der Satellitenumgebung gekoppelt. Die Kopplung erfolgt über ein Plugin der TSC. Der prinzipielle Aufbau der verwendeten Simulation der Satellitenumgebung wird weiter unten beschrieben.

4.1.2.1Testmaster

Die Web-Anwendung Testmaster dient zur Testfall-Verwaltung und erlaubt die Zuordnung mehrerer Testfälle zu Testsuiten, Projekten und Abteilungen. Integriert ist auch ein rollenbasiertes Rechtekonzept für Nutzer.

Es wurde ein Projekt "BIRD" im Testmaster angelegt, welches zwei Testsuiten "finaltests" und "testingsuite" enthält. Letztere enthält Testfälle, die sich nur auf das Simulationsmodell beziehen, das Prozessorboard unberücksichtigt lassen und damit nur zu Testzwecken der Testumgebung dienen. Die Testsuite "finaltests" enthält die oben beschriebenen Testfälle für alle Nutzszenarien.

Das Testmaster-Projekt wurde so konfiguriert, dass alle in der Versionsverwaltung vorhandenen Konfigurations-, Label- und Testfall-Dateien bequem per Mausklick im Web-Browser angezeigt werden können (siehe Abbildung 4-1).

Die Testsuite-Übersicht listet, wie in Abbildung 4-2 dargestellt, neben einigen Statistiken die gruppierbaren Testfälle mit ihrem aktuellen Status auf und kann mit der notwendigen Zugriffsberechtigung weltweit eingesehen werden. Ebenso können sowohl der Testfalleditor als auch die Testablaufsteuerung direkt gestartet werden. Die Testablaufsteuerung ruft die im Testmaster referenzierten Testfall-Dateien ab und aktualisiert den Testfall-Status nach Verarbeitung. Dazu wurden das Testmaster Plugin entwickelt und der Testmaster selbst geringfügig erweitert.

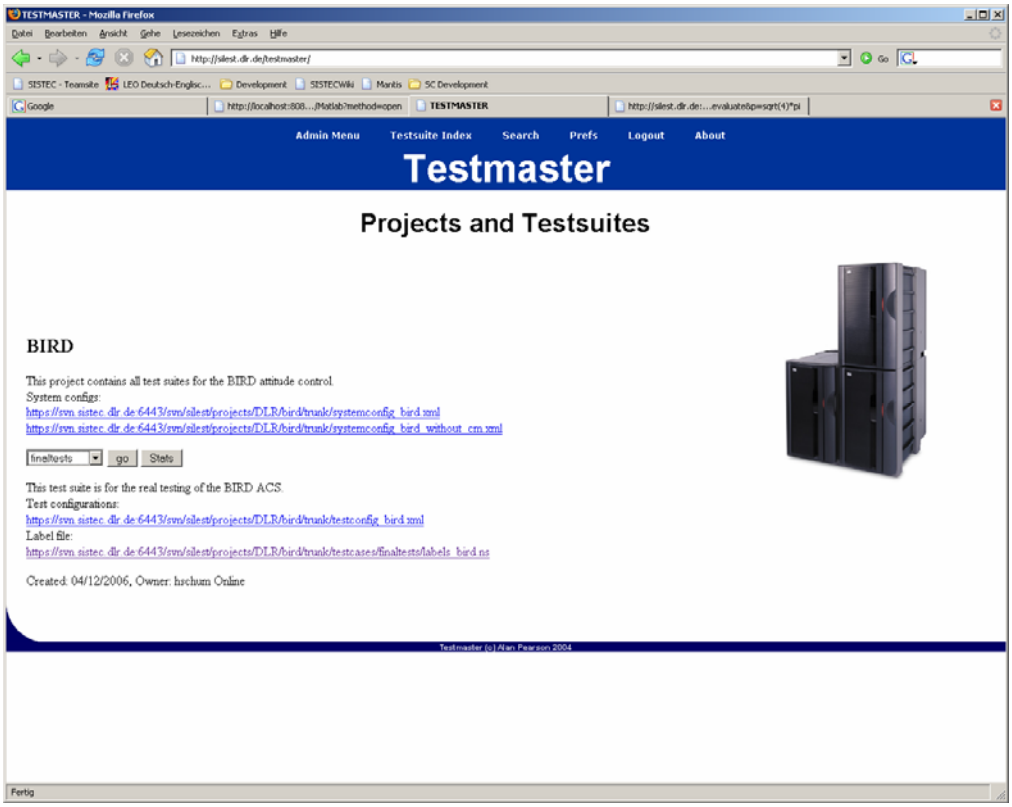

Abbildung 4-1: Projektübersicht im Testmaster

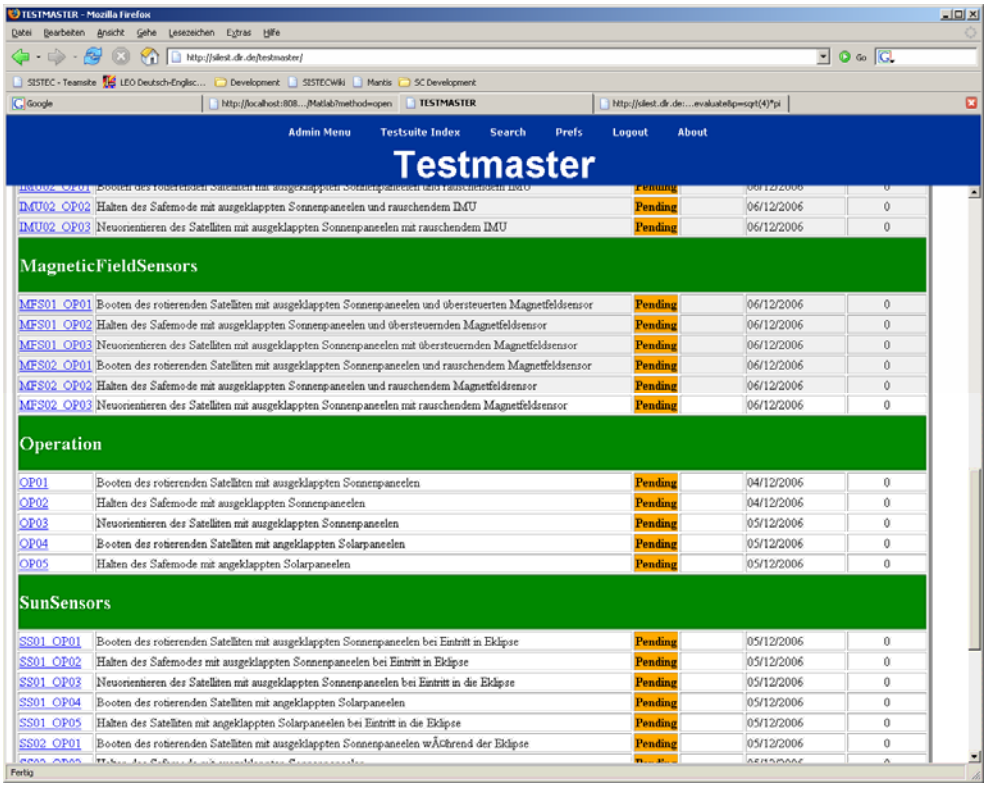

Abbildung 4-2: Testfall-Übersicht im Testmaster

4.1.2.2Subversion

Das Versionsverwaltungssystem Subversion wurde als Plugin in die Testumgebung integriert und erlaubt das automatisierte Auschecken der Konfigurations- und Testfall-Dateien sowie Einchecken der Testreporte.

4.1.2.3Testsystem

Für den SiL Test wird eine Umweltsimulation benötigt, die uns den Satelliten und das Verhalten der erforderlichen Hardwarekomponenten simuliert. Für die Simulation wurde der Kleinsatellit Bird nur aus ausgewählten Baugruppen des Satellitenträgers aufgebaut. Die Simulation erfasst lediglich Komponenten des Subsystems Lageregelung mit den Drallrädern als Aktuatoren und den Sensoren Sonnensensor, Magnetfeldsensor, Temperatursensor und Lasergyroskop. Die Lageregelungs-Baugruppen Magnet-spulensystem und Sternensensor wurden in der Simulation nicht mit berücksichtigt. Die Housekeeping-Informationen für diese Geräte enthielten den Status "Power off".

Die Erstellung der Umweltsimulation erfolgte als Simulink-Modell. Die einzelnen Komponenten des Simulink-Modells wurden zur Erleichterung der Widerverwendung in eine Modell-Bibliothek eingepflegt. Abbildung 4-3 zeigt eine grobe Übersicht über das Simulink-Modell der Umweltsimulation.

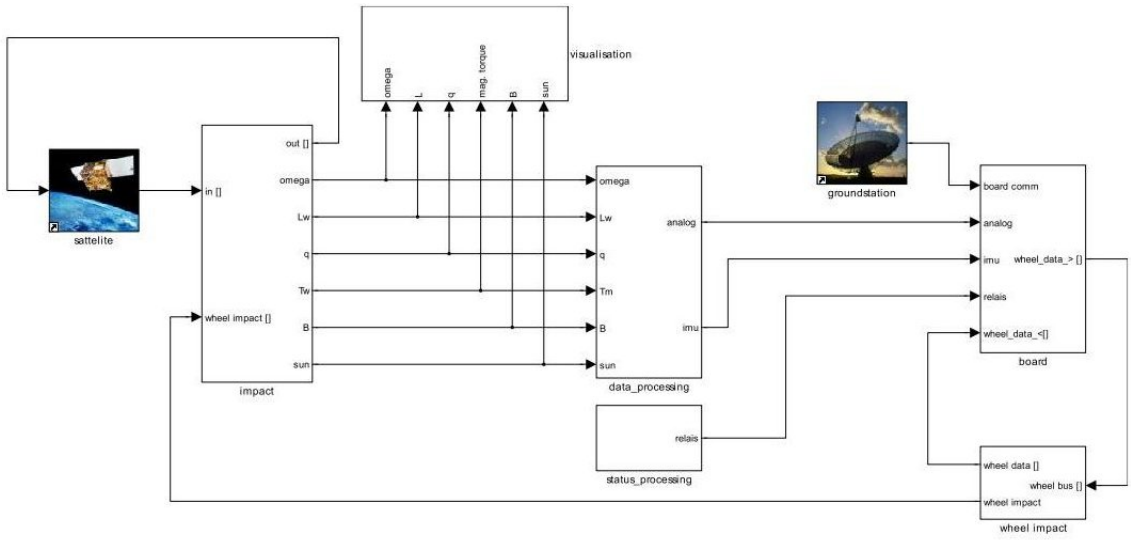

Abbildung 4-3 Aufbau der Satellitensimulation

Die implementierte Funktionalität der dargestellten Komponenten wird im Folgenden dargestellt.

- **Satellite**

  Das dynamische Verhalten des Satelliten wurde in dem Modell *satellite* beschrieben. Dieses Modell ist so realisiert, dass das physikalische Verhalten eines Satelliten im Satellitenkoordinatensystem beschrieben ist. Die Lage und die Kinematik des Satelliten werden durch einen 25-dimensionalen Vektor dargestellt.

- **Impact**

  Der Satelliten-Status wird durch einen 25-dimensionalen Vektor der physikalischen Parameter dargestellt. Zur Aufteilung der Informationen im 25-

dimensionalen Satellitenstatus wird dieser Vektor in einzelne Skalare und Vektoren aufgeteilt, welche auf die einzelnen Simulationsmodelle der Geräte aufgeteilt werden können. Das Modul *impact* realisiert die Aufteilung und spätere Komposition dieses Vektors. Weiterhin wird der Einfluss der Reaktionsräder auf die Dynamik des Satelliten in dieser Komponente umgesetzt.

- **Data_processing**

  Dieser Container beinhaltet die Simulationsmodelle für die Sensoren des Satellitenlageregelungssystems. Dies sind im Einzelnen der Magnetfeldsensor, die Sonnensensoren das Lasergyroskop. Die Messwerte des Temperatursensors lassen sich aufgrund der geringen Abhängigkeit von der Lage des Satelliten für die Testzwecke durch eine Konstante oder durch einen im Testfall vorgegebenen Temperaturverlauf vorgeben.

- **Board**

  Diese Komponente beinhaltet das Interface zur Einbindung der zu testenden Software in die Simulation. Durch entsprechende Ein- und Ausgabevariablen im Matlab-Workspace erfolgt ein Datenaustausch zwischen der zu testenden Software, welche auf einem separaten Hardwareboard läuft, und der Simulation der Dynamik des Satelliten und seiner Hardware. Das verwendete Hardwareboard enthält eine ähnliche Hardwareausstattung wie der Hauptcomputer des Bird-Satelliten. Der Unterschied zwischen der Originalrechner-Hardware und dem verwendeten Hardwareboard besteht lediglich in den Interfaces, welche für den SiL-Test durch den in Abschnitt 3.4 beschriebenen Simulationskern ersetzt werden.

- **Wheel_impact**

  Diese Komponente enthält ein vereinfachtes Modell der Reaktionsräder des Kleinsatelliten Bird. Dieses Modell setzt das Kommunikationsprotokoll auf Seiten der Reaktionsräder um und berechnet aus den empfangenden Kommandos einen Drehimpuls der in einem Tetraeder angeordneten Reaktionsräder.

- **Status_processing**

  Zur Verwaltung der Stromversorgung wird in einem Satelliten eine Power Management Unit verwendet. Diese Einheit kann einzelne Baugruppen des Satelliten über Relais ein- und abschaltet. Innerhalb dieser Komponente wird der Schaltzustand dieser Relais simuliert.

- **Groundstation**

  Um den Satelliten zu kommandieren ist eine Simulation der Bodenstation notwendig. Diese Komponente erzeugt zeitgesteuert die Information welche über die Funkstrecke von der Bodenstation an den Satelliten gesendet wird. Die zu sendenden Kommandos bestehen zum einen aus fest vorgegebenen Kommandos zur Konfiguration des Satelliten und speziellen Kommandos, welche Teil der Testfälle sind.

- **Visualisation**

  Als Feedback für die Entwicklung wurde in dieser Komponente eine Visualisierung realisiert. Mit dieser Komponente können alle physikalischen Parameter zur visuellen Kontrolle des Satelliten dargestellt werden.

4.1.3 Durchführung und Ergebnisse

Der definierte Testprozess wurde bis zur Testdurchführung angewandt und zeigte sich für den Aufbau der Testumgebung und der Definition der Testfälle als geeignet. Eine Testdurchführung mit den definierten Testfällen war nicht möglich, da eine Lücke im Modul *wheel_impact* des Testsystems nicht geschlossen werden konnte. Hierdurch war es nicht möglich, Closed-Loop-Tests durchzuführen.

Zum Test der Ablaufsteuerung in einem Open-Loop-Betrieb sowie zur Evaluierung der Analyse-Funktion wurden einfache Testfälle erstellt, mit denen innerhalb einer kurzen Zeitspanne von insgesamt ca. 10 min die benötigten Testergebnisse vorlagen. Des Weiteren gelang es, die Tests nicht nur mit einer lokal vorliegenden Simulation, sondern auch mit einer Remote-Simulation auf einem je nach Berechtigung weltweit erreichbaren Rechner per Webservice auszuführen. Diese Tests zeigen, dass die Testausführung mit einer voll funktionsfähigen Umgebungssimulation auch für den Anwendungsbereich der Raumfahrtsysteme möglich ist.

Um die implementierten Modelle zu validieren, wurden das Satellitenmodell sowie die implementierten Sensoren einer qualitativen Prüfung unterzogen. Eine quantitative Überprüfung wurde an den restlichen Komponenten mit entsprechenden Tests durchgeführt. Dazu wurden entsprechende Matlab-Skripte geschrieben, mit denen die Validierung der Komponenten durchgeführt wurde.

Die aufgetretenen Schwierigkeiten bei der Durchführung der Arbeiten lagen außerhalb des beschriebenen Testprozesses in der Informationsbeschaffung. Insbesondere enthielt die vorliegende Dokumentation nicht die für die Erstellung der Simulation benötigten Informationen, sondern sie mussten umständlich ermittelt werden. Dieser Umstand sollte bei der Anwendung des in SiLEST definierten Testprozesses als Risikofaktor für ein Projekt mit einkalkuliert werden.

4.1.4 Testaufwand

Aufwendig bei der Testdurchführung ist zum einen die Erstellung der Testfälle, zum anderen der Aufbau der Umgebungssimulation. Der Aufwand für die eigentliche Durchführung und die Auswertung der Testläufe fällt durch die Testautomatisierung gering aus.

Für die Definition eines Testfalls können im Mittel ca. zwei Stunden Arbeitszeit veranschlagt werden. Den größten Aufwand verlangt die Definition der Betriebsszenarien, insbesondere die Definition eines korrekten Analyseteils. Die Definition des Analyseteils benötigt eine besondere Kenntnis über das erwartende Verhalten des zu testenden Systems. Die hierzu nötigen gröberen Randbedingungen lassen sich in der Regel der Spezifikation des Systems entnehmen. Die genaue Reaktion auf ein Kommando mit ihrem zeitlichen Verhalten ist jedoch in der Regel nicht Teil der Systemspezifikation des Testobjekts.

Die Durchführung der Tests ist vollständig automatisiert und benötigt keine Eingriffe durch den Testingenieur. Genauere Aussagen über die Laufzeiten der Tests können leider nicht gegeben werden und hängen stark von der Komplexität der Simulation und der in den Testfällen berücksichtigten Laufzeit ab. Zum Beispiel, bei Lageänderung beträgt die Laufzeit eines Testfalls mindestens fünf Minuten. Diese Zeit wird sich durch die Synchronisation für die Laufzeit des Tests in etwa verdoppeln. Aufwandsabschätzungen für die Auswertung der Testergebnisse können leider auch nicht gegeben werden.

Den größten Kostenpunkt für den SiL-Test stellt der Aufbau des Testsystems dar. Dieser Aufbau entspricht einem eigenen Projekt. Im Fall der im Projekt durchgeführten

Anwendungserprobung des Testprozesses besteht das größte Hindernis aus der Beschaffung und Aufbereitung der Unterlagen eines fünf Jahre alten Projekts. Eine zeitnah durchgeführte Modellierung im Rahmen eines Raumfahrtprojekts würde den Aufwand wesentlich verringern. Ebenso kommt hinzu, dass Verhaltensmodellierungen bereits in einer frühen Phase eines Raumfahrtprojekts durchgeführt werden. Diese Modellierung sollten zukünftig in einer für den Testansatz geeigneten Form durchgeführt werden Auch die Wiederverwendung und Neuparametrierung von bestehenden qualifizierten Modellen sollten den Aufwand für den Testansatz senken.

4.2 IAV

Die IAV nutzte im Rahmen des SiLEST-Projekts die zweistufige Aufladung eines Dieselmotors als Beispiel für die Entwicklung von Motorsteuerfunktionalitäten. In diesem Abschnitt werden die Testdurchläufe für die Regelung der zweistufigen Aufladung beschrieben. Es handelt sich dabei um die automatische Durchführung mit der Testablaufsteuerung (TSC).

4.2.1 Testfälle

Für den Regler der zweistufigen Aufladung wurden insgesamt 60 Systemtests identifiziert und definiert. Diese dienen als Referenz für die Beurteilung des Vergleichs von Software-in-the-Loop (SiL) und Hardware-in-the-Loop (HiL), einem Ziel des SiLEST-Projekts.

Es wurden 3 Testfallkategorien unterschieden, die in Tabelle 4 aufgelistet sind:

Testfall	Beschreibung	Auswertung	Notation	Anz.
Statische Testfälle	Diese untersuchen das statische Verhalten bei einem Sprung der Regelgröße zwischen zwei Arbeitspunkten	Toleranzband ±10%	*za_fkt_<aaa>*[7]	36
Dynamische Testfälle	Diese untersuchen das dynamische Verhalten bei einem Sprung der Regelgröße zwischen zwei Arbeitspunkten; abhängig von den statischen Test-fällen	Dynamik-Vorgabe	*za_fkt_dyn <aaa>*	6
Testfälle mit Sensor- und Aktuatorfehlern	Diese untersuchen die Auswirkungen von Sensor- und Aktua-torfehlern (z. B. Rauschen) auf das statische Verhalten bei einem Sprung der Regelgröße zwischen zwei Arbeitspunkten; abhängig von den statischen Testfällen	Toleranzband ±10%	*za_sen_<aaa>*	12
			za_act_<aaa>	6

[7] <aaa> entspricht einer dreistelligen Nummer

Tabelle 4 Untersuchte Testfallkategorien

4.2.2 Aufbau der Testumgebung

Der prinzipielle Aufbau der SiLEST-Testumgebung ist im Abschnitt 3.1 beschrieben. An dieser Stelle soll er mit seinen entsprechenden Konfigurationen, so wie er für die Tests mit der Testablaufsteuerung in der IAV benutzt wurde, vorgestellt werden.

Als Testmanagementtool wird TestDirector / Quality Center und für die Versionsverwaltung StarTeam verwendet. Das Testsystem besteht in der MiL- und der SiL-Variante aus einem Matlab- / Simulink-Modell. Die folgenden Abschnitte beschreiben detailliert die entsprechenden Bestandteile TestDirector, StarTeam, Testsystem und Testablaufsteuerung.

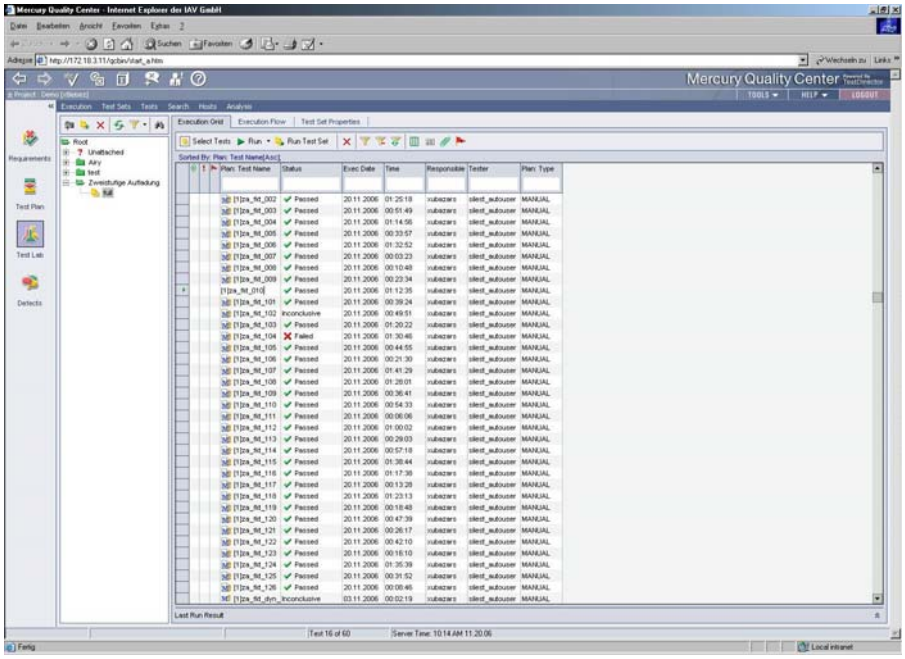

Abbildung 4-4 Screenshot des Projekts in TestDirector

4.2.2.1TestDirector

Projekt und Zugang

Für die Durchführung der Tests existiert ein TestDirector-Projekt namens *Demo* in der Domain "TEST_MST_43" auf dem IAV-internen TestDirector-Server. Verwendet wird die Version 8.2 von TestDirector. Der Zugang erfolgt über eine Nutzerkennung, weshalb entsprechende Accounts angelegt wurden, die mit den IAV-internen Nutzerkennungen übereinstimmen. Zusätzlich existiert der in seinen Rechten beschränkte Nutzer *silest_autouser* für den automatisierten Zugriff.

Der TestDirector bietet eine breite Funktionalität zur Verwaltung von Tests. Im Rahmen von SiLEST werden nur die Kernfunktionalitäten genutzt.

Testfälle (Test Plan)

Die Tests sind im Ordner *Zweistufige Aufladung* definiert. Hier existiert jeweils ein Test für jeden Testfall. Die insgesamt 60 Tests sind entsprechend den Testfallkategorien in Unterordnern thematisch gruppiert (siehe Abschnitt 4.2.1).

Testresultate (Test Lab)

Im Test Lab von TestDirector existiert eine Testsuite *full* im Ordner *Zweistufige Aufladung*, die alle definierten Tests enthält. Für jeden Test werden die erzielten Resultate von der Testablaufsteuerung separat abgespeichert. Die Abbildung 4-4 zeigt das Test Lab von TestDirector mit den definierten Testfällen.

4.2.2.2StarTeam

Als Repository für die Testfälle wird das SiLEST-Projekt in StarTeam 2005 R2 verwendet. Im entsprechenden Unterordner liegen alle Daten im Zusammenhang mit den automatisierten Tests. Der Ordner ist wie folgt definiert:

- test_system
- test_cases
- results

Die weitere Untergliederung richtet sich nach den Testmethoden (MiL, SiL, PiL) und enthält Bestandteile des Modells der zweistufigen Aufladung, die von den verschiedenen Testmodi verwendet werden, sowie die jeweiligen Testkonfigurationen und Testreporte.

Ein speziell für den automatisierten Zugriff eingerichteter Nutzer *silest_autouser* besitzt aus Sicherheitsgründen nur einen Schreib- und Lesezugriff auf dieses Verzeichnis. Abbildung 4-5 ist ein Screenshot des Projekts in StarTeam und zeigt einerseits die Struktur und andererseits die eingecheckten Testreporte.

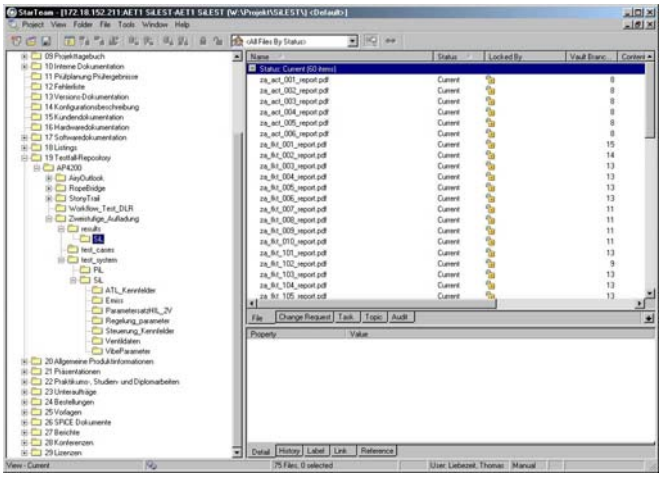

Abbildung 4-5 Screenshot des Projekts in StarTeam

4.2.2.3Testsystem

Das zu testende System besteht aus einem Simulink-Modell für Matlab R14SP2.

Modell

Das Ziel der Aufladung ist es, auf beliebige Art und mit Hilfe eines geeigneten Systems die Ladungsdichte des Arbeitsmediums (Luft bei Diesel- bzw. Luft-Brennstoff-Gemisch bei Ottosaugmotoren) vor dem Einströmen in den Arbeitszylinder anzuheben, d. h. vorzuverdichten. Bei der zweistufig geregelten Aufladung, die für das Projekt SiLEST im Mittelpunkt der Betrachtung steht, sind zwei unterschiedlich große Abgasturbolader, die für verschiedene Betriebspunkte optimiert sind, in Serie geschaltet. Die physikalischen Grundlagen sind in der Matlab- / Simulink-Bibliothek enDyna THEMOS® umgesetzt. Das Modell bildet die Regler-Funktionalität, die später auf dem Steuergerät laufen soll. Das Umgebungsmodell beinhaltet die Submodelle Fahrer, Motor, Fahrzeug inklusive Getriebe und Kupplung und bildet die Regelstrecke.

TargetLink-Modell der zweistufigen Aufladung

Für die automatische Testablaufsteuerung wird das Simulink-Modell der zweistufigen Aufladung im Software-in-the-Loop-Modus genutzt. Dazu wurde das Teilmodell für das Steuergerät und den Regler in ein TargetLink-Modell umgewandelt und daraus ein C-Code generiert.

Bibliotheken

- TargetLink (dSPACE)
- SALib (Sensors and Actuators Library, siehe Abschnitt 3.4)
- enDyna THEMOS

Die Bibliotheken müssen auf dem Testrechner installiert sein. Der Pfad zu SALib und den zugehörigen Unterverzeichnissen muss in den Matlab-Pfad eingebunden werden.

4.2.2.4Testablaufsteuerung

Für die automatische Abarbeitung der Tests wird die Testablaufsteuerung verwendet. Die Testablaufsteuerung wird per Batchdatei als geplante Task von Windows regelmäßig auf dem Testrechner ausgeführt. Installationshinweise finden sich in der zugehörigen TSC-Dokumentation. In der Tabelle 5 werden die für die Ausführung notwendigen Konfigurationen und Bestandteile wie Pythonskript, Systemkonfiguration, Testkonfiguration und Batchdatei vorgestellt.

Bestandteil	**Beschreibung**
Pythonskript	Für den Testablauf wird das unveränderte Pythonskript der Testablaufsteuerung verwendet. Ablageort: *data/scripts/* im Verzeichnis der TSC (lokal) Dateiname: *Pythonscript.py*
Systemkonfiguration	Die verwendete Systemkonfiguration für die Tests konfiguriert das StarTeam- und das TestDirector-Plugin. Ablageort: *data/systemconfigs/* im Verzeichnis der TSC (lokal) Dateiname: *IAV_ZA.xml*
Label	Die Label-Datei definiert die Labels und wird vom Testfalleditor verwendet, um die Eingabe und Bearbeitung von Testfällen und -

	konfigurationen zu vereinfachen. Ablageort: *test_cases/* (StarTeam) Dateiname: *silest.iav.za.ns*
Testkonfiguration	Die Testkonfiguration beschreibt das zu verwendende Testsystem und das Mapping. Das Testsystem besitzt die Parameter, Eingänge und Ausgänge. Dabei handelt es sich nur um die für die Testablaufsteuerung relevanten Daten. Das Mapping nutzt die vereinbarten Label und verbindet diese mit den definierten Parametern, Ein- und Ausgängen. Ablageort: *test_system/* (StarTeam) Dateiname: *config_zweistufige_aufladung_sil_targetlink.xml*
Testfälle	Die Testfälle werden durch XML-Dateien beschrieben. Ablageort*: test_cases* (StarTeam) *Dateiname: siehe Abschnitt 4.2.1*
Batchdatei	Die Batchdatei ruft die Testablaufsteuerung auf. Ablageort: im Hauptverzeichnis der Testablaufsteuerung (lokal) Dateiname: *ZA.bat*

Tabelle 5 Konfigurationen und Bestandteile der Testablaufsteuerung

4.2.3 Durchführung und Ergebnisse

Der automatisierte Testablauf lief seit Oktober 2006 auf einem speziell dafür verwendeten Rechner ab. Die Testablaufsteuerung wird per Batchdatei als geplante Task von Windows regelmäßig auf dem Testrechner ausgeführt. Die auf diese Weise und gemäß dem Workflow erzielten Ergebnisse und Erfahrungen werden im Folgenden vorgestellt.

Testresultate

Die Testresultate sind in TestDirector einsehbar, und dort lässt sich auch ein Bericht mit allen generellen, die Testfälle betreffenden Informationen generieren. Insgesamt erhalten 55 der 60 Testfälle das Resultat *Passed*, während 4 fehlschlagen (Testurteil *Failed*). Ein Testfall wird mit dem Testurteil *Inconclusive* bewertet, da der entsprechende Testfall von einem fehlgeschlagenen Testfall abhängt und somit nicht ausgeführt wurde.

Testreporte

Für jeden Testfall mit dem Testresultat *Passed* oder *Failed* wird von der Testablaufsteuerung ein Testreport im PDF-Format automatisch erstellt. Dieses Verhalten ist konfigurierbar und wurde in den Testfällen so ausgewählt. Ein Testfall mit dem Resultat *Inconclusive* oder ein abgebrochener Testfall besitzt keinen Testreport.

Die Testreporte werden aus den verschiedenen im SiLEST-Prozess vorhandenen XML-Dateien von einer XSLT-Transformation zusammengebaut. Die Testreporte und -archive werden prinzipiell in StarTeam automatisch eingecheckt.

Auswertung der erzielten Ergebnisse

Eine detailliertere Übersicht über den Verlauf der Anzahl der Testfälle und die erreichten Resultate gibt Abbildung 4-6. Sie zeigt die Historie der erhaltenen Testergebnisse, mit im Hintergrund eingezeichneter absoluter Anzahl der Tests. Des weiteren ist für die relevanten Testresultate (*Passed*, *Failed*, *Inconclusive*, *Pending*) jeweils die erzielte Anzahl aufgetragen. Die Testablaufsteuerung ist täglich gelaufen. Es lassen sich verschiedene Resultatbereiche identifizieren:

- Anfangsphase: Diese zeigt sich in einer hohen Anzahl an *Inconclusive*-Resultaten (bis Anfang Dezember 2006). Hier funktionierte die TSC noch nicht korrekt.
- Korrektes Funktionieren: 55 *Passed*, 4 *Failed*, 1 *Inconclusive*.
- Abstürze: Diese sind gekennzeichnet durch eine hohe Anzahl an Tests mit *Pending*. Deutlich erkennbar sind die Probleme um den Jahreswechsel 2006 / 2007, die wegen Urlaub und Feiertagen erst im Neuen Jahr erkannt und behoben wurden. Hier kam es permanent zum Absturz der Testablaufsteuerung aufgrund einer gesperrten Datei im Matlab-Verzeichnis.

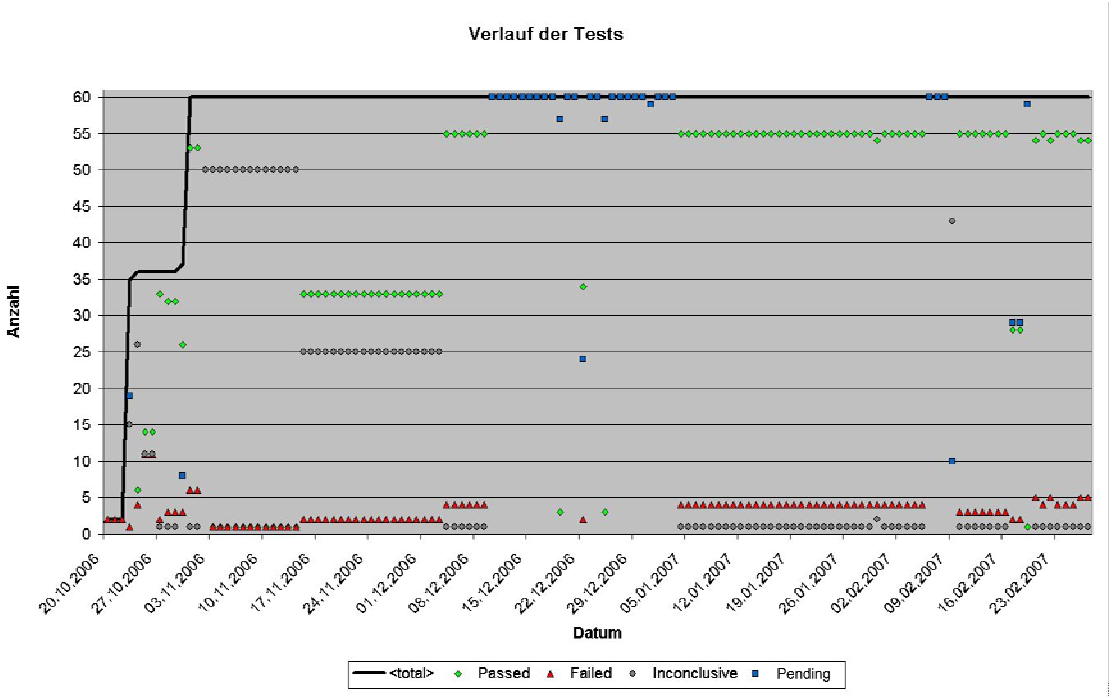

Abbildung 4-6 Auszug aus dem TestDirector-Bericht

4.2.4 Testaufwand

Zeitlicher Aufwand

Die Erstellung von Testfällen mit dem Testfalleditor dauert je nach Ähnlichkeiten unter den Testfällen 5 bis 30 min pro Testfall. Der zeitliche Aufwand für die Erstellung der Simulationsmodelle fällt in jedem Fall an. Die Dauer der Vorbereitungszeit ist stark vom vorhandenen Einarbeitungsgrad abhängig.

Die durchschnittliche Durchführungszeit für die gesamte Testsuite mit den 60 Testfällen betrug auf dem Testrechner 2 h 47 min bzw. 45 min auf einem moderneren Rechner. Die gemessenen Zeiten sind jedoch vom Umfang des Modells, der Leistungsfähigkeit des Rechners und der zu simulierenden bzw. zu testenden Zeitdauer stark abhängig.

Die Nachbereitung umfasst die im Workflow verankerte Überprüfung der Testresultate. Sie erfolgt in regelmäßigen Abständen nach jedem Regressionsschritt und umfasst den Vergleich der Resultate, die Analyse von neuen *Failed*-Resultaten sowie die stichprobenartige Überprüfung der *Passed*-Resultate. Die Sichtung der Testresultate dauerte z. B. insgesamt ca. 10 min, während eine detaillierte Überprüfung eines Testfalls 0,5-1 h in Anspruch nahm. Abweichend vom definierten Testprozess wurden im vorliegenden Fall sämtliche Testergebnisse überprüft. Hintergrund war die Überprüfung der korrekten Arbeitsweise des Prototypen der Testablaufsteuerung.

Speicherplatzbedarf

Die Tabelle 6 gibt den benötigten Speicherplatz für die zweistufige Aufladung in Abhängigkeit von der Testmethode an.

Testsystem	MiL	4,1 MB
	MiL zeitdiskret	3,5 MB
	MiL TargetLink	4,5 MB
	SiL	4,9 MB
Testfälle	einer (Durchschnitt)	2,3 KB
	60 (Summe)	142 KB
Testreporte	einer (Durchschnitt)	167 KB
	60 (Summe)	9,62 MB
Archivdateien	eine mit Ergebnissen aller 60 Testfälle (Durchschnitt)	160 MB

Tabelle 6 Speicherplatzbedarf in Abhängigkeit von der Testmethode

Die Größe der Reporte ist maßgeblich von der Anzahl der Bilder im Report abhängig, und diese hängt von der Anzahl der Inputs und der Subtests ab. Die clevere Ablage der gemeinsam genutzten Dateien spart Platz. Das Zip-Archiv wird sehr groß. Eine feinere Steuerung der Archivierung wäre an dieser Stelle wünschenswert.

5 Bewertung des Testprozesses

5.1 Projektablauf

Die erzielten (Zwischen-)Ergebnisse wurden in zwei- bis dreimonatlich stattfindenden Projekttreffen ausgetauscht. Ebenso wurde hier das weitere Vorgehen mit den anderen Projektpartnern abgestimmt. In der Phase der gemeinsamen Software-Entwicklung für die Testablaufsteuerung fanden zudem zusätzlich regelmäßige Entwicklertreffen der beteiligten Projektmitarbeiter statt. Diese regelmäßigen Abstimmungstreffen erwiesen sich als sehr hilfreich für das zügige Vorankommen des gesamten Projekts und trugen wesentlich zum Projekterfolg bei.

Für das SiLEST-Projekt stand eine umfangreiche Infrastruktur zur Verfügung:

- Email-Verteiler
- Sharepoint-Teamsite
- Website: www.silest.de
- Change Request Tool: MANTIS
- Konfigurationsmanagement: Subversion-Server
- Simulationsrechner: 1/8 Blade

Gegenüber der ursprünglichen Projektplanung gab es während der Projektlaufzeit folgende Änderungen, die dem Projektträger im Rahmen der halbjährlichen Zwischenberichte mitgeteilt wurden.

- Aufgrund eines deutlich verzögerten Projektbeginns stellte sich ein Zeitverzug von ca. vier Monaten ein. Dieser wurde über weite Phasen des Projekts konstant gehalten. Da auch die finanziellen Mittel in entsprechend geringerem Umfang abgerufen wurden, beantragten die Projektpartner eine kostenneutrale Verlängerung um vier Monate. Dieser wurde vom Projektträger stattgegeben.
- Nicht zu realisieren war die Kalibrierung der Simulation mit Hilfe von Daten, welche im STEP-Format vorliegen. Nach eingehendem Studium des STEP-Standards wurde dieses zwar nach wie vor für möglich gehalten. Aufgrund der Komplexität von STEP ließ es sich jedoch nicht im Rahmen des Verbundprojekts umsetzen.
- Ähnliches galt auch für den Aufbau der Simulation aus der UML bzw. dessen Pendant SysML heraus. Dies wurde nach wie vor als möglich angesehen. Allerdings ist aufgrund der Eigenschaften der automatischen Code-Generierung mit Hilfe eines UML-Werkzeugs und des Aufbaus von Matlab / Simulink-Simulationen mit Schwierigkeiten zu rechnen. Insbesondere die Vermengung von visuellen Informationen zur Modelldarstellung in Matlab / Simulink und von Informationen für die eigentliche Simulation in den Simulationsmodelldateien stand diesem Schritt im Wege. Dieser Ansatz wurde daher im Verbundprojekt nicht weiter verfolgt.
- Vom Aufbau einer Modelldatenbank wurde innerhalb des Projekts abgesehen, da es hier bereits kommerzielle Lösungen auf dem Markt gab und bei möglichen Anwendern des SiLEST-Testprozesses auf die dort bereits vorhandenen Tools zurückgegriffen werden muss.
- Synergieeffekte mit dem HCCI-Projekt der IAV ergaben sich nicht. Statt dessen ergab sich eine Kooperation mit dem IAV Projekt AMeDA. Dort werden große

Datenmengen von realen Dauerlauffahrzeugen automatisch auf das Auftreten bestimmter Signalmuster hin untersucht. Für einen gemeinsamen Demonstrator von SiLEST und IAV AMeDA wurden wichtige Software-Bestandteile aus beiden Projekten so kombiniert, dass es nun möglich ist, in großen Datenmengen automatisch SiLEST-Testfälle zu finden und zudem zu überprüfen, ob die dazugehörigen Ausgangssignale den in den Testfällen spezifizierten Ausgangssignalen entsprechen.

5.2 Ergebnisse und Erkenntnisse

Die folgenden Erfahrungen beziehen sich auf die Arbeit mit der SiLEST-Testautomatisierung. Für die Erstellung von Testfällen und Testkonfigurationen ist der Testfalleditor bestens geeignet. Er ist jedem normalen XML-Editor vorzuziehen. Zu beachten ist, dass der Testfalleditor die jeweils aktuellen Versionen von Testfall und Testkonfiguration unterstützen muss. Die Systemkonfiguration kann leider noch nicht mit dem Testfalleditor erstellt werden. Gerade wegen der guten Erfahrungen mit den Testfällen, sollte dies implementiert werden. Bei der Erstellung der Testfälle hat sich gezeigt, dass das definierte Testfallformat sehr flexibel bei der Definition von Signalen ist. Es bietet z.T. mehrere alternative Möglichkeiten, die zum gleichen Ergebnis führen, damit der Ersteller, die für ihn intuitivste verwenden kann. Dank der Trigger müssen Zeiten in der Auswertung nicht ausschließlich explizit definiert werden, sondern lassen sich auch implizit formulieren. Der von der Testablaufsteuerung realisierte Ablauf entspricht dem im Workflow definierten Vorgehen und hat sich bewährt.

Der Verzicht auf eine Modellierung von Anforderungen im Rahmen der Tests hat sich nicht negativ auf die Arbeiten ausgewirkt. Vielmehr handelt es sich hierbei um eine Aufgabe, die in die reine Abarbeitung der Tests nicht hineinspielt. Eine Kopplung der Testverwaltung mit einem Werkzeug für das Anforderungsmanagement ist je nach verwendeter Software aber möglich.

Als Simulationsmodell-Repository wurde mit der Datei-orientierten Versionierung eine pragmatische, sich an den benötigten Erfordernissen orientierende Lösung umgesetzt, die sich für die betrachteten Aufgaben als ausreichend erwiesen hat.

Die Testresultate sind über die verwendete Testfallverwaltung wie beispielsweise TestDirector oder Testmaster einsehbar. Während der Abarbeitung einer Testsuite lässt sich der aktuelle Status von jedem beliebigen Rechner aus verfolgen, da er von der Testablaufsteuerung an die Testverwaltung gesendet wird. Bei der Testauswertung verlangt die reine Überprüfung der Testresultate keine vertieften Kenntnisse. Muss allerdings das Testresultat als solches noch einmal überprüft werden, so ist das Systemverständnis bzgl. des Testobjekts von immanenter Bedeutung.

Die Arbeit mit der Testablaufsteuerung hat sich über einen langen Zeitraum im Regressionsbetrieb bewährt und dies trotz des frühen Entwicklungsstandes. Der Umfang der automatisiert generierten Testreporte hat sich im untersuchten Fall als den Anforderungen entsprechend erwiesen. Alle Arbeiten wurden von qualifizierten Entwicklern durchgeführt, was im Prototypencharakter der Testautomatisierung begründet liegt. Insbesondere das Aufsetzen der Testablaufsteuerung verlangt im Moment noch Expertenwissen, welches sich aber mit einer verbesserten Dokumentation reduzieren lässt.

5.2.1 Vergleich der Durchführungen

5.2.1.1Vorausbemerkungen

In diesem Kapitel werden nur die Durchführungsvarianten für die Testfunktion der IAV aus dem Bereich Motorsteuerung miteinander verglichen, da es beim DLR nicht möglich war, Closed-Loop-Test durchzuführen (vgl. Abschnitt 4.1.3).

Für die zweistufige Aufladung der IAV wurden sowohl manuelle als auch automatisierte Tests durchgeführt. Daraus lassen sich zwei wesentliche Vergleiche extrahieren:

- Software-in-the-Loop (SiL) vs. Processor-in-the-Loop (PiL), jeweils manuell
- Manuell vs. automatisiert, jeweils SiL

Für alle Durchführungen wurden die gleichen Testfälle verwendet. Die Testfälle wurden zuerst textuell definiert und dann für die unterschiedlichen Tests (manuell bzw. automatisiert) entsprechend umgesetzt.

Die Testresultate ergeben sich aus den im Testfall vereinbarten erwarteten Signalverhalten und den dort festgelegten Grenzen für dieses. Alle Tests haben über die verschiedenen Durchführungen (SiL manuell, SiL automatisiert, PiL manuell) vergleichbare Signalverläufe und die gleichen Resultate erzeugt. Somit liefern alle Testmethoden das gleiche Ergebnis.

5.2.1.2Vergleich SiL (manuell) – PiL (manuell)

Der manuelle SiL-Durchlauf ist über Matlab-Skripte realisiert. Dies schließt eine automatische Bewertung mit ein. Die Lösung über die Matlab-Skripte ist anfällig bei Änderungen an den Testfällen und bei nicht korrekten Testfällen, da die Skripte manuell erstellt und gepflegt werden müssen.

Der PiL-Durchlauf erfolgt demgegenüber durch manuelle Eingabe sämtlicher Parameter der Testbedingungen in der ControlDesk-Maske. Für die anschließende Bewertung wird ein Matlab-Skript verwendet. Damit ist für den PiL-Durchlauf keine abgesicherte Reproduzierbarkeit gegeben: Bei einer Wiederholung können u.U. andere Ergebnisse erzielt werden.

Die Zeit zum Erstellen eines Testfalls ist nur für die manuelle SiL-Durchführung vorhanden. Beim manuellen PiL existiert keine wiederverwendbare Testfallvorgabe. Hier erfolgt die Eingabe bei jedem Test neu.

Die Testdurchführung dauert für die PiL-Tests wesentlich länger. Dabei nimmt der Test selbst den geringsten Teil der Zeit in Anspruch, da die Testfälle in Echtzeit ausgeführt werden. Die meiste Zeit beanspruchen das Einstellen der Testbedingungen und die Testauswertung. Für die Erstellung der Testreporte ist zudem eine sehr lange Nachbereitungszeit notwendig, da die Reporte manuell zu erstellen sind. Für beide Testmethoden wird für die Durchführung ein Tester mit entsprechendem Expertenwissen (Matlab / Simulink bzw. ControlDesk) benötigt. Die Anforderungen an diesen sind beim PiL-System höher, da der Aufbau komplexer ist. Das PiL-Testsystem besteht aus mehr Komponenten, die zudem noch teurer sind als das SiL-Testsystem (PC). Für PiL wird des Weiteren auch teure, zusätzliche Software und zusätzliches Expertenwissen benötigt. Zudem ist die Verfügbarkeit der Testinfrastruktur für die PiL-Tests geringer, weil weniger Echtzeit-Systeme als PCs zur Verfügung stehen und die verfügbaren Echtzeit-Systeme eine höhere Auslastung besitzen.

5.2.1.3Vergleich manuell – automatisiert (SiL)

Die textuell definierten Testfälle müssen für beide Vorgehensweisen umgesetzt werden. Für die manuellen Tests wurde dies in Form von Matlab-Skripten durchgeführt. Diese individuell erstellten Skripte sind eine hier vorgenommene Arbeitserleichterung, die keinesfalls die Regel in der praktischen Arbeit darstellt.

Für die automatisierten SiLEST-Tests muss eine Überführung in das SiLEST-XML-Format vorgenommen werden. Diese erfolgt mit Hilfe des SiLEST-Testfalleditors, der keinerlei XML-Kenntnisse voraussetzt und sehr intuitiv zu bedienen ist. Der SiLEST-Testprozess setzt zwingend die Verwendung einer Testfallverwaltung und einer Versionsverwaltung voraus. Für die manuellen Tests ist das nicht verpflichtend, d. h. es wird in diesem Fall nicht sichergestellt, dass die Ergebnisse archiviert werden.

Des Weiteren werden durch den SiLEST-Testprozess stets Testreporte mit allen verfügbaren Informationen in einem einheitlichen Format erzeugt. Dies ist bei der manuellen Durchführung nicht sichergestellt. Für beide Durchführungen genügt ein normaler Computer (PC). Das Testsystem ist identisch, was zu vergleichbaren Simulationsdauern führt.

Für die Wiederverwendung der Skripte der manuellen Durchführung ergeben sich die folgenden Aspekte:

- Die Realisierung zusätzlicher Testfälle ist auf einfache Weise möglich, sofern diese den bisherigen Testfällen ähnlich sind. Sie kann durch einfache Abänderung erfolgen, setzt jedoch beträchtliches Wissen um die Funktionsweise der Skripte voraus.
- Das manuelle Vorgehen ist fehleranfällig und verlangt deshalb ein sorgfältiges und konzentriertes Arbeiten. Eine Übernahme der Skripte in andere Projekte verlangt verzweigte Änderungen.

Für die manuelle Durchführung müssen die Testreporte zudem manuell im Anschluss erzeugt werden. Demgegenüber erstellt die automatisierte Durchführung mit der TSC die Testreporte in einem einheitlichen Format automatisch mit. Das Aussehen der erzeugten PDF-Reporte lässt sich durch den Nutzer an die eigenen Dokumenten-Richtlinien anpassen. Damit relativiert sich auch die längere Durchführungszeit erheblich zugunsten der automatisierten Vorgehensweise.

In Abhängigkeit vom Automatisierungsgrad der Skripte lassen sich die manuellen Tests mit einer teilweisen Anwesenheit des Testingenieurs durchführen. Für die Durchführung wird beim automatisierten Test kein Testingenieur benötigt. Die Testablaufsteuerung verwendet ein Matlab-Plugin, das eine Matlab / Simulink-Simulation sowohl lokal als auch remote ansprechen kann. Damit ist es möglich, auf einen per Netzwerk erreichbaren Simulationspool für die Durchführung der Regressionstests zuzugreifen. Dafür können mehrere alternative Verbindungen angegeben werden, von denen dann die erste erreichbare verwendet wird. Die Testablaufsteuerung bietet eine größere Freiheit bezüglich der Ausführungszeit. So kann diese frei gewählt werden, was es ermöglicht, die Tests nachts oder am Wochenende durchzuführen.

Kenntnisse bezüglich Matlab / Simulink werden für beide Ansätze gebraucht. Da das Programm im vorhandenen Umfeld ein Standardtool darstellt, ist dies keine beschränkende Forderung. Für die manuellen Tests sind die benötigten Kenntnisse umfassender, denn die Verwendung der Matlab-Skripte setzt entsprechendes Wissen über ihre Programmierung und ihren Aufbau voraus. Für das Aufsetzen und die Betreuung der Testablaufsteuerung ist derzeit ein Experte notwendig. Mit der geplanten Produktisierung sollte dies jedoch nicht mehr nötig sein.

5.2.2 Bewertung

Die für die manuellen SiL-Tests verwendete Testdefinition ist in ihrer Art kein Standard für alle Testmethoden. Gerade die konkrete Definition des erwarteten Verhaltens ist normalerweise nicht ständiger Bestandteil der anzutreffenden Testfalldefinition. Diese Tatsache verzerrt den Vergleich, da die manuelle Testdurchführung häufig noch nach Augenmaß erfolgt. Entsprechend muss in diesem Fall auch die Bewertung manuell erfolgen.

Als Vorteile von SiL gegenüber von PiL wurden die folgenden Punkte identifiziert:

- SiL benutzt weniger Komponenten, zudem wird teure Hard- (Echtzeit-System) und Software (Compiler) für die Tests nicht benötigt. Damit gehen geringere Anforderungen an nötiges Expertenwissen einher, da allein schon die Anzahl der verwendeten Softwareprogramme geringer ist.
- Auf dem Steuergerät muss der volle Funktionsumfang bereitstehen. Dies ist für SiL nicht der Fall, für PiL nur eingeschränkt. Damit können SiL-Tests früher im Entwicklungsprozess erfolgen.

Demgegenüber steht das nur mit PiL / HiL vorhandene Echtzeitlaufzeitverhalten, das ein Alleinstellungsmerkmal darstellt. Für die SiLEST-automatisierte Durchführung sprechen die folgenden Gründe:

- Eine vollständig automatisierte Abarbeitung und Dokumentation von Tests wird ermöglicht. Diese ist zudem im Besonderen für Regressionstests geeignet, erfordert jedoch für die Einrichtung einmalig zusätzliche Ressourcen in Form von Expertenwissen.
- Die manuelle Durchführung verfügt über eine größere Fehleranfälligkeit. Die automatisierte Variante bietet damit einen sichereren Prozess.
- Die bei der manuellen Durchführung verwendeten Matlab-Skripte sind eine nicht standardisierte Lösung, die zwar bereits eine Vereinfachung des Arbeitsaufwands darstellt, aber nicht als allgemein akzeptierte und verbreitete Lösung verstanden werden kann. Demgegenüber stellt das SiLEST-Testfallformat einen formalen Rahmen für Tests mit analogem Signalverhalten bereit, der zudem bei der Eingabe auf formale Vollständigkeit überprüft wird.
- Die Wiederverwendung von Testfällen ist auf Grund der vom Testsystem unabhängigen Definition in einem höheren Maß gegeben.
- Der SiLEST-Testprozess integriert die Testverwaltung und Versionierung in den Testablauf. Damit wird die Reproduzierbarkeit und Nachvollziehbarkeit (Dokumentation) wesentlich gesichert.
- Die vom SiLEST-Testprozess automatisch generierten Testreporte stellen einen weiteren Vorteil der automatisierten Testdurchführung dar.

Durch das nicht vorhandene Echtzeitverhalten stellt SiL kein Ersatz für HiL dar. Vielmehr kann eine Verlagerung von Aufwänden in die früheren Entwicklungsphasen erfolgen. Auch später stellt SiL eine weniger aufwendige Testmöglichkeit bereit, die für alle Fragestellungen geeignet ist, in denen das Timing (Echtzeitverhalten) eine untergeordnete Bedeutung besitzt.

Durch die Automatisierung und die im Vergleich zum PiL weniger umfangreichen Ressourcen ist eine Reduzierung der Kosten zu erwarten.

In Bezug auf die für Vergleiche bzw. den Testfortschritt gern verwendeten Metriken lassen sich die folgenden Aussagen treffen: Da die Durchführung wesentlich

beschleunigt werden kann[8], lassen sich in der gleichen Zeit mehr Testfälle untersuchen oder die Zeit pro Testfall sinkt. Mit einer erhöhten Anzahl an Tests lässt sich potentiell eine höhere Abdeckung erreichen. Potentiell können so auch mehr Fehler gefunden werden. Weniger Zeit pro Testfall bedeutet auch geringere Kosten pro Testfall. Zusammen mit den geringeren Ressourcen, die benötigt werden, sind somit insgesamt sinkende Kosten zu erwarten.

5.3 Bewertung der Teststrategie und Sicherheitsanalyse

SiLEST leistet einen Beitrag zur effektiven Gewährleistung der Korrektheit, Sicherheit und Zuverlässigkeit technischer Systeme unter Beachtung der künftigen Trends des rechnergestützten System- / Software-Engineerings. Mit dem SiLEST-Projekt insgesamt wurden neue Technologien der Entwicklung eingebetteter kritischer Echtzeitsoftware in den folgenden Bereichen:

- Software-in-the-Loop-Test eingebetteter zeitlich kontinuierlicher kritischer Echtzeit-Software einschließlich der Verifizierung des Echtzeitverhaltens unter Nutzung der verteilten Simulation und von State-of-the-Art Teststrategien. Hierzu gehören die Erweiterungen des Betriebssystems BOSS und die Kopplung mit externen Simulatoren.
- Dynamische Safety- und Zuverlässigkeits- / Dependability-Analyse einschließlich der Verifikation und Validation von eingebetteter Software mittels komplexer Fehler- und Problemsimulation im System unter realistischen Echtzeit-Betriebsbedingungen.
- Detaillierte Analyse und Bestimmung der Einsatzbereiche und Anwendungsgrenzen von HiL- und SiL-Simulation und Entwicklung weitgehend automatisierbarer verallgemeinerter Testprozesse.

Die Frage, welche zu bewerten ist, ist ob und in wie weit mit der in SiLEST gewählten Methodik der Testszenariogenerierung die sicherheits- und verlässlichkeitsrelevanten Aspekte über die beiden zentralen SiLEST-Fallstudien hinaus technisch abgedeckt werden können.

Ziel des Projekts war es, eine geschlossene Regelschleife zwischen eingebetteter Steuergeräte-Software und simulierter Umgebung zu schaffen und dabei sowohl Fehlerinjektion als auch software-unterstützte Testerzeugung und eine automatisierte Ausführung zu ermöglichen. Wichtig sind vor allem diejenigen Merkmale der SiLEST-Architektur, die gegenüber früheren X-in-the-loop-Ansätzen neuartig sind. Dies ist die Zeitsynchronisation im PiL-Ansatz, die durchgängige Testprozess-Unterstützung, die erweiterten Testauswertungsfunktionalität und die Sensor- / Aktuator-Modellbibliothek.

5.3.1 Zeitsynchronisation

Die Kommunikation zwischen der Steuergerätesoftware und der Umweltsimulation erfolgt über ein spezielles Kommunikationsprotokoll, das die Unabhängigkeit der für beide Prozesse sichtbaren Zeit von der realen Zeit sicherstellt. Es nutzt damit die zusätzlichen Manipulationsmöglichkeiten aus, die eine reine Software-Lösung im Unterschied zu einer reinen HiL-Testumgebung bietet.

Dadurch wird es erstmalig möglich, grundsätzlich beliebig komplexe Umweltmodelle einsetzen zu können, ohne durch die Rechengeschwindigkeit des ausführenden

[8] Vergleich: SiL automatisiert vs. PiL manuell

Prozessors beschränkt zu sein. Hierdurch lassen sich der Anwendungsbereich der Testmethodik einerseits und die erreichbare qualitative (Modellkomplexität) und quantitative (Granularität der simulierten physikalischen Größen) Präzision ausweiten lässt.

Die Zusammenfassung einer gemeinsamer Anfangssequenzen verschiedener Testfälle kommt der üblicherweise baumförmigen Strukturierung von Fehlerszenarien, wie sie etwa bei einer Fault Tree Analysis anfällt, sehr entgegen. Musste bislang für jeden Pfad von der Wurzel bis zu einem Blatt eines Fehlerbaums ein kompletter Testdurchlauf veranschlagt werden, so genügte mit dem verfolgten Ansatz ein Teildurchlauf für jede Kante im Fehlerbaum. Dabei wird der einem internen Knoten entsprechende erreichte Zustand des Steuergeräts und der Umweltsimulation abgespeichert und mehrfach als Initialzustand für nachfolgende Teildurchläufe verwendet.

5.3.2 Durchgängige Testprozess-Unterstützung

SiLEST bietet erstmalig eine vollständige Formalisierung des SiL-Testprozesses. Testkonfiguration und -Parametrierung im weitesten Sinne wurden bislang ad hoc durchgeführt und konnten allenfalls in Form von z. B. Shell-Scripts festgehalten werden. Das im SiLEST-Projekt erarbeitete Konzept der Notation von Testfällen und -konfigurationen in einem jeweils einheitlichen XML-Format erlaubt prinzipiell eine durchgängige Werkzeugunterstützung von der Testfallerstellung bis zu einer Wiederholung früherer Testdurchläufe und eliminiert so eine wichtige Quelle von Bedienungsfehlern oder -inkonsistenzen in diesen Phasen.

Zum Beispiel wird durch den Ansatz ausgeschlossen, dass ein Testdurchlauf bei einer Wiederholung versehentlich in einer anderen Konfiguration ausgeführt wird als ursprünglich. Eine Versionierung der gesamten in diesen Phasen eingesetzten Software, einschließlich verwendeter Bibliotheken und der Testbeschreibungen selbst, erlaubt zudem die automatische Überprüfung auf exakt gleiche Rahmenbedingungen. Bislang wird allerdings die verwendete Software selbst nicht versioniert, sondern lediglich die Daten, mit denen die Software arbeitet.

5.3.3 Erweiterte Testauswertungsfunktionalität

Das SiLEST-Projekt hat sich vor allem auf Tests gegen Sensor- und Aktuatorfehler konzentriert. Es hat Klassen neuartiger Auswertungsfunktionen definiert und implementiert, die speziell für Tests in diesem Gebiet geeignet sind.

Sogenannte "Trigger-Funktionen" erlauben es, sich in der Spezifikation von Analogsignalen geforderter Eigenschaften nicht nur wie bisher auf feste, sondern auch auf ereignisbasierte Zeitpunkte zu beziehen. Beispielsweise ist es durch Triggerfunktionen möglich, die Testanforderung derart zu formulieren, dass ein Notstromaggregat spätestens zwei Sekunden, nachdem eine Netzspannung unter 200 Volt gesunken ist, anspringen muss.

Ohne Verwendung von Triggerfunktionen hatte man dagegen nur die Möglichkeit, die Netzspannung z. B. eine Sekunde nach Testbeginn abzusenken und zu verlangen, dass das Aggregat spätestens drei Sekunden nach Testbeginn anspringen muss. In solchen Fällen, in denen die Netzspannung nicht einfach beliebig setzbar ist, da sie selbst Ergebnis komplexer Umweltsimulationsberechnungen ist, scheidet diese Möglichkeit aus. Hier musste, um nicht auf die Überprüfung gänzlich zu verzichten, bislang in das Simulationsmodell eingegriffen und etwa eine zusätzliche boolesche Größe "Netzspannung lag vor zwei Sekunden über 200 Volt" mit einer Größe "Notstromaggregat läuft jetzt" durch "oder" verknüpft zur Testauswertung weitergeleitet werden. Weitere Eingriffe waren notwendig, um den Fall zu behandeln,

dass die Netzspannung nach einer gewissen Zeit wieder über den Grenzwert gestiegen ist und das Aggregat sich korrekterweise abgeschaltet hat.

5.4 Einsatzempfehlungen

Der SiLEST-Testprozess empfiehlt sich für den durchgängigen Einsatz über den gesamten Entwicklungsprozess mit seinen unterschiedlichen Testmethoden. Insbesondere die im Rahmen des Projekts neu geschaffene Testablaufsteuerung zur automatischen Ausführung der Testfälle stellt eine wesentliche Verbesserung für den Test eingebetteter Regelungssysteme dar.

Die vorhandene und für die Durchführung eingesetzte Version 1.0 unterstützt bereits die Testmethoden Model-in-the-Loop (MiL), Processor-in-the-Loop (PiL) und Software-in-the-Loop (SiL). Geplant ist zudem eine Erweiterung in Richtung Hardware-in-the-Loop (HiL).

Basierend auf den durchgeführten Vergleichen empfiehlt sich der frühest mögliche Einsatz im Entwicklungsprozess. Zu jedem späteren Zeitpunkt ist SiL immer dann gegenüber HiL zu bevorzugen, wenn das Timing (Echtzeitverhalten) eine geringe Bedeutung besitzt. Dies trifft im Besonderen für prinzipielle Untersuchungen von Reglerfunktionalitäten an Closed-Loop-Systemen zu.

Der Einsatz eines HiL-Testsystems ist außerdem angebracht, wenn es um schwer zu modellierende Effekte von Umgebungsbestandteilen (z. B. Sensoren oder Aktuatoren) geht, die einfacher als vorhandenes Bauteil in ein HiL-System einzubinden sind. Ebenso empfiehlt es sich zur Validierung des Timings und für EMV-Untersuchungen.

Für komplexere Simulationen der Umgebung, welche sich nicht unter Einhaltung der zeitlichen Randbedingungen berechnen lassen, ist der PiL-Ansatz vorzuziehen. Durch das definierte Synchronisationsprotokoll ist es mit diesem Ansatz auch möglich, das Testobjekt in einer Quasi-Echtzeit zu betreiben, obwohl die Simulation der Umgebung nicht in Echtzeit erfolgt. Dieser Testansatz erlaubt jedoch nur, Aussagen über die Anwendungslogik zu treffen. Tests auf der Ebene der Firmware erfordern einen Test mit einem HiL-System oder an einem Labormuster, gegebenenfalls auch im Open-Loop-Betrieb.

Allen Testansätzen gemeinsam ist, dass im Unternehmen eine Simulationskultur gelebt werden muss. Dies heißt, dass Simulationsmodelle einer ständigen Pflege bedürfen und gemäß neuen Anforderungen zu erweitern und anzupassen sind. Der Aufwand für die Modellierung einer Umgebungssimulation für ein einzelnes Projekt ist aufgrund der Anforderungen an die Qualität der Simulation zu hoch. Aufgrund der Ähnlichkeiten innerhalb der Produktpalette eines Unternehmens, ist der Weg des SiL-Testansatzes mit einer entsprechenden Pflege der Simulationsmodelle aus unserer Sicht wirtschaftlich möglich.

6 Ausblick

Es wurden von den Projektpartnern hauptsächlich vier verwertbare "Bestandteile" identifiziert: Die XML-Formate zur Testfallbeschreibung, die Testautomatisierung "TSC", der Testfalleditor "TFE" und die Sensor- / Aktuatorbibliothek für Simulink "SALib". Die nachhaltige Nutzung dieser Projektergebnisse ist von den Projektpartnern geplant. Dies erfordert eine Weiterentwicklung der Projektergebnisse zur Produktreife. Die Rahmenbedingungen hierfür wurden durch eine "Nutzungs- und Kooperationsvereinbarung über die Nutzung und Zusammenarbeit bei der Weiterentwicklung der Software SiLEST" zwischen den Projektpartnern DLR, IAV und webdynamix geregelt.

Die im Projekt offen gebliebenen Fragestellungen und das weitere Vorgehen der Projektpartner werden im Folgenden beschrieben.

6.1 Offene Punkte

Da der Aufwand für die Erstellung von Simulationsmodellen ein Kostenfaktor für die Anwendung des verfolgten Testansatzes ist, ist ein weiteres Verbesserungspotential in diesem Bereich zu sehen. Dies erfordert zum einen Maßnahmen zur Qualifizierung und Verbesserung der Wiederverwendung von Simulationsmodellen, zum anderen auch dem Aufbau eines Modell-Repositorys, welches das in den Simulationsmodellen enthaltene intellektuelle Wissen recherchierbar macht. Vorarbeiten zum Aufbau eines solchen Modell-Repositorys wurden bereits im Projekt geleistet.

Eine weitere Möglichkeit zur Verringerung des Aufwands für die Simulationserstellung besteht in einer Verknüpfung mit dem System-Engineering. Hierdurch lassen sich eine durchgehende Modellierung des Systems und seine Verifikation an einem konsistenten System-Modell durchführen.

Verbesserungspotential liegt noch in der Erstellung von Testfällen, insbesondere für den Bereich der Raumfahrt mit seinen Einzelfertigungen. Insbesondere eine Werkzeugunterstützung für das Ableiten von Testfällen aus einer FMEA und das mischen dieser mit operationellen Szenarien ist wünschenswert. Zurzeit erfordert dieser Schritt ein hohes Fachwissen über die Wechselwirkungen des Systems, insbesondere bei der Erstellung des Analyse-Abschnitts im Testfall.

Die für den SiLEST-Testprozess geschaffenen Werkzeuge haben einen Prototypen-Status und dienen lediglich als Demonstratoren. Dies bedeutet, dass die Nutzung ein hohes Expertenwissen erfordert. Zum Beispiel muss die Erstellung der Systemkonfigurationen zurzeit in einem XML-Editor durchgeführt werden. Eine Erweiterung des Testfalleditors zur Bearbeitung der Systemkonfigurationen ist für eine effektive Vorbereitung der SiLEST-Testautomatisierung notwendig.

6.2 Weiteres Vorgehen

Die IAV plant, den SiLEST-Testprozess so weiterzuentwickeln, dass er in allen Phasen der Funktionsentwicklung genutzt werden kann.

Dazu wurde die TSC bereits prototypisch für Nachsimulationen eingesetzt: Bei einer Nachsimulation werden reale Messdaten, z. B. aus Fahrzeugmessungen, verwendet, um daraus (teil-)automatisiert eine Testkonfiguration sowie Testfälle zu generieren. Diese werden anschließend mit der TSC simuliert. Auch die Ergebnisbewertung und Reporterstellung funktionieren – wie gewohnt – automatisch. Hierdurch können zu einem späten Entwicklungszeitpunkt die ausführbare Spezifikation sowie die reale Umsetzung im Steuergerät gegeneinander verglichen werden.

Des Weiteren läuft aktuell eine Diplomarbeit, die das Testen auf einem HiL-Simulator der Firma dSPACE mittels der TSC automatisieren soll.

Zudem will die IAV die Ergebnisse des SiLEST-Projekts intern noch bekannter machen und aktiv bewerben. Ziel ist die Nutzung der XML-Formate, der TSC, des TFE sowie der SALib auch in anderen IAV-Fachabteilungen. Gegebenenfalls soll eine Anpassung der Tools an die Belange bzw. eine Weiterentwicklung gemäß den Anforderungen anderer Projekte erfolgen.

Das DLR plant den SiLEST-Testprozess in zukünftigen Satellitenprojekten zum Einsatz zu bringen. Hierzu müssen die bei der Erstellung der Umgebungssimulation aufgetretenen Schwierigkeiten noch genauer analysiert werden und deren Ursachen behoben werden. Dies betrifft insbesondere die organisatorische Umsetzung des Prozesses zur Erstellung von Simulationsmodellen und deren Verifikation innerhalb der an einem Satellitenprojekt beteiligten Einrichtungen.

Zur Verbesserung der Wiederverwendbarkeit von Simulationsmodellen soll ein entsprechend der im Projekt definierten Anforderung genügendes Simulationsmodell-Repository aufgebaut werden. Des Weiteren soll die Erstellung der notwendigen Simulation enger mit dem System-Engineering und den in frühen Phasen eines Projekts durchzuführenden Simulationen gekoppelt werden. Als Ansatz wird hierzu eine Modellierung auf hohem Abstraktionsniveau mit SysML und die Verwendung von Techniken der MDA und automatischen Codegenierungen unter Einbeziehung des Simulationsmodell-Repositorys verfolgt.

Zur Erreichung dieser Verbesserungsmaßnahmen wurde im direkten Anschluss an das Projekt SiLEST ein DLR-internes Projekt gestartet. Die Arbeiten in diesem Projekt unterliegen einer Harmonisierung mit ähnlichen Arbeiten zum System-Engineering bei der europäischen Raumfahrtorganisation.

Webdynamix ist in der Konstellation des Verbundprojekts SiLEST als Softwarehersteller und Vermarkter vorgesehen. Hierbei ist Webdynamix auf weitere Kooperationen und Förderungen angewiesen, um die Weiterentwicklung der Werkzeugdemonstratoren zu marktfähigen Werkzeugen zu meistern.

Ein Teil dieser Kooperationen schlägt sich bereits in der Nutzungs- und Kooperationsvereinbarung nieder, in der die beteiligten Partner IAV, DLR und webdynamix ihre Zusammenarbeit in der SiLEST-Weiterentwicklung festgeschrieben haben.

Webdynamix plant die Produktentwicklung des Testfalleditors zu einem eigenständigen Produkt und hat hierzu nach dem Projektende erste Schritte eingeleitet. Die TSC soll in Folgeprojekten weiterentwickelt werden. Ihr Einsatz wird aufgrund der Komplexität zum Aufsetzen des Testverfahrens in Verbindung mit einer Dienstleistung zur Einrichtung des Testsystems sowie zur Schulung der Mitarbeiter in den zum Einsatz kommenden Methoden angestrebt.

Anhang A Veröffentlichungen

A.1 Veröffentlichungen in Zeitschriften und Präsentationen bei Konferenzen und Workshops

Im Rahmen des SiLEST-Projekts wurden folgende Beiträge in Fachzeitschriften veröffentlicht bzw. auf Konferenzen angenommen und präsentiert. (Die Veröffentlichungen und Präsentationen sind chronologisch angeordnet.)

Maibaum, O.: Einsatz von Simulationen bei der Softwareentwicklung in Raumfahrtprojekten. Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, 2004

Maibaum, O.: Einsatz von Simulationen in der Softwareentwicklung. Messtechnik-Seminar des Instituts für Energie- und Automatisierungstechnik, Technische Universität Berlin, 2004

Maibaum, O.: SiLEST – Entwicklung und Erprobung von Methoden und Werkzeugen für den Test und die Sicherheitsanalyse eingebetteter Echtzeitsoftware. Eröffnungskonferenz zur Forschungsoffensive "Software Engineering 2006", 2004

Maibaum, O.: Simulationsbasierter Test kritischer eingebetteter Software. Workshop Software-Technologie, DLR Braunschweig, 2004

Maibaum, O.; Rebeschieß, S.: Test von adaptiven Softwaremechanismen zur Fehlerkompensation. Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, Seiten 179–186. Expert Verlag, 2005

Maibaum, O.: Testing the Robustness of Spacecraft Control Software. Ercim News Nr. 65, Seiten 17-18, 2006

IAV GmbH: SiLEST – Software-in-the-Loop zum Testen eingebetteter Systeme. Info-Flyer, http://www.iav.de/_downloads/de/handouts/powertrainmechatronik/silest_de_11_04.pdf, 2006

Rebeschieß, S.; Liebezeit, Th.; Bazarsuren, U.: Automated Closed-Loop Testing of Embedded Engine Control Software. 11. Software & Systems Quality Conferences 2006, 7. ICS Test, 2006

Liebezeit, Th.; Schumann, H.; Rebeschieß, S.; Gühmann, C.; Bazarsuren, U.: XML Format for Automated Software-in-the-Loop Tests. 11. Software & Systems Quality Conferences 2006, 7. ICS Test, 2006

Rebeschieß, S.; Liebezeit, Th.; Bazarsuren U.; Gühmann C.: Automatisierter Closed-Loop-Softwaretest eingebetteter Motorsteuerfunktionen. Moderne Elektronik im Kraftfahrzeug, Seiten 201-211. Expert Verlag, 2006

Liebezeit, Th.; Schumann, H.; Gühmann C.; Rebeschieß, S.; Bazarsuren U.: XML-Format für den automatischen Software-in-the-Loop-Test. Moderne Elektronik im Kraftfahrzeug, Seiten 212–220. Expert Verlag, 2006

Montenegro, S.; Jähnichen, St.; Maibaum, O.: Simulation-Based Testing of Embedded Software in Space Applications. Embedded Systems - Modelling, Technology and Applications, Seiten 73-82. Springer Netherlands, 2006

Maibaum, O., Berres, A.: Präsentation des SiLEST-Workflows und der SiLEST-Tools im Rahmen der Ausstellung der 9th International Workshop on Simulation for European Space Programs (SESP), 2006

Rebeschieß, S.; Liebezeit, Th.; Bazarsuren, U.; Gühmann, C.: Automatisierter Closed-Loop-Testprozess für Steuergerätefunktionen. ATZ elektronik 01/2007, Seiten 34-41. Vieweg Verlag, 2007

Maibaum, O.: Comparison of Approaches to the Test of Control Unit Software. 12th Software & Quality Conferences International 2007. Düsseldorf, 2007

Maibaum, O.; Schumann, H.; Berres, A.: Test of satellite control software with simulated faults. Small Satellites for Earth Observation - Digest of the 6th International Symposium of the International Academy of Astronautics, Seiten 389-392. Wissenschaft und Technik Verlag Berlin, 2007

Schumann, H.; Berres, A.; Maibaum, O.; Liebezeit, T.: Simulation-Based Testing of Small Satellite Attitude Control Systems. Small Satellites for Earth Observation - Digest of the 6th International Symposium of the International Academy of Astronautics, Seiten 151-154. Wissenschaft und Technik Verlag Berlin, 2007

Rebeschieß, S.; Bazarsuren U.; Liebezeit, Th.; Gühmann C.: Automatisierter Closed-Loop-Testprozess für eingebettete Steuergerätefunktionen. 6. Symposium Steuerungssysteme für den Antriebsstrang von Kraftfahrzeugen, 2007

Rebeschieß, S.; Liebezeit, Th.; Bazarsuren, U.: Automatisierter Closed-Loop-Testprozess für eingebettete Steuergerätefunktionen, erscheint in der automotion 3/2007, 2007

A.2 Studien- und Diplomarbeiten

Die folgenden Studien- und Diplomarbeiten wurden während der Laufzeit des SiLEST-Projekts angefertigt. (Die Arbeiten sind chronologisch angeordnet.)

Wang, Z.Q.: Erstellung einer Simulink-SiL-Bibliothek mit Modellen für die Sensorik und Aktuatorik, Diplomarbeit, Technische Universität Berlin, 2005

Portl, S.: Einsatz automatischer Codegenerierung für Boden- und Onboard-Software zur Satellitenkommandierung. Diplomarbeit, TU Braunschweig (Mathematik und Informatik), 2006

Yazici, I.: Einbindung von Kfz-Steuergerätecode in Matlab / Simulink: Aufgabenstellung, Diplomarbeit, Technische Universität Berlin, 2006

Seidel, K.: Erweiterung einer Testfallbeschreibungssprache um Zustandsautomaten und abhängige Ereignisse, Diplomarbeit, Universität Potsdam, 2006

Kaprolat, S.: Anwendung einer Testautomatisierung auf ein seriennahes Closed-Loop-Getriebemodell, Studienarbeit, Hochschule Merseburg (FH), 2007

Anhang B Glossar

Dieser Anhang enthält Begriffdefinitionen von im SiLEST-Projekt verwendeten Worten. Innerhalb der Begriffsdefinitionen wird das Zeichen "→" vor Worten verwendet, welche ebenfalls im Glossar definiert sind.

Abstrakte(s) Simulink-{Template, -Klasse}

Ist ein Simulink-spezifisches, abstraktes und nicht kalibriertes → Template. Es ist Teil einer Simulink-Library.

Abstrakte(s) {Template, Simulationsmodul-Template, Modellklasse}

Ist ein → Simulationsmodul, welches in plattformunabhängiger und / oder simulatorunabhängiger Form in einem → Modell-Repository gespeichert ist. Es ist nicht kalibriert, wie z. B. ein spezielles Sensor- oder Motormodul. Die Überführung in eine simulatorabhängige Form wird durch → Wrapper erreicht, welche für das abstrakte Simulationsmodul-Template die Schnittstellen für den verwendeten Simulator bereitstellt. Dies ist z. B. die XML-Beschreibung.

Exemplar

→ Modellobjekt

Fehlersimulation

Eine alternative Verhaltensberechnung eines Bauteils, welche ein fehlerhaftes Verhalten des Bauteils nachbildet. Sie kann das Verhalten des Bauteils ersetzen oder verfälschen. Ausgelöst wird eine Fehlersimulation durch einen → Fehlertrigger (dynamisch) oder durch zeitlich statische Planung im → Testfall.

Fehlertrigger

Ereignis, zu dem auf eine → Fehlersimulation umgeschaltet wird. Dies können sowohl zeitliche Ereignisse als auch bestimmte Zustände sein.

Hardware-in-the-Loop (HiL)

Ein → Testbed, welches der zu testenden Software die → Zielhardware-Umgebung bietet, in der sie später auch ausgeführt wird. Das Verhalten der Umwelt wird simuliert, und die Signale werden über die elektrischen Schnittstellen der → Zielhardware in das zu testende System eingespeist bzw. ausgelesen. Die Simulation muss in diesem → Testbed echtzeitfähig sein. Alternativ lässt sich die reale Hardware auch durch ein → Labormuster ersetzen.

Inputs

Daten, welche in das zu testende System fließen. Dies können Nachrichten auf einem Bus, elektrische Spannungen und Ströme von Sensoren oder Aktuatoren und Interrupts sein.

Kalibration

Das Parametrieren eines → Simulationsmoduls eines Gerätes mit den Eigenschaften des spezifischen Geräts. Die Kalibrationsdaten werden aus der Spezifikation des Geräts gewonnen oder vermessen.

Kalibrations-Parameter

Einzelner Wert, mit dem sich eine Eigenschaft eines Geräts einstellen lässt. Siehe auch → Kalibration.

Konfigurationsmanagement (CM)

Prozess zur Versionierung von Daten, der es erlaubt, alte Versionen von Daten zugreifbar zu machen und Unterschiede zwischen einzelnen Versionen von Daten zu ermitteln. Daten können → Testfälle, → abstrakte Simulationsmodelle, → Simulationen usw. sein. SiLEST setzt hierfür die Programme Subversion (DLR) und StarTeam (IAV) ein.

Labormuster

Nachbau des technischen Systems, in dem die zu testende Software später eingesetzt werden soll. Durch das Labormuster werden nur die für den Test relevanten Einflussfaktoren in dem späteren System abgedeckt. In der Regel ist das Labormuster ein alternatives Steuergerät, welches ähnliche Eigenschaften wie die → Zielhardware hat.

Majorzyklus

Ein Vielfaches eines → Minorzyklus, mit dem standardmäßig mit der zu testenden Software zeitlich synchronisiert wird.

Minorzyklus

Kleinster → Simulationsschritt, in dem gerechnet wird. Der Minorzyklus bestimmt die feinste zeitliche Auflösung der Simulation. Er muss entsprechend den Erfordernissen der Simulation gewählt werden. Ein zu groß gewählter Minorzyklus kann zu Rechenungenauigkeiten führen, sofern in der Simulation eines dynamischen Systems Rückkopplungen enthalten sind. Ein zu klein gewählter Minorzyklus vergrößert die Simulationsdauer.

Modell

→ Simulationsmodell

Modell-Bibliothek

→ Modell-Repository

Model-in-the-Loop (MiL)

Bei diesem Test ist ein Modell der zu entwickelnden Softwarefunktionalität in eine Simulation eingebunden. Die spätere → Zielhardware wird hierbei noch nicht berücksichtigt. Ziel der Untersuchung ist der funktionale Nachweis der Algorithmen.

Modellobjekt

Ist ein plattform- bzw. simulatorspezifisches → Simulationsmodul, welches sich kalibriert und ausführbereit innerhalb einer → Simulationsumgebung befindet. Ein → Simulink-Template z. B. wird zu einem Modellobjekt, wenn es aus einem → Modell-Repository in die → Simulink-Umgebung geschoben wird.

Modell-Repository

Eine unter → Konfigurationsmanagement stehende Sammlung von → Templates und → Simulationsmodulen. Die → Templates und → Simulationsmodule sind recherchierbar.

Outputs

Daten, welche aus dem zu testenden System herausfließen. Dies können Nachrichten auf einem Bus sein oder Signale, welche an den Ausgangsports des Mikrocontrollers erzeugt werden.

Parameter

Sie lassen sich in → Kalibrations-Parameter und → Simulations-Parameter unterscheiden. Parameter werden vor der Durchführung eines Testfalls gesetzt und bleiben während der Durchführung konstant.

Processor-in-the-Loop (PiL)

Ein → Testbed, bei dem die → Zielhardware durch ein → Labormuster ersetzt wird. Im Gegensatz zur → HiL-Simulation erfolgt die Kopplung der Komponenten des → Testsystems nicht über die Original-Schnittstellen sondern über die Schnittstellen eines Entwicklungssystems.

Reportsystem

Dieses System generiert aus den während eines → Testlaufs erzeugten → Testergebnissen und Ausgaben einen → Testreport. In SiLEST werden für das Reportsystem FOP und eine spezielle XSLT-Transformation eingesetzt.

Repository

Verzeichnisstruktur oder Datenbank, die Objekte inklusive Änderungsinformationen enthält. Es wird vom → Konfigurationsmanagement erstellt.

Safety Integrity Level (SIL)

Einteilung von Systemkomponenten nach ihrer Gefährdungsstufe, um das Risiko für den sicherheitskritischen Ausfall einer Komponente zu quantifizieren. Das Safety Integrity Level ist normiert nach der IEC 61508.

Setup

Bezeichnet die Konfiguration eines Systems (→ Simulator oder Systemkomponente), die → Kalibration von Simulationsmodulen und das Setzen der Eingabedaten.

Simulation

Eine ausführbare Verknüpfung von einzelnen Modellen zur Beschreibung des Umweltverhaltens, welche in Form von Simulationsmodellen oder von Nachbauten der späteren Einsatzumgebung(z. B. Luftlagertisch oder Testfahrzeug) vorliegen. Das → Testbed hängt von der jeweiligen Methode der Simulation ab (→ MiL, → SiL, → PiL, → HiL).

Simulationsmodell

Ein Simulationsmodell ist innerhalb einer → Simulationsumgebung (wie z. B. Matlab / Simulink) ausführbar und besteht typischerweise aus mehreren → Simulationsmodulen.

Simulationsmodul

Teil eines → Simulationsmodells, das entweder als → abstraktes Template, → Template oder → Exemplar vorliegt. Es modelliert die für Fragestellung relevanten Eigenschaften des Systems oder der Umwelt, wobei andere Systemeigenschaften vernachlässigt werden.

Simulationsparameter

Einstellungen für die Simulation. Dies können zum Beispiel die Start- und Endzeiten eines Tests, die Zyklusweite eines → Simulationsschritts, der zu verwendende mathematische Löser, Defaultwerte für die → Fehlersimulation etc. sein.

Simulator

Die Plattform, auf der die Simulationen ausgeführt werden. Dies kann zum Beispiel Matlab / Simulink sein.

Simulationsschritt

Ein Zeitschritt der Simulation. Für einen Simulationsschritt ist zwischen → Minorzyklus und → Majorzyklus zu unterscheiden.

Simulationsumgebung

→ Simulator

Simulink-Template (Simulink-Klasse)

Ist ein Simulink-spezifisches, kalibriertes Template. Es ist Teil einer Simulink-Bibliothek.

Software-in-the-Loop (SiL)

Ein → Testbed, in dem das Verhalten der → Zielhardware und der Umwelt nachgebildet werden. Des Weiteren umfasst es die Softwarefunktionalität, die als Modell und / oder als Sourcecode vorliegt. Alle für den Test notwendigen Softwareschnittstellen werden angeboten. Die Signale werden über Schnittstellen des Betriebssystems oder die Hardware-Abstraktionsschicht (HAL) eingespeist bzw. ausgelesen. Die Simulation muss in diesem → Testbed nicht echtzeitfähig sein.

Systemkomponente

Ein einzelnes Gerät in einem eingebetteten System, zum Beispiel ein Sensor oder ein Aktuator.

Template (Modellklasse)

Ist ein → Simulationsmodul, welches in plattformunabhängiger (simulatorunabhängiger) Form in einem → Modell-Repository gespeichert ist. Es ist im Gegensatz zum → abstrakten Template kalibriert, z. B. ein Sensormodul eines ganz bestimmten Typs oder Herstellers (siehe auch → Simulink-Template).

Testablaufsteuerung (Test Sequence Control, TSC)

Zentrales Softwareprogramm zur automatischen Testdurchführung. Dabei wird eine → Testsuite aus dem → Testfall-Management geholt, im → Testbed ausgeführt, und die Ergebnisse werden an das → Testfall-Management zurückgeliefert.

Testbed

Die Umgebung, in der die Software für den Test eingebunden wird. Das Testbed stellt alles zur Verfügung, was für den ordnungsgemäßen Betrieb der Software während eines → Testlaufs benötigt wird.

Testcase

→ Testfall

Testcase Management (TCM)

→ Testfall-Management

Testergebnis

Resultat eines → Testlaufs für einen → Testfall. Es sind drei Resultate als Testergebnis möglich, *Passed*, *Failed*, *Inconclusive* oder *Aborted*. *Passed* bedeutet, dass das Verhalten der zu testenden Software dem definierten Verhalten entsprach, *Failed* wenn es diesem nicht entsprach. Bei dem Testergebnis *Aborted* ist ein Fehler während des → Testlaufs oder der Initialisierung aufgetreten, wodurch keine → Testauswertung durchgeführt werden konnte. Beim Testergebnis *Inconclusive* wurde der Testlauf nicht ausgeführt, da die Vorbedingungen des Testfalls nicht erfüllt wurden.

Testfall

Ein einzelner Test, welcher das System aus einem im Testfall definierten initialen Zustand über einen im Testfall definierten Zeitraum mit einer Reihe von externen Ereignissen oder einem Benutzerverhalten betreibt.

Testfall-Management

Prozess zum Erstellen, Prüfen, Ausführen, Sichern und Verwalten von → Testfällen bzw. → Testsuiten und deren Ergebnissen. In SiLEST werden hierfür die Programme Testmaster (DLR) und TestDirector (IAV) eingesetzt.

Testlauf

Ausführung einer → Testsuite.

Testreport

Eine Zusammenfassung der → Testergebnisse und ausgewählter Ausgaben eines einzelnen → Testlaufs. In SiLEST werden diese automatisch vom → Reportsystem generiert.

Teststatus

Zustand, den ein → Testfall annehmen kann. Neben den → Testergebnissen *Passed*, *Failed*, *Inconclusive* und *Aborted* ist dies *Pending*, *Running* und *Executed*. Den Teststatus *Pending* hat ein → Testfall, welcher zum Test angewiesen wurde, der aber noch nicht initialisiert ist. Ist ein → Testfall initialisiert, so ist der → Teststatus *Running*. Den

Teststatus *Executed* erhalten → Testfälle, welche bereits ausgeführt, aber nicht ausgewertet wurden.

Testsuite

Sammlung von mehreren → Testfällen oder weiterer Testsuites.

Testsystem

Eingebettetes System, das die zu testende Softwarefunktionalität enthält.

Testumgebung

Die Testumgebung umfasst die zu testende Software sowie das → Testbed.

Versionsverwaltung

→ Konfigurationsmanagement

Wrapper

Eine Softwareschnittstelle, welche die Schnittstellen eines einzubindenden → abstrakten Templates an die Schnittstellen des → Simulators anpasst.

Zielhardware

Reale Hardware, auf dem die zu entwickelnde Funktionalität in Form von Software ausgeführt wird. In SiLEST handelt es sich dabei um ein Prozessorboard (DLR) bzw. ein Motorsteuergerät (IAV).

Zustand

Die Eigenschaften, welche die Simulation und / oder das zu testende System zu einem Zeitpunkt besitzen.

Anhang C Definitionen, Akronyme und Abkürzungen

Dieser Anhang enthält ein Verzeichnis der in allen Dokumenten verwendeten Abkürzungen.

AP	Arbeitspaket
API	Application Programming Interface
ATML	Automatic Test Markup Language
Bird	Bispectral Infrared Detection
BMBF	Bundesministerium für Bildung und Forschung
BOSS	Bird Operating System by Sergio
CM	Konfigurationsmanagement
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
ECU	Engine Control Unit (auch: Electronic Control Unit)
FhG	Fraunhofer Gesellschaft
FMEA	Failure Modes and Effect Analysis
FOP	Formatting Objects Processor
GND	Ground (Masse, 0 V)
HAL	Hardware Abstraction Layer
HCCI	Homogenous Charge Compression Ignition
HiL	Hardware-in-the-Loop
IAV	Ingenieurgesellschaft Auto und Verkehr GmbH
I/O	Input / Output (Ein- / Ausgabe)
MDA	Model Driven Architecture
MiL	Model-in-the-Loop (nicht zu verwechseln mit Man-in-the-Loop)
OS	Operating System (Betriebssystem)
PiL	Processor-in-the-Loop
SALib	Sensors and Actuators Library
SIL	Safety Integrity Level
SiL	Software-in-the-Loop
SiLEST	Software-in-the-Loop for Embedded Software Test
SOAP	Simple Object Access Protocol (used by Webservices)
STEP	International Standard for the computer-interpretable representation and exchange of product data
SysML	Systems Modeling Language
TCG	Testcase Generator (auch Testfalleditor, TFE)
TCM	Testcase Management (Testfall-Management)

TFE	Testfalleditor (auch Testcase Generator, TCG)
TSC	Test Sequence Control (Testablaufsteuerung)
TTCN	Testing and Test Control Notation
TUB	Technische Universität Berlin
UART	Universal Asynchronous Receiver-Transmitter
UB	U_{BAT} (Positive Betriebsspannung)
UML	Unified Modeling Language
VCS	Version Control System (Versionsmanagement)
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Anhang D Ablaufdiagramme zum Testprozess

Dieser Anhang enthält alle Ablaufdiagramme des in Kapitel 2 vorgestellten Testprozess zum in the Loop Test. Die in den Ablaufdiagrammen verwendeten Prozesssymbole sind der Abbildung D-1 zu entnehmen.

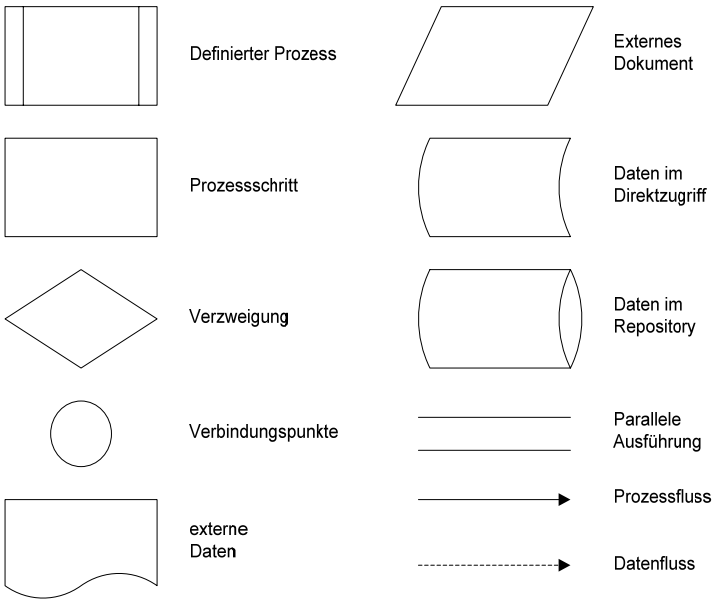

Abbildung D-1 Prozesssymbole

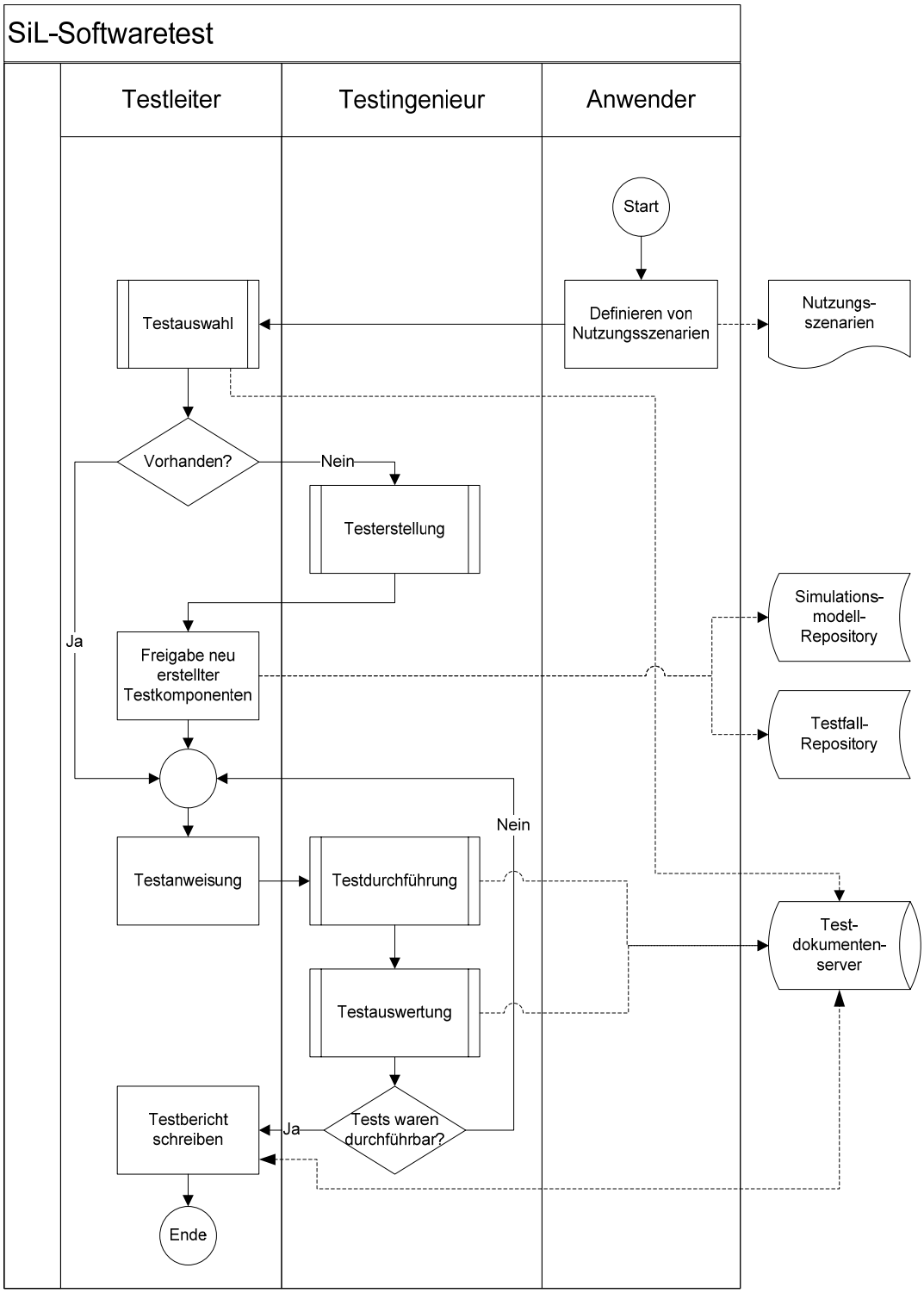

Abbildung D-2 Prozess SiL-Softwaretests

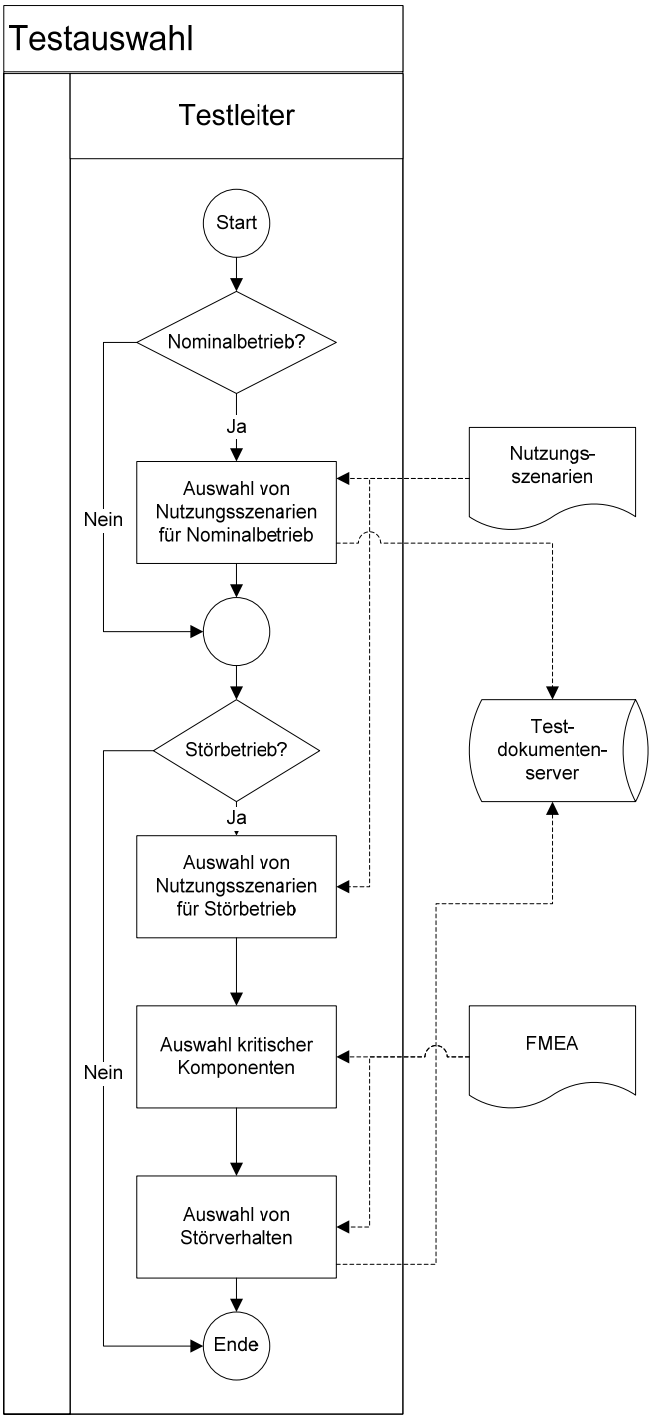

Abbildung D-3 Prozess Testauswahl

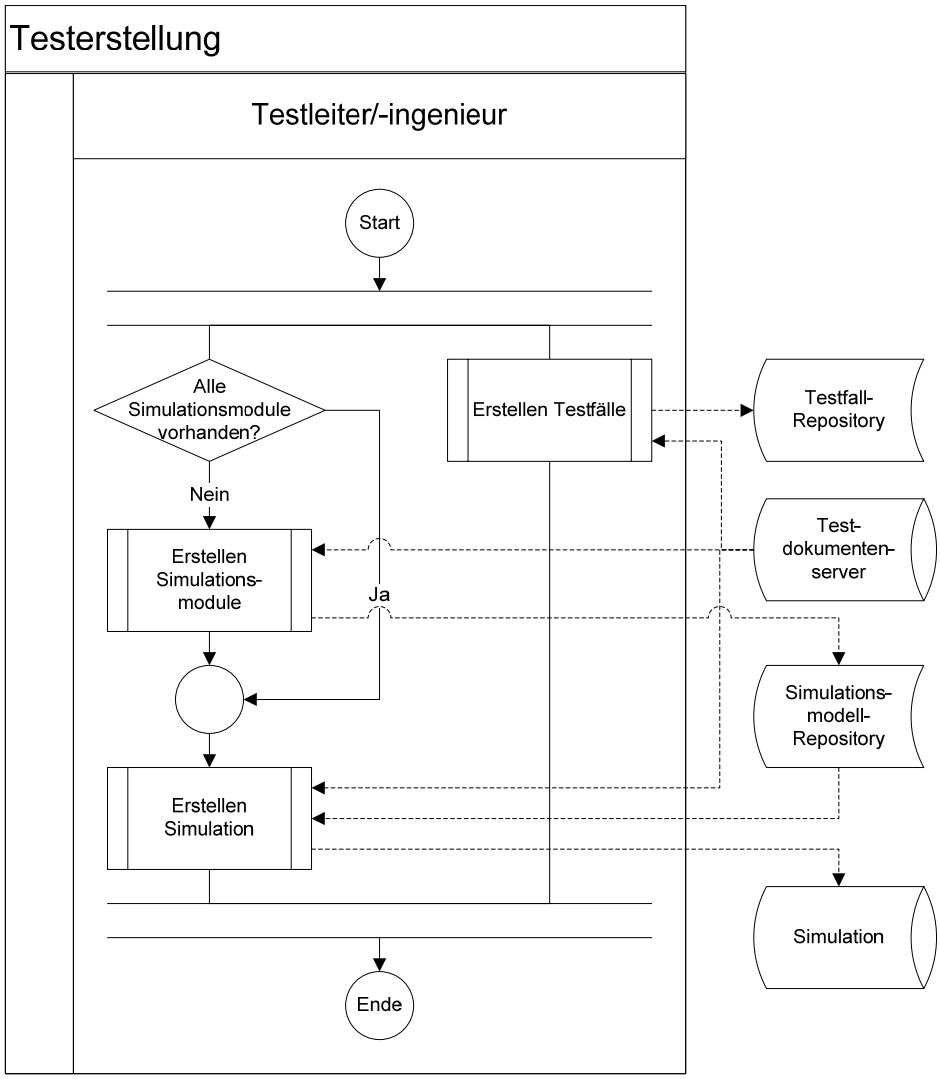

Abbildung D-4 Prozess Testerstellung

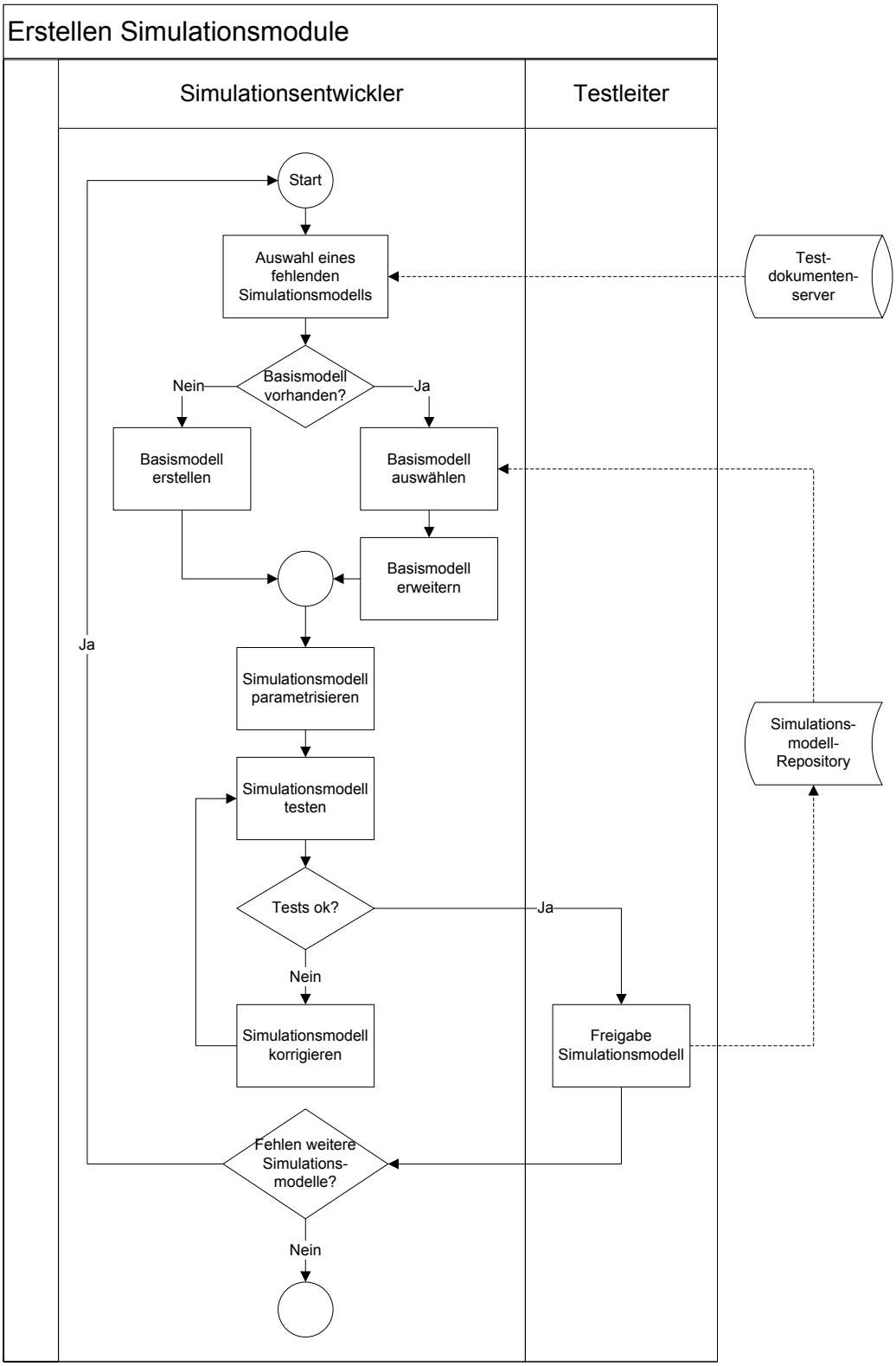

Abbildung D-5 Prozess Erstellen Simulationsmodule

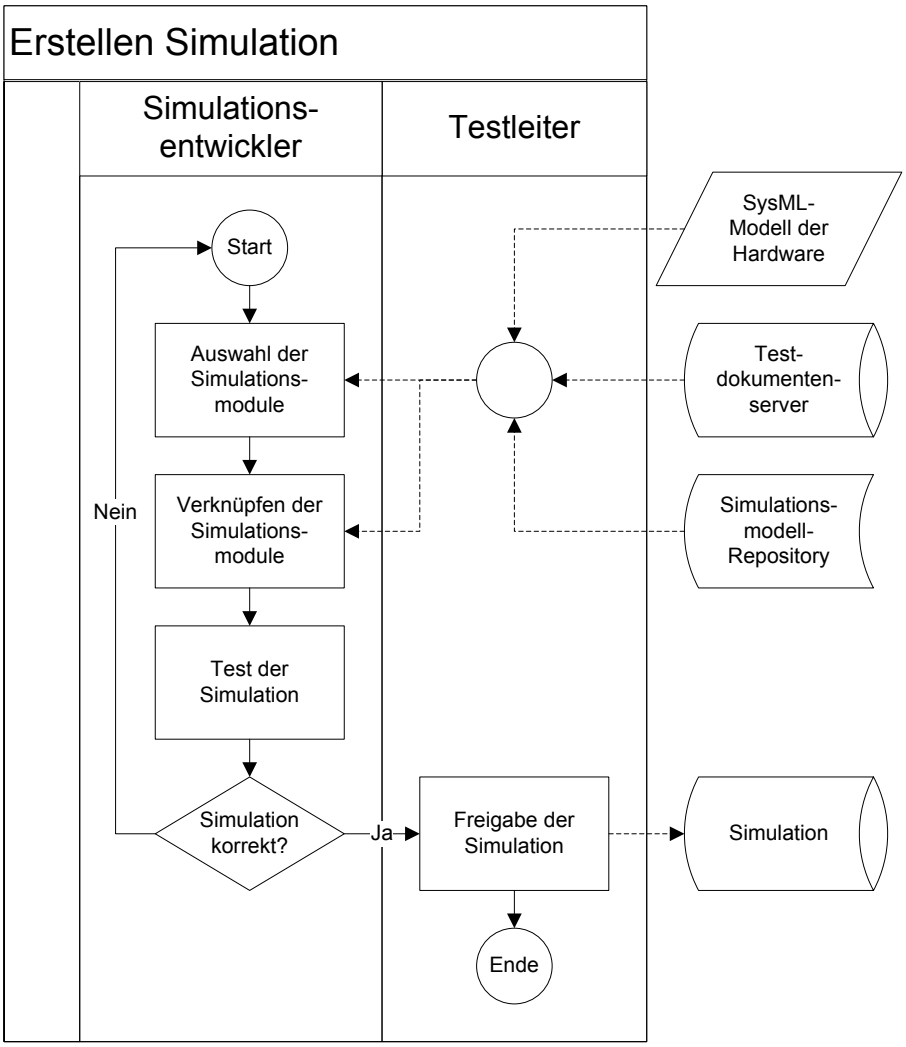

Abbildung D-6 Prozess Erstellen Simulation

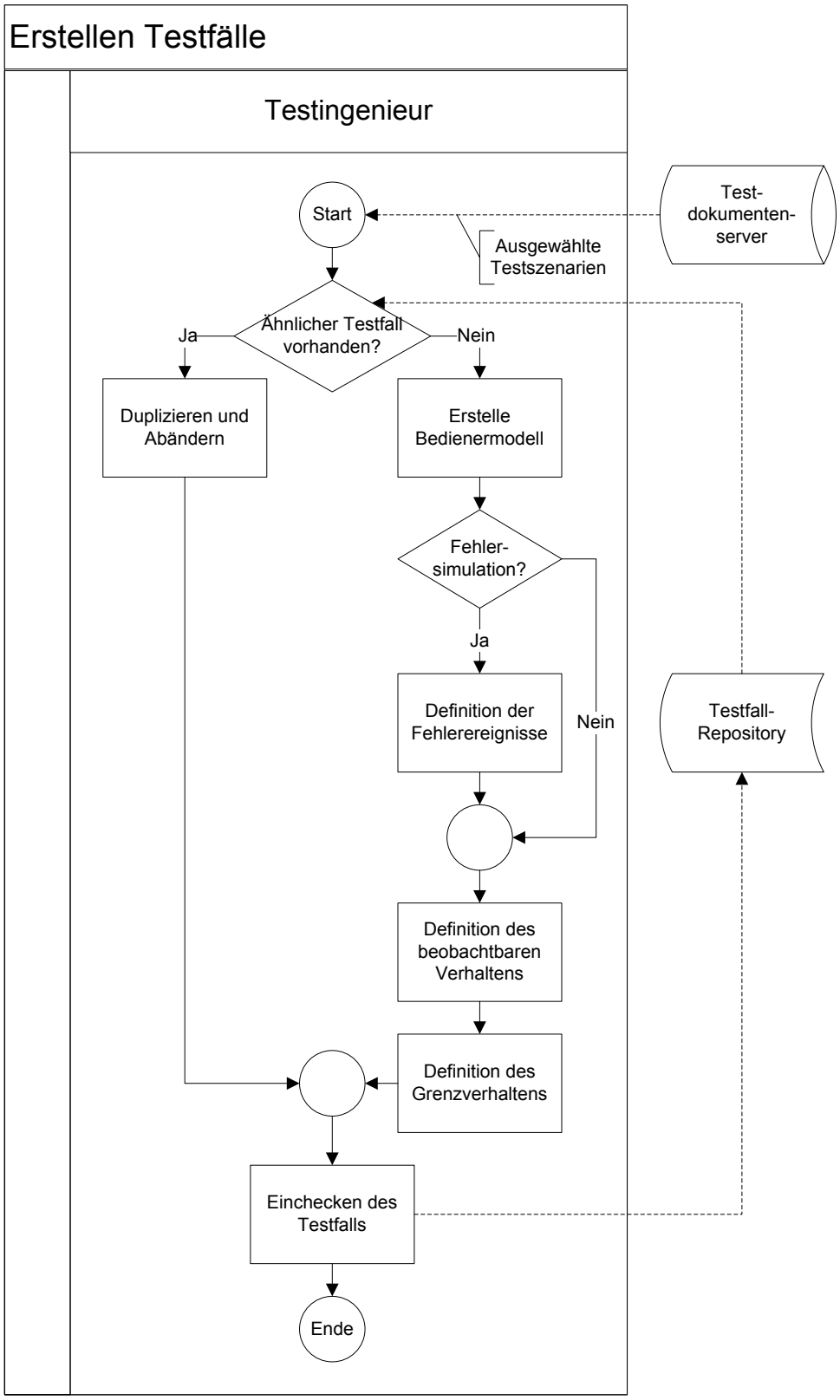

Abbildung D-7 Prozess Erstellen Testfälle

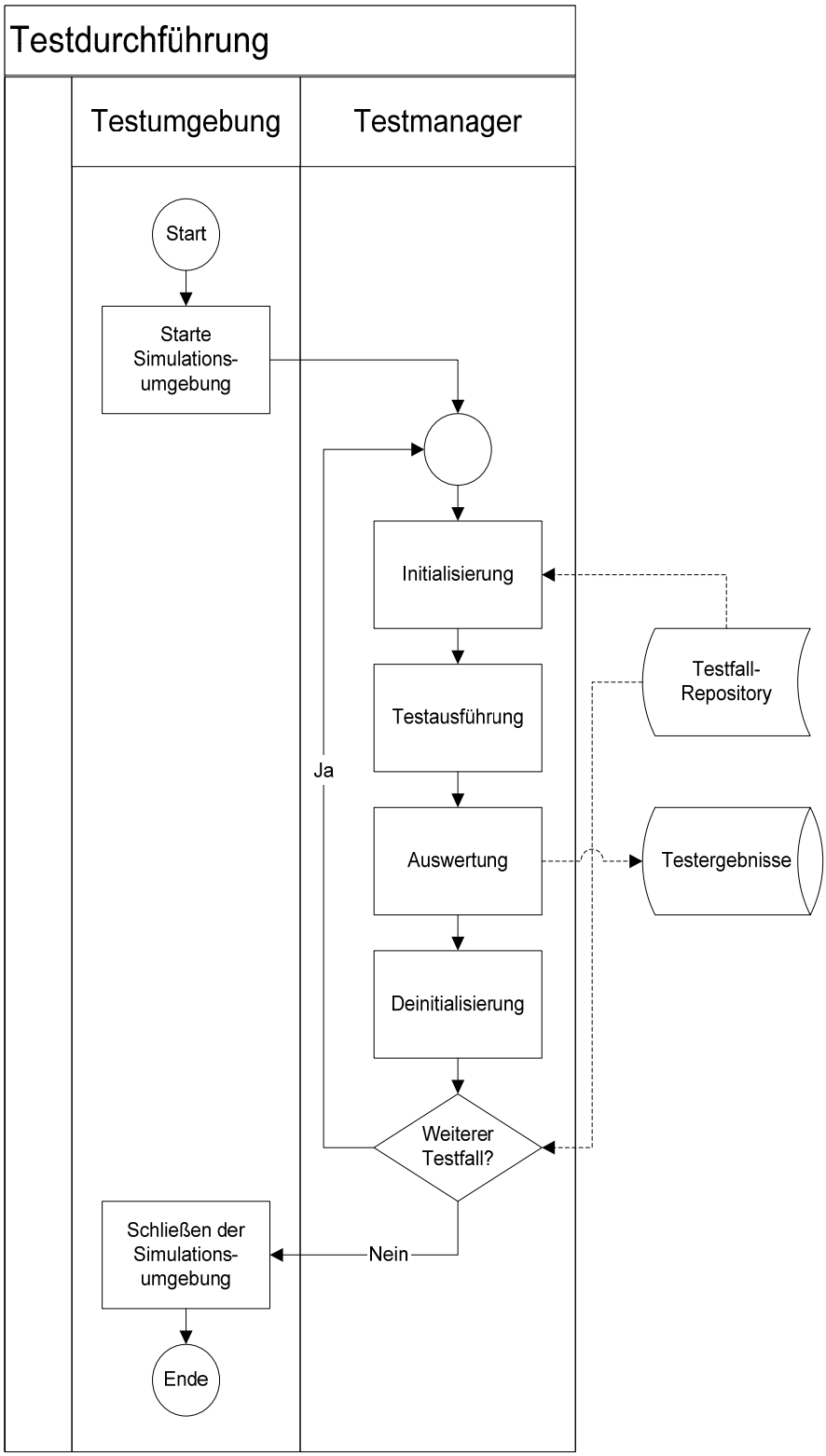

Abbildung D-8 Prozess Testdurchführung

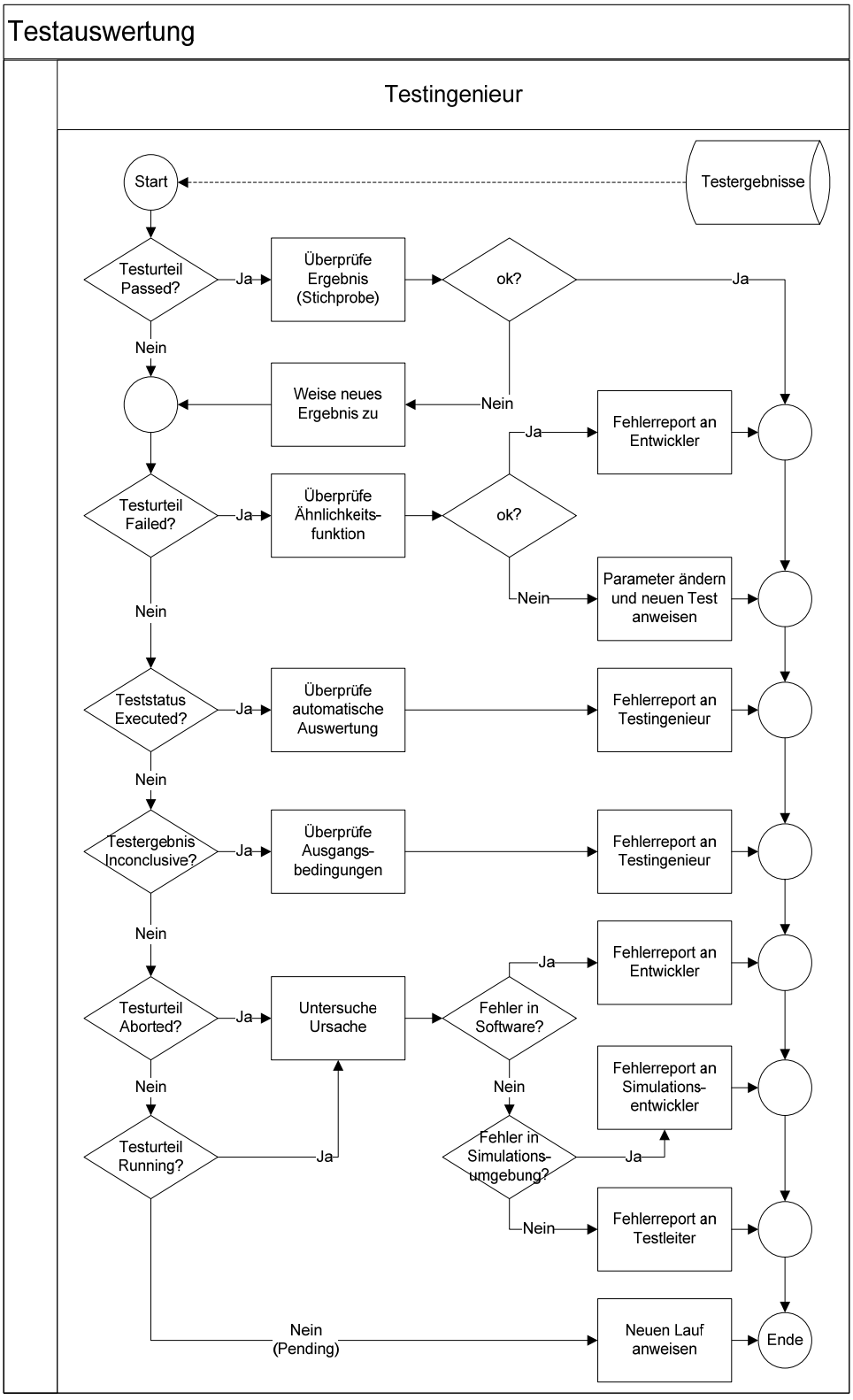

Abbildung D-9 Prozess Testauswertung