1 Einführung

Dieses Buch richtet sich an Personen, die in der Automobilindustrie Testaktivitäten für softwarebasierte Systeme planen, vorbereiten, durchführen oder beurteilen. Das Buch soll das Thema Automotive-Softwaretest möglichst allgemein darstellen, weshalb es sich nicht mit allen relevanten Spezialthemen befasst.

Für die vertiefende Behandlung einiger Spezialthemen gibt es eigene ISTQB®-Lehrpläne und zugehörige Bücher im dpunkt.verlag, beispielsweise [Linz 2016] zum Testen in der agilen Softwareentwicklung, [Simon et al. 2019] zum Testen der IT-Sicherheit und [Winter et al. 2016] zum modellbasierten Testen. Zum momentan sehr aktuellen Thema des Testens von Systemen, die auf künstlicher Intelligenz (KI) basieren, wie autonomen Fahrzeugen, ist ein ISTQB®-Lehrplan gerade im Entstehen.

1.1 Lehrpläne

ISTQB® CTFL Automotive Software Tester

Das Buch unterstützt bei der Vorbereitung auf die Zertifizierungsprüfung zum *ISTQB® Foundation Level Specialist – CTFL Automotive Software Tester* (CTFL-AuT), sowohl im Selbststudium als auch begleitend zu einer Schulung. Es deckt die Inhalte des Lehrplans in der zum Zeitpunkt des Erscheinens aktuellen Version 2.0.2 [ISTQB 2020] vollständig ab. Anhang E enthält eine Tabelle mit Querverweisen, wo im Buch die einzelnen Abschnitte des Lehrplans zum CTFL-AuT behandelt werden. Das Buch geht aber auch über den Lehrplan hinaus, um wichtiges Hintergrundwissen zu vermitteln sowie einzelne Aspekte zu vertiefen.

Vorbereitung Zertifizierungsprüfung

Für die Vorbereitung auf die Zertifizierungsprüfung muss auf jeden Fall neben dem Buch auch der Lehrplan durchgearbeitet werden, da die Prüfungsfragen aus dem Lehrplan abgeleitet sind, und der Lehrplan manche Themen etwas anders behandelt als das Buch. Abweichungen vom Lehrplan sind im Buch aber gekennzeichnet.

ISTQB® Certified Tester Foundation Level

Der CTFL-AuT baut auf dem Lehrplan zum ISTQB® Certified Tester Foundation Level (CTFL) [ISTQB 2018] auf und ergänzt diesen um automobilspezifische Inhalte. Daher erfordert eine Zertifizierung zum CTFL-AuT eine vorhandene Zertifizierung zum CTFL. Das Buch setzt im Wesentlichen die Kenntnisse der Inhalte des CTFL voraus, die beispielsweise im Buch »Basiswissen Softwaretest« [Spillner & Linz 2019] vertieft nachzulesen sind.

ISTQB®-Glossar

Die hier im Buch verwendeten Fachbegriffe des Testens basieren auf dem lehrplanübergreifenden ISTQB®-Glossar [ISTQB & GTB 2020]. Sollte ein testspezifischer Begriff nicht geläufig sein, findet sich dort die zugehörige Definition.

1.2 Übersicht über das Buch

Das Buch orientiert sich im Wesentlichen an der Struktur des Lehrplans zum CTFL-AuT (siehe Anhang E).

Grundlagen

Kapitel 2 geht auf die *Grundlagen* aus dem CTFL ein. Es skizziert die Grundprinzipien aus dem CTFL, die für das Verständnis des CTFL-AuT notwendig sind. Darüber hinaus beschreibt es den für die Automobilindustrie typischen Produktentstehungsprozess (PEP) und diskutiert die Mitwirkung der Tester im PEP, beispielsweise bei Freigaben.

Normen und Standards

Kapitel 3 umfasst die *Normen und Standards*, mit denen ein Softwaretester in der Automobilindustrie typischerweise in Kontakt kommt. Der besondere Fokus liegt auf Automotive SPICE, ISO 26262 und AUTOSAR. Dabei werden die Grundstrukturen der Normen und Standards erklärt sowie die Herausforderungen bzw. relevanten Anforderungen an den Tester erläutert. Im Vergleich zum Lehrplan finden sich im Buch mehr Informationen und Hintergründe zu den jeweiligen Normen und Standards. Zielsetzung des Kapitels ist es, dem Tester ausreichend Informationen an die Hand zu geben, sodass er fundiert bei den Diskussionen um die aus den Normen und Standards abgeleiteten Testanforderungen mitsprechen kann. Das Kapitel schließt mit einer Gegenüberstellung der Zielsetzung, der Teststufen und der Testverfahren und Testansätze in den Normen und Standards.

Virtuelle Testumgebungen

Kapitel 4 beschreibt die unterschiedlichen *virtuellen Testumgebungen*, die in der Automobilindustrie zum Einsatz kommen. Nach einer allgemeinen Einführung in Testumgebungen stellt es die Unterschiede zwischen Closed- und Open-Loop-Systemen dar. Danach vertieft es die Charakteristika virtueller Testumgebungen wie MiL (Model-in-the-Loop), SiL (Software-in-the-Loop) und HiL (Hardware-in-the-Loop). Abschließend erfolgt ein Vergleich der unterschiedlichen virtuellen Testumgebungen und ihrer Einsatzgebiete bei der Produktentwicklung.

Testansätze und Testverfahren

Kapitel 5 erklärt spezielle *Testansätze und Testverfahren*, die in der Automobilindustrie eingesetzt werden, und zeigt ihre Anwendung anhand von Beispielen auf. Im Softwaretest sind Testansätze von genereller Natur und repräsentieren prinzipielle Vorgehensweisen und Theorien zur Lösung von Testaufgaben. Im Gegensatz dazu sind Testverfahren konkrete Vorgehensweisen und Techniken zur Lösung von Testaufgaben. Hierzu zählen Verfahren sowohl für statische als auch für dynamische Tests. Bei den statischen Testverfahren liegt der Fokus auf der Codeanalyse nach MISRA-C sowie auf den Qualitätsmerkmalen für das Review von Anforderungen. Bei den dynamischen Testverfahren wird insbesondere auf die Testverfahren eingegangen, die die ISO 26262 empfiehlt. Hierzu gehören beispielsweise modifizierte Bedingungs-/Entscheidungstests (MC/DC-Test), Back-to-Back-Tests und Fehlereinfügungstests. Darüber hinaus zeigt Kapitel 5 auf, wie eine Auswahl der Testverfahren in einem konkreten Projektkontext erfolgen kann.

Anhang

Beim Schreiben des Buches hat der eine oder andere Autor Inhalte entwickelt, die die Hauptkapitel des Buches sprengen würden. Da es sich aber um wertvolle Informationen handelt, sind diese Inhalte im Anhang zu finden. Beispielsweise die Zusammenfassung aller Bände der ISO 26262, inkl. einer Aussage zu den Neuerungen in der Version von 2018.

Beispiele

Um dem Leser das Verständnis für die Inhalte zu erleichtern, haben wir das Projekt *ULV* des Fahrzeugherstellers Bavarian Electric Cars (BEC) als durchgängiges Beispiel eingeführt (siehe Abschnitt 1.3). Wo möglich und sinnvoll, stellen die einzelnen Abschnitte im Buch einen Bezug zu dem Beispielprojekt her. Dies soll dem Leser das Verständnis der Inhalte und den Transfer auf den Projektalltag erleichtern.

1.3 Einführung in das Beispielprojekt

Ein durchgängiges Beispiel soll in diesem Buch zum besseren Verständnis der Inhalte beitragen. Alle Beispiele sind im Buch grau hinterlegt:

Beispiel Tempomat

Bei der zu entwickelnden Tempomatfunktion handelt es sich um einen reinen Geschwindigkeitsregler, nicht um einen Abstandsregeltempomaten.

1.3.1 Projekthintergrund

Systemlieferant Eddison Electronics

Dreh- und Angelpunkt des Beispielprojekts ist die Firma *Eddison Electronics*. Eddison Electronics ist ein mittelständisches Unternehmen, das sich auf elektrische Antriebe für Kraftfahrzeuge spezialisiert hat. Der Fahrzeughersteller Bavarian Electric Cars (BEC) hat Eddison Electronics beauftragt, den elektrischen Antriebsstrang für ihr neues Stadtfahrzeug *Urban Lite Vehicle (ULV)* zu entwickeln. Eddison Electronics hat in diesem Projekt die Rolle des Systemlieferanten (Tier-1) und entwickelt für BEC den kompletten elektrischen Antrieb – einschließlich damit verbundener Funktionen, wie Antriebsschlupfkontrolle oder Tempomat. Abbildung 1–1 zeigt, für welche Umfänge des Fahrzeugs Eddison Electronics zuständig ist.

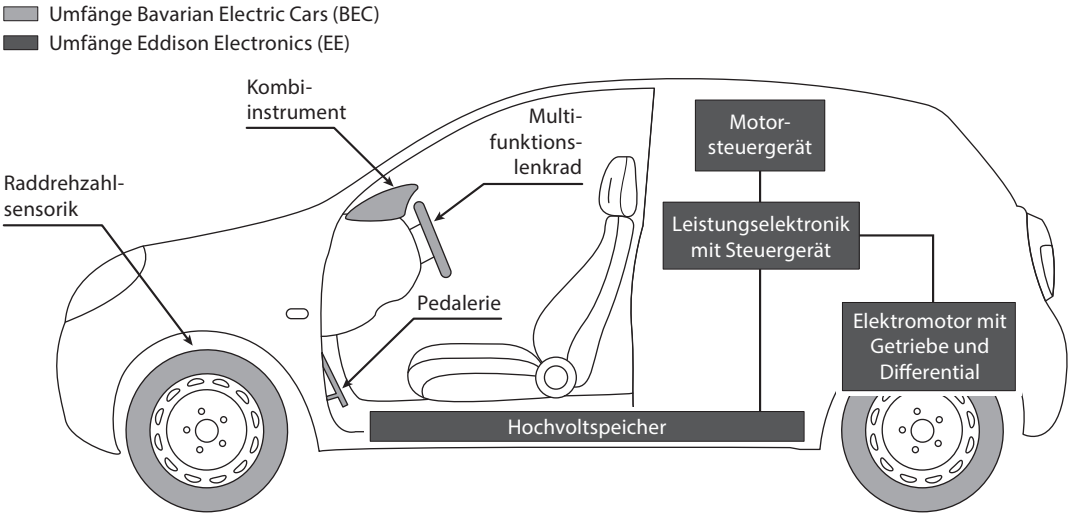


Abb. 1–1
Umfänge der Firma Eddison Electronics (EE) am Projekt ULV

1.3.2 Aufbau des Systems

Der Antriebsstrang des *ULV* besteht aus dem E/E-Antriebssystem und den rein mechanischen Umfängen, wie Getriebe und Differenzial. Das E/E-Antriebssystem umfasst den Elektromotor, die Leistungselektronik (Inverter), den Hochvoltspeicher und die Steuergeräte für Leistungselektronik und Elektromotor. Sie sind in das elektrische Bordnetz und die Bussysteme des Fahrzeugs (u.a. CAN) eingebunden. Die Steuergeräte sind moderne Plattformsteuergeräte. Hardware und Software werden von Eddison Electronics selbst entwickelt.

Abbildung 1–2 zeigt die Struktur des Antriebsstrangs bis hin zu den Softwarekomponenten, aus denen die Software der beiden Steuergeräte besteht. Für die Beispiele in diesem Buch sind speziell die Strukturelemente *mit* Softwareanteil relevant. Sie sind in der Abbildung grau eingefärbt.

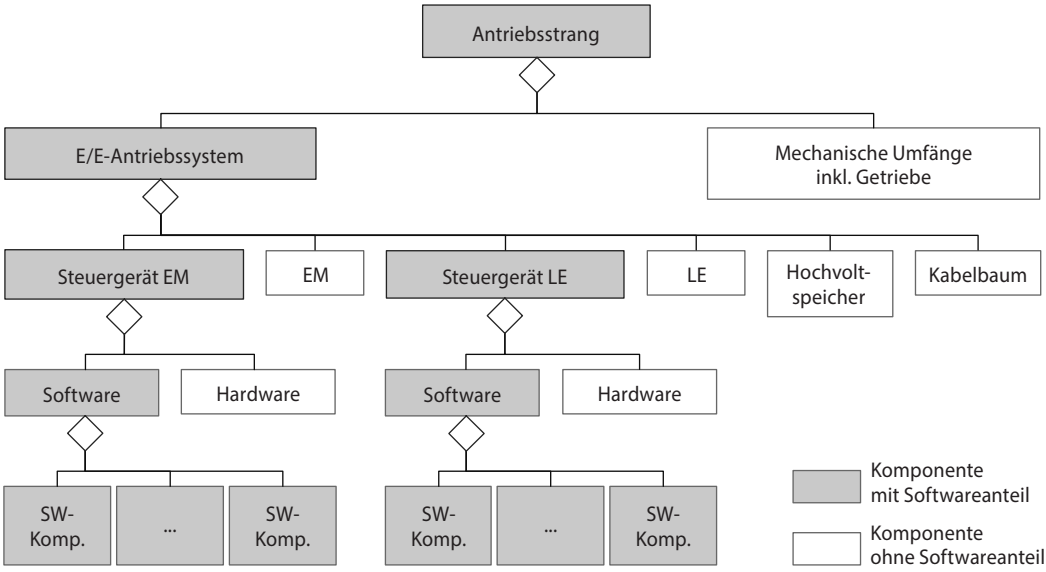


Abb. 1–2 *Struktur des elektrischen Antriebsstrangs*

Feature Tempomat

Da der gesamte elektrische Antriebsstrang sehr umfangreich ist, konzentrieren sich die Beispiele im Folgenden auf das Feature *Tempomat*. BEC hat den Tempomaten zu Projektbeginn grob beschrieben:

- Es handelt sich um einen reinen Geschwindigkeitsregler, nicht um einen Abstandsregeltempomaten.
- Die Geschwindigkeitsregelung erfolgt ausschließlich über den elektrischen Antrieb. Ein Bremseingriff durch den Tempomaten ist nicht vorgesehen.
- Der Fahrer kann den Tempomaten über das Multifunktionslenkrad aktivieren und deaktivieren. Beim Aktivieren stellt er die Wunschgeschwindigkeit ein. Während der Tempomat aktiv ist, kann er die Wunschgeschwindigkeit ändern.
- Eine Betätigung des Fahrpedals über die aktuell eingestellte Wunschgeschwindigkeit hinaus (z.B. zum Beschleunigen) unterbricht die Geschwindigkeitsregelung des Tempomaten. Sobald die Fahrpedalstellung wieder unter diesen Wert fällt, nimmt der Tempomat die Geschwindigkeitsregelung wieder auf.

- Eine Betätigung der Bremse durch den Fahrer deaktiviert den Tempomaten sofort.
- Aus Komfortgründen ist die Beschleunigung durch den Tempomaten auf 4 m/s^2 limitiert.

Eddison Electronics hat anhand der Beschreibung von BEC eine Systemanforderungsspezifikation verfasst (siehe Anhang D) und auf dieser Basis eine funktionale Systemarchitektur erarbeitet – einschließlich verfeinerter Anforderungen an die Systembestandteile (ebenfalls im gleichen Anhang). Abbildung 1–3 zeigt die funktionale Architektur des Features Tempomat.

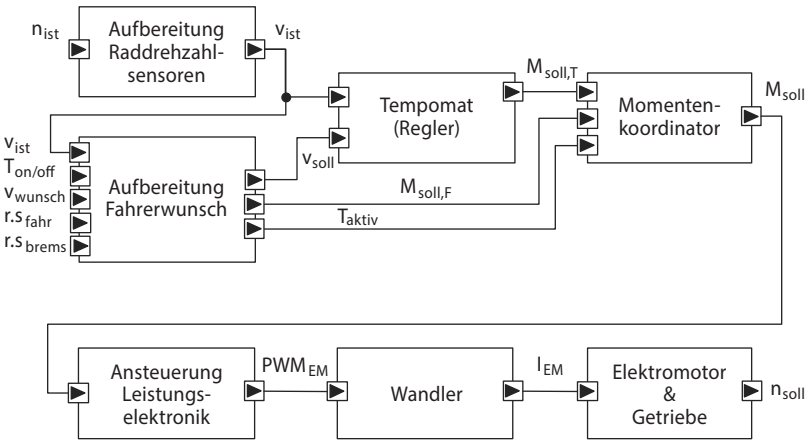


Abb. 1–3
Funktionale Architektur der Tempomatfunktion

Das Feature Tempomat besteht aus sieben Architekturelementen:

- *Aufbereitung Raddrehzahlsensoren* wertet die Messwerte der Drehzahlsensoren an den vier Rädern aus und berechnet daraus die Ist-Geschwindigkeit des Fahrzeugs.
- *Aufbereitung Fahrerwunsch* bereitet den Fahrerwunsch (z.B. die Wunschgeschwindigkeit und den Status des Bremspedals) auf. Dieses Element wertet aus, ob der Tempomat aktiv ist, und liefert die Soll-Geschwindigkeit für den Tempomaten.
- *Tempomat (Regler)* berechnet auf Basis von Soll- und Ist-Geschwindigkeit das Soll-Drehmoment des Motors.
- *Momentenkoordinator* ermittelt das richtige Soll-Drehmoment, abhängig davon, ob der Tempomat aktiv ist oder nicht.
- *Ansteuerung Leistungselektronik* leitet aus dem Soll-Drehmoment die nötige Ansteuerung des Elektromotors ab.
- Der *Wandler* setzt die Ansteuerung in einen konkreten Stromfluss um.
- Der *Elektromotor* treibt über ein integriertes Getriebe die Räder an.

Die kleinen Kästen an den Rändern der Architekturelemente in Abbildung 1–3 stellen die Schnittstellen dar. Das Dreieck im Kästchen zeigt die Signalrichtung an und markiert eine Schnittstelle als Sender bzw. Empfänger von Daten. Die an den Schnittstellen übertragenen Signale sind in Tabelle 1–1 genauer beschrieben.

Tab. 1–1
Signale der Tempomatfunktion

Signal	Beschreibung
I_{EM}	Stromstärke des Stromflusses zum Elektromotor
M_{soll}	Konsolidiertes Soll-Drehmoment
$M_{soll,F}$	Soll-Drehmoment auf Basis des Fahrpedals
$M_{soll,T}$	Soll-Drehmoment auf Basis des Tempomatreglers
n_{ist}	Werte der vier Raddrehzahlsensoren
n_{soll}	Raddrehzahl nach Elektromotor und Getriebe
PWM_{EM}	Pulsweitenmodulationssignale zur Ansteuerung des Elektromotors
$r.s_{brems}$	Stellung des Bremspedals
$r.s_{fahr}$	Stellung des Fahrpedals
T_{aktiv}	Tatsächlicher Aktivitätsstatus des Tempomaten
$T_{on/off}$	Fahrerwunsch zum Aktivierungsstatus des Tempomaten (an/aus)
v_{ist}	Aktuelle Geschwindigkeit des Fahrzeugs
v_{soll}	Soll-Geschwindigkeit des Fahrzeugs für den Tempomaten
v_{wunsch}	Wunschgeschwindigkeit des Fahrers für den Tempomaten

1.3.3 Einzuhaltende Standards

Entwicklung nach Stand der Technik

BEC erwartet, dass Eddison Electronics nach dem Stand der Technik entwickelt, der unter anderem durch Normen und Standards definiert ist. Hier sind drei für die Fahrzeugentwicklung zentrale Standards hervorgehoben, auf die Kapitel 3 ausführlicher eingeht.

- Der Entwicklungsprozess bei Eddison Electronics muss konform zu Automotive SPICE (siehe Abschnitt 3.1) sein. BEC hat sich auf den VDA-Scope festgelegt und erwartet, dass im Projekt für diese Prozesse die Fähigkeitsstufe 2 erreicht wird. Der VDA-Scope ist im Anhang in Abschnitt B.2 dargestellt.
- Da es sich bei dem elektrischen Antriebsstrang des *ULV* um ein sicherheitskritisches System handelt, muss Eddison Electronics bei der Entwicklung die Anforderungen der ISO 26262 berücksichtigen (siehe Abschnitt 3.2).
- Bei der Entwicklung der Steuerungssoftware muss Eddison Electronics konform zum AUTOSAR-Standard (siehe Abschnitt 3.3) vorgehen.

Darüber hinaus gibt es noch viele weitere Normen und Standards, die für das Projekt relevant sind. Kapitel 3 führt einige davon auf.

1.3.4 Beteiligte Personen

Die folgenden Personen sind bei Eddison Electronics im Projekt tätig:

- **Karsten, Kaufmännischer Leiter**
  Karsten ist neben seinen kaufmännischen Aufgaben auch für die Steuerung und Überwachung der Zulieferer zuständig (Supplier Monitoring).
- **Petra, Projektleiterin**
  Petra ist die Gesamtprojektleiterin für das Projekt *ULV* und zentrale Ansprechpartnerin für BEC.
- **Lars, Leiter der Entwicklungsabteilung**
  Lars ist in seiner Funktion als Entwicklungsleiter auch Process Owner für den Produktentwicklungsprozess bei Eddison Electronics.
- **Thorsten, Teilprojektleiter**
  Thorsten verantwortet das Teilprojekt *Tempomat*.
- **Stefan, Safety Manager**
  Stefan ist u.a. für das Sicherheitskonzept der Tempomatfunktion verantwortlich. Bei seiner Arbeit wird er von einem Expertenteam unterstützt.
- **Quentin, Qualitätsmanager**
  Quentin ist für die Qualitätssicherung, das Konfigurationsmanagement und das Änderungsmanagement verantwortlich. Er ist Provisional ASPICE-Assessor und unterstützt das Projekt bei der Vorbereitung des Assessments.
- **Thomas, Testmanager**
  Thomas verantwortet alle Testaktivitäten im Projekt bei Eddison Electronics. Da Eddison Electronics einen Teil der Testaktivitäten an externe Dienstleister vergeben hat, koordiniert er auch die Aktivitäten bei den Dienstleistern.
- **Tim, Softwaretester**
  Tim führt die Tests für die Tempomatfunktion durch.
- **Erika, Entwicklerin**
  Erika entwickelt die Software der Tempomatfunktion.
- **Simon, System- und Softwarearchitekt**
  Simon ist als Chefarchitekt für die Plattformarchitektur verantwortlich und hat die konkrete System- und Softwarearchitektur für den Antriebsstrang im Projekt *ULV* abgeleitet.

- **Rolf, Anforderungsmanager**
  Rolf ist für das Requirements Engineering und das Requirements Management verantwortlich. Er ist Autor der Anforderungsspezifikation des Antriebsstrangs (siehe Anhang D).
- **Rudi, Releasemanager**
  Rudi ist für die zeitliche und inhaltliche Aussteuerung der Releases zuständig.

4 Virtuelle Testumgebungen

Dieses Kapitel stellt spezielle Testumgebungen vor, die beim Testen in der Automobilindustrie besonders häufig zum Einsatz kommen, insbesondere:

- Model-in-the-Loop-(MiL-)Testumgebung
- Software-in-the-Loop-(SiL-)Testumgebung
- Hardware-in-the-Loop-(HiL-)Testumgebung

Außerdem zeigt es auf, für welche Einsatzzwecke sich die unterschiedlichen Testumgebungen eignen und für welche nicht.

4.1 Grundlagen

Im Verlauf der Entwicklung eines automobilen (Teil-)Systems entstehen in den einzelnen Entwicklungsschritten unterschiedliche Arbeitsprodukte, beispielsweise:

- Funktionsmodelle
- Softwarekomponenten
- Integrierte Software
- Steuergeräteplatinen
- Einzelne Steuergeräte
- Verbünde mehrerer Steuergeräte
- Gesamtes E/E-System des Fahrzeugs
- Fahrzeug

Um Fehler möglichst früh zu finden, prüft der Tester nicht nur das E/E-System oder das Fahrzeug als Ganzes, da diese erst spät in der Entwicklung vollständig testbar sind. Er testet auch Arbeitsprodukte wie Softwarekomponenten, die frühzeitig in der Entwicklung für den Test bereitstehen. Dabei sollen die Testergebnisse möglichst realistische Rück-

schlüsse auf das spätere Verhalten des jeweiligen Testobjekts im Fahrzeug erlauben. Das stellt besondere Anforderungen an die unterschiedlichen Testumgebungen.

Virtuelle Testumgebungen

Der Test von Arbeitsprodukten erfordert spezialisierte Testumgebungen, die die zur Ausführung benötigten, aber fehlenden Teile des E/E-Systems eines Fahrzeugs simulieren. Wegen des Simulationsanteils heißen sie virtuelle Testumgebungen.[1] Virtuelle Testumgebungen unterstützen die Simulation besonderer Fehlersituationen, die sich beim Test des E/E-Systems ansonsten nicht oder nur mühsam nachstellen lassen, beispielsweise Kurzschlüsse auf der Platine eines Steuergeräts oder Kabelbrüche im Kabelbaum. Schließlich muss eine virtuelle Testumgebung häufig auch die Umgebung des E/E-Systems simulieren, beispielsweise um die Regelstrecke eines Reglers im Testobjekt korrekt abzubilden. Die Regelstrecke bindet oft die Umgebung des E/E-Systems mit ein. Bei einem Abstandsregeltempomaten simuliert beispielsweise das Umgebungsmodell unter anderem das Verhalten des eigenen Fahrzeugs auf der Straße sowie vorausfahrende Fahrzeuge.

Beispiel Tempomat

Die Entwicklerin Erika programmiert den Code für die SW-C *Aufbereitung Raddrehzahlsensoren* (zu den SW-Cs siehe Abschnitt 3.3.3). Gemäß der Teststrategie des Projekts sollen die Entwickler selbst ihren Code einem Softwarekomponententest unterziehen. Sobald Erika einen testbaren Stand der SW-C erreicht hat, führt sie den zugehörigen Softwarekomponententest in einer Testumgebung auf ihrem Entwicklungsrechner durch. Für den Test benötigt sie weder die anderen SW-Cs noch die Steuergerätehardware oder das Fahrzeug. Der Test kann bereits gut die Funktionalität der SW-C prüfen. Allerdings sind Aussagen zum Echtzeitverhalten im Fahrzeug nur eingeschränkt möglich, da die Software im Test nicht auf der Zielhardware läuft und noch keine Effekte im Zusammenspiel mit der übrigen Software des Steuergeräts beobachtbar sind.

Testumgebung

Der Tester bettet das Testobjekt in eine Testumgebung ein, um es ausführen zu können (siehe Abb. 4–1). Dabei kann er die Eingänge des Testobjekts stimulieren (Point of Control) und seine Ausgänge beobachten (Point of Observation). Die Testumgebung nutzt die Zugangspunkte des Testobjekts, beispielsweise einen CAN-Transceiver, um das Testobjekt zu stimulieren und zu beobachten. In Abbildung 4–1 sind

1. Virtuelle Testumgebungen sind nicht zu verwechseln mit *virtualisierten* Testumgebungen. Virtualisierte Testumgebungen laufen in einer virtuellen Maschine auf einem Rechner, beispielsweise in einer Cloud.

die Zugangspunkte als weiße Quadrate dargestellt. Die Zugangspunkte sind typischerweise externe Schnittstellen. Sie können sich aber auch innerhalb des Testobjekts befinden, was insbesondere für interne Stimulation und internes Monitoring nützlich ist, z.B. zum Setzen oder Auslesen des Werts einer Variablen in der Software.

Der Testrahmen ist der Teil der Testumgebung, der direkt mit dem Testobjekt über dessen Zugangspunkte interagiert. Für die Steuerung der Testdurchführung ist die Systemsteuerung zuständig, die über den Testrahmen das Testobjekt stimuliert. Die Ausgaben des Testobjekts bekommt die Systemsteuerung vom Testrahmen in aufbereiteter Form übermittelt. Daher ist eine Kommunikationsschnittstelle zwischen dem Testrahmen und der Systemsteuerung nötig.

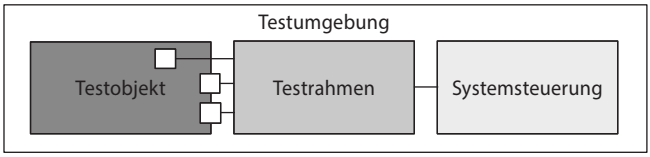


Abb. 4–1
Aufbau Testumgebung

4.1.1 Testobjekt

Um eine passende Testumgebung spezifizieren zu können, benötigt der Tester umfassende Informationen zu den Eigenschaften des Testobjekts. Besonders wichtig sind die Schnittstellen des Testobjekts nach außen, da diese als Zugangspunkte nutzbar sind. Ein Steuergerät hat typischerweise analoge und digitale Anschlüsse (Eingänge und Ausgänge) sowie Schnittstellen zu Bussystemen und Diagnoseschnittstellen, aber auch Testschnittstellen. Die Schnittstellen eines Steuergeräts sind überwiegend über elektrische Pins am Steuergerät realisiert. In seltenen Fällen hat ein Steuergerät auch Funkschnittstellen, z.B. WLAN oder Mobilfunk.

Die Anforderungen an die Testumgebung eines Steuergeräts lassen sich beispielsweise aus folgenden Quellen ableiten:

- Kommunikationsinformationen, d.h. über Bussysteme versendete und empfangene Nachrichten. Typischerweise gibt es eine Datenbank pro Bussystem, beispielsweise gemäß dem ASAM-Standard MCD-2 NET [ASAM MCD-2 NET].
- Protokollinformationen, d.h. Beschreibungen der bei der Kommunikation verwendeten Protokolle zur Übertragung einzelner Bits bis hin zur Übertragung ganzer Nachrichten. Typischerweise sind diese in Standards beschrieben. Beispiele sind Busprotokolle wie CAN [ISO 11898] oder LIN [ISO 17987], Diagnoseprotokolle wie UDS [ISO 14229] sowie Mess-/Kalibrierprotokolle wie XCP [ASAM MCD-1 XCP].

- Signalinformationen, d.h. über Busnachrichten oder andere Anschlüsse empfangene oder gesendete Daten, mit Informationen zur Codierung/Decodierung der Signale. Bei einem Bussystem sind die Signalinformationen oft in der Datenbank für die Kommunikationsinformationen integriert, da eine Nachricht ein oder mehrere Signale enthält.
- Diagnoseinformationen, d.h. typischerweise über ein Bussystem oder eine spezielle Diagnoseschnittstelle empfangene und gesendete Diagnosenachrichten und deren Bedeutung. Typischerweise gibt es eine Datenbank pro Steuergerät, beispielsweise gemäß dem ASAM-Standard MCD-2 D [ASAM MCD-2 D].
- Pinning, d.h. die Liste der Pins des Steuergeräts mit Zweck und Beschaltungsinformationen zu den einzelnen Pins. Typischerweise gibt es eine Datenbank pro Steuergerät.
- Testschnittstellen, d.h. spezielle Schnittstellen für den Test, um beispielsweise Speicherinhalte auszulesen oder zu verändern. Typischerweise gibt es eine Datenbank pro Steuergerät, beispielsweise gemäß dem ASAM-Standard MCD-2 MC [ASAM MCD-2 MC].

AUTOSAR hat eine Möglichkeit geschaffen, Informationen zum E/E-System, den Steuergeräten, den Bussystemen und den Applikationssoftwarekomponenten (SW-Cs) zu dokumentieren und in einer XML-Datei (ARXML) abzulegen [AUTOSAR 2019d]. Testwerkzeuge können bei AUTOSAR-Systemen daher viele der oben aufgezählten Informationen der ARXML-Datei entnehmen.

4.1.2 Testrahmen

Der Testrahmen der Testumgebung zerfällt in drei Bestandteile (siehe Abb. 4–2):

- Schnittstelle zum Testobjekt
- Schnittstelle zur Systemsteuerung
- Automationskern

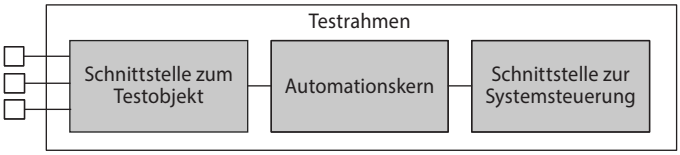


Abb. 4–2
Aufbau Testrahmen

Schnittstelle zum Testobjekt

Die Schnittstelle zum Testobjekt realisiert den Zugriff auf die Zugangspunkte durch passende Konnektoren. Bei Eingängen wie den Eingangspins eines Steuergeräts erzeugt die Schnittstelle elektrische Signale, um beispielsweise Sensorwerte eines dort normalerweise angeschlossenen Sensors zu simulieren. Bei Ausgängen wie den Ausgangspins eines Steuergeräts liest die Schnittstelle die elektrischen Signale aus und interpretiert diese gemäß dem Typ des Zugangspunkts, beispielsweise Pulsweitenmodulation (PWM). Busschnittstellen eines Steuergeräts sind sowohl Eingänge als auch Ausgänge.

Schnittstelle zur Systemsteuerung

Die Schnittstelle zur Systemsteuerung steuert und parametriert den Automationskern gemäß den Vorgaben der Systemsteuerung. Außerdem zeichnet sie die vom Automationskern bereitgestellten Messwerte auf und gibt diese an die Systemsteuerung weiter.

Automationskern

Der Automationskern des Testrahmens hat zwei wesentliche Aufgaben: die äußere Logik und die Testautomation. Die äußere Logik stellt sicher, dass die vom Testobjekt erwarteten Zusammenhänge zwischen Eingängen und Ausgängen (einschließlich Busnachrichten) gegeben sind. Die äußere Logik kann beispielsweise bei einem Busanschluss durch eine Restbussimulation realisiert werden, oder bei einem Regler mit Sensor und Aktuator durch ein Umgebungsmodell.

Abbildung 4–3 zeigt den Aufbau des Automationskerns. Schnittstellenmodell, Streckenmodell, Kommunikationsmodell und Verhaltensmodell sind dabei für die äußere Logik zuständig. Für die Testautomation gibt es eine eigenständige Komponente, die sich um die Testausführung kümmert.

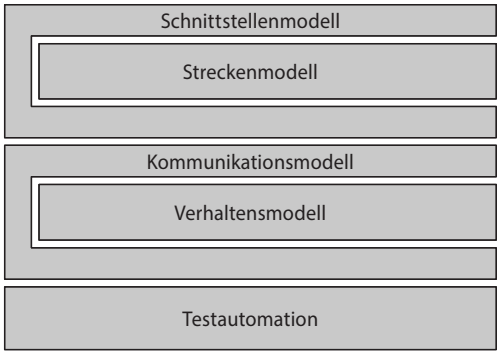


Abb. 4–3
Aufbau Automationskern

Schnittstellenmodell und Streckenmodell

Das Umgebungsmodell setzt sich zusammen aus einem Streckenmodell und einem Schnittstellenmodell. Das Streckenmodell simuliert die Regelstrecke. Das Schnittstellenmodell übersetzt die von der Schnittstelle zum Testobjekt bereitgestellten Ausgänge des Testobjekts (Steuergrößen) in passende Eingabegrößen für das Streckenmodell. Außerdem wandelt

das Schnittstellenmodell die für das Testobjekt relevanten Größen des Streckenmodells (Regelgrößen) passend in Eingaben des Testobjekts um und gibt sie an die Schnittstelle zum Testobjekt weiter. Enthält das Testobjekt keinen Regler, ist das Streckenmodell deutlich einfacher oder entfällt sogar ganz.

Kommunikationsmodell und Verhaltensmodell

Bei Kommunikationsschnittstellen wie Bussen vermittelt das Kommunikationsmodell zwischen der Schnittstelle zum Testobjekt und dem Verhaltensmodell. Das Verhaltensmodell simuliert das Verhalten der anderen Busteilnehmer, die beispielsweise von sich aus regelmäßig Nachrichten versenden, aber auch auf Nachrichten des Testobjekts reagieren. Diese Simulation nennt man Restbussimulation. Wenn keine Simulation eines reaktiven Verhaltens anderer Busteilnehmer nötig ist, kann das Verhaltensmodell deutlich einfacher sein oder ganz entfallen.

Grenzen der Simulation

Verhaltensmodelle und Streckenmodelle sind typischerweise aus Aufwandsgründen in ihrer Leistungsfähigkeit eingeschränkt. Bei komplexem Verhalten des Originals stößt das Verhaltensmodell daher oft an seine Grenzen. Gleiches gilt für das Streckenmodell bei komplexem Verhalten der Regelstrecke. Deshalb muss der Tester bei der Auswahl der auszuführenden Testfälle berücksichtigen, ob er unter Verwendung des Verhaltensmodells und des Streckenmodells ein belastbares Testergebnis bekommen kann oder nicht.

Testautomation

Die Testautomation beeinflusst die äußere Logik des Automationskerns, um Testfälle auf dem Testobjekt auszuführen. Dazu stellt die Testautomation bestimmte Ausgangssituationen her, führt einzelne Testschritte aus, sammelt die Reaktionen des Testobjekts zu diesen Testschritten und bewertet die Reaktionen im Hinblick auf die vorgegebenen Sollreaktionen.

Technische Realisierung Testrahmen

Die technische Realisierung eines Testrahmens umfasst typischerweise die folgenden Bestandteile:

- Hardware wie Steuerrechner und ggf. echtzeitfähige Simulationsrechner,
- Software wie Betriebssystem, Simulationssoftware und Umgebungsmodelle,
- reale Teile aus der Umgebung des Testobjekts (z. B. Sensoren oder Aktuatoren),
- Kommunikationsmittel wie Netzwerkzugänge und Datenlogger,
- Werkzeuge wie Oszilloskope und Messgeräte sowie
- Einrichtungen zum Schutz vor unerwünschten Störungen von außen, z. B. elektromagnetischer Strahlung oder Erschütterungen.

4.1.3 Systemsteuerung

Die Systemsteuerung stellt eine interaktive Schnittstelle für die Funktionen der Testumgebung bereit. Damit implementiert der Tester Testfälle in Form von automatisch ausführbaren Testskripten und lässt sie automatisiert ablaufen. Er kann mehrere Testfälle zu einer Testsuite kombinieren, die als Ganzes auszuführen ist. Alternativ führt der Tester einzelne Testfälle manuell aus. Sowohl bei einer manuellen als auch einer automatisierten Ausführung nutzt die Systemsteuerung den Automationskern zur Testausführung und stellt dem Tester den Ablauf sowie die Ergebnisse der einzelnen Testfälle dar.

Zu den Ergebnissen gehören die während des Tests aufgezeichneten Messungen der benötigten Eingangs- und Ausgangsgrößen (ggf. auch interner Größen) des Testobjekts sowie die aufgezeichneten Nachrichten auf allen angeschlossenen Bussystemen. Diese Aufzeichnungen kann der Tester nutzen, um das Testergebnis selbst zu beurteilen oder von einem Werkzeug beurteilen zu lassen (automatisierte Testauswertung). Die Aufzeichnungen können aber auch dem Entwickler helfen, Fehlerwirkungen auf deren Ursachen hin zu analysieren, um den zugrunde liegenden Fehlerzustand einzugrenzen. Außerdem kann es im Produkthaftungsfall notwendig sein, detaillierte Aufzeichnungen von Testläufen vorzuweisen. Daher speichert die Systemsteuerung die Aufzeichnungen zu jedem Testlauf.

Die Form der Aufzeichnung unterscheidet sich je nach Art der aufgezeichneten Daten. Elektrische Ein- und Ausgänge sowie interne Größen des Testobjekts zeichnet die Systemsteuerung mit einer bestimmten Abtastrate kontinuierlich mit Zeitstempeln auf, Busnachrichten hingegen als einzelne Datagramme mit dem Zeitstempel des Empfangs. Bei manchen Testobjekten liegen auch digitale Audio-, Video- oder Sensorrohdatenströme vor, welche die Systemsteuerung ebenfalls mit Zeitstempeln versehen aufzeichnet. Die Zeitstempel sind dabei meistens relativ zum Beginn der Ausführung des Testfalls angegeben.

Bei einem automatisierten Testfall entscheidet die Systemsteuerung anhand der Aufzeichnungen und der an den Testfällen hinterlegten, erwarteten Ergebnisse, ob ein Testfall bestanden ist oder nicht. Bei manuellen Tests muss der Tester diese Entscheidung selbst treffen und dokumentieren. Aus den Aufzeichnungen und dem Teststatus (bestanden, nicht bestanden, nicht ausgeführt etc.) erzeugt die Systemsteuerung dann automatisch Testberichte zu einem Testlauf.

4.2 Arten von Testumgebungen

Open-Loop-Testsystem

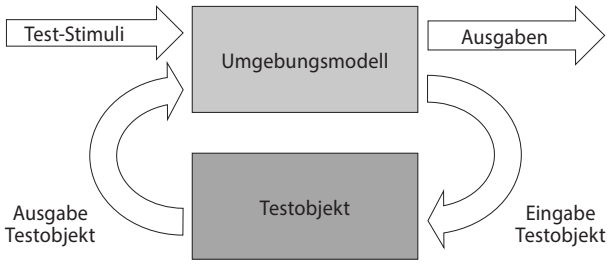
Testumgebungen unterscheiden sich in der Eignung für bestimmte Typen von Testobjekten. Ein Open-Loop-Testsystem ist vor allem geeignet für Systeme mit einer Steuerung, wie sie in Bereichen wie Komfort und Infotainment vorkommen. Das Open-Loop-Testsystem erzeugt Test-Stimuli für das Testobjekt und beobachtet dessen Ausgaben (siehe Abb. 4–4). Ein Beispiel ist der Rückfahrscheinwerfer, der nur leuchten soll, solange der Rückwärtsgang eingelegt ist. Also prüft der Tester den Rückfahrscheinwerfer mit unterschiedlichen Gangwechseln.

Abb. 4–4
Open-Loop-Testsystem

Closed-Loop-Testsystem

Ein Closed-Loop-Testsystem schließt die Rückkopplungsschleife zwischen der Ausgabe des Testobjekts und seinen Eingaben über ein Umgebungsmodell (siehe Abb. 4–5). Das ist vor allem für Systeme mit einer Regelung nötig, beispielsweise für einen Tempomaten. Die Ausgaben des Tempomaten (z. B. das Anfordern einer Beschleunigung) führen zu Änderungen an der zu regelnden Größe (der Geschwindigkeit). Der neue Wert der zu regelnden Größe fließt wieder in die Regelung ein. Für einen möglichst realitätsnahen Test muss die Testumgebung also die Rückkopplungsschleife schließen, d. h. bei einem Regler den Regelkreis. Dazu dient im Testrahmen das Umgebungsmodell, das die Regelstrecke simuliert. Im Unterschied zum Open-Loop-Testsystem gehen die Test-Stimuli nicht mehr direkt an das Testobjekt, sondern an das Umgebungsmodell. Allerdings gibt es auch Closed-Loop-Testrahmen, die ebenfalls Test-Stimuli direkt an das Testobjekt erlauben.

Abb. 4–5
Closed-Loop-Testsystem



Die reale Regelstrecke besteht typischerweise aus drei Teilen:

- die vom Regler beeinflusste Aktuatorik, die auf das Fahrzeug oder dessen Umwelt einwirkt,
- das Fahrzeug und dessen Umwelt sowie
- Sensorik, die das Fahrzeug und dessen Umwelt für den Regler wahrnimmt.

Das Umgebungsmodell muss diese drei Teile geeignet simulieren. Die Realitätsnähe dieser Simulation bestimmt die Brauchbarkeit der Testergebnisse. Daher ist es wichtig, dass die Umgebungsmodelle hinsichtlich ihrer Leistungsfähigkeit validiert sind, bevor der Tester sie in den Testrahmen integriert.

In-the-Loop-Testumgebungen

Je nach Testobjekt und Teststufe gibt es unterschiedliche Arten von Testumgebungen. Typischerweise geht das Testobjekt in den Namen der Testumgebung ein. Tabelle 4–1 gibt einen Überblick über die gebräuchlichsten davon. Dabei steht *in-the-Loop* für ein Closed-Loop-Testsystem.[2]

***Tab. 4–1** Typen von Testumgebungen in der Fahrzeugentwicklung*

Bezeichnung	Testobjekt	Erläuterungen
Model-in-the-Loop (MiL)	ausführbares Entwicklungsmodell, z.B. MATLAB/Simulink-Modell	Testobjekt läuft auf Entwicklungsrechner, Test prüft Funktionalität des Modells, nicht echtzeitfähig
Software-in-the-Loop (SiL)	Software als Maschinencode für Entwicklungsrechner	Testobjekt läuft auf Entwicklungsrechner, Test prüft Funktionalität der Software, nicht echtzeitfähig
Processor-in-the-Loop (PiL)	Software als Maschinencode für Zielprozessor	Testobjekt läuft auf Entwicklungsplatine mit Zielprozessor, Test prüft Funktionalität und Effizienz der Software, eingeschränkt echtzeitfähig
Hardware-in-the-Loop (HiL)	einzelnes Steuergerät, Steuergeräteverbund oder gesamtes E/E-System	Test prüft Funktionalität, Effizienz und Zuverlässigkeit des Testobjekts, echtzeitfähig
Vehicle-in-the-Loop (ViL)	E/E-System im Fahrzeug	Test prüft Funktionalität, Effizienz und Zuverlässigkeit des E/E-Systems im Fahrzeug auf einem Prüfstand, echtzeitfähig
reale Welt (Straße)	Fahrzeug	Test prüft Funktionalität, Effizienz, Zuverlässigkeit und Gebrauchstauglichkeit des Fahrzeugs in einer realen Umgebung, echtzeitfähig

2. In der Praxis kann eine als »in-the-Loop« bezeichnete Testumgebung trotzdem ein Open-Loop-System sein, da es für Open-Loop-Testsysteme keine gebräuchliche Bezeichnung gibt.

In der Tabelle ist die reale Welt die einzige Testumgebung ohne eine Simulation der Umgebung des Testobjekts. Alle anderen Testumgebungen sind *virtuelle* Testumgebungen. Die in der Praxis wichtigsten drei Arten von virtuellen Testumgebungen sind MiL, SiL und HiL. Die folgenden Abschnitte stellen diese drei Testumgebungen im Detail vor.

4.2.1 Model-in-the-Loop-Testumgebung (MiL)

Die MiL-Testumgebung kommt typischerweise in der modellbasierten Softwareentwicklung (MBSE) zum Einsatz. Testobjekt ist ein ausführbares Modell. Die Ausführung erfolgt in der zugehörigen Modellentwicklungsumgebung[3].

Exkurs: Modellbasierte Softwareentwicklung

Hauptmotivation für die MBSE sind Kosteneinsparungen. Da das Modell einen höheren Abstraktionsgrad aufweist als der Code, ist beim Modell typischerweise die Verständlichkeit für alle Projektbeteiligten höher. Eine höhere Verständlichkeit führt zu einer geringeren Fehlerwahrscheinlichkeit und damit zu geringeren Fehlerfolgekosten. Aus diesem Grunde werden modellbasierte Methoden beispielsweise auch von den Normen der funktionalen Sicherheit empfohlen. Typischerweise benötigt man für die MBSE auch weniger Softwareentwickler, wenn man aus den Modellen den Code direkt generieren kann, was Personalkosten spart.

In der MBSE entstehen Modelle zum einen, um die benötigte Funktionalität besser zu verstehen, beispielsweise das angemessene Verhalten eines Reglers im Normalfall und bei Sonderfällen. In dieser Hinsicht dienen Modelle als explorative Prototypen zur Anforderungsklärung. Zum anderen sind Modelle die Basis für die Generierung von Code. Modelle zur Codegenerierung nennt man auch Implementierungsmodelle. Der aus dem Modell generierte Code wird im weiteren Verlauf der Entwicklung Teil der gesamten Software eines Steuergeräts.

Der aus dem Modell generierte Code soll zum einen das Modellverhalten möglichst korrekt wiedergeben. Zum anderen soll der Code bei eingebetteten Plattformen auf der Zielhardware effizient ausführbar sein, um Echtzeitanforderungen zu erfüllen und um Speicherplatz zu sparen. Da diese Ziele zueinander im Widerspruch stehen, sind Kompromisse bei der Codegenerierung nötig. Beispielsweise ersetzt der Codegenerator Signale in Gleitkommadarstellung durch Signale in Ganzzahldarstellung (sog. Skalierung), weil der Prozessor mit der Ganzzahldarstellung schneller rechnen kann und weil die Ganzzahldarstellung weniger Speicherplatz benötigt.

→

3. Besonders verbreitete Modellentwicklungsumgebungen in der Automobilindustrie sind MATLAB/Simulink (mit Stateflow) und ASCET.

Die Skalierung kann dazu führen, dass sich das Verhalten des generierten Codes von dem des Modells unterscheidet, da die Skalierung typischerweise den Wertebereich und die Genauigkeit reduziert. Außerdem könnte der Generator bei der Generierung des Codes Fehler gemacht haben oder der Compiler Fehler bei der Übersetzung des Codes in Maschinensprache. Daher sollte der Tester nicht nur das Modell, sondern auch den generierten und übersetzten Code testen. Verwendet er dabei das Modell als Testorakel, spricht man von einem Back-to-Back-Test (siehe Abschnitt 5.3.4.1).

Aufbau

Der MiL-Testrahmen (siehe Abb. 4–6) läuft typischerweise zusammen mit dem Modell in der Modellentwicklungsumgebung auf einem Entwicklungsrechner. Ein Modelltestwerkzeug generiert die Schnittstelle zum Testobjekt automatisch aus dem Modell, indem es dessen Ein- und Ausgabegrößen sowie dessen interne Größen analysiert. Während der Testausführung können Tester anhand der internen Größen das modellinterne Verhalten beobachten und aufzeichnen. Bei Bedarf ist auch eine Fehlereinfügung (Fault Injection) möglich (siehe Abschnitt 5.3.4.2). Die Tester können während der Testdurchführung die Simulation jederzeit anhalten, um detaillierte Analysen des Modellzustands durchzuführen.

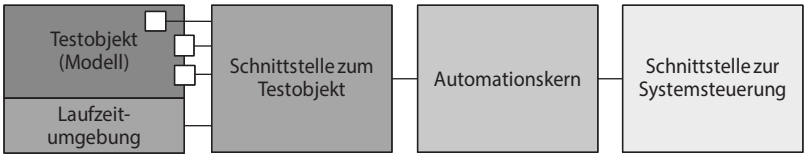


Abb. 4–6
Model-in-the-Loop-Testumgebung

Für die Ausführung des Testobjekts benötigt der MiL eine Laufzeitumgebung, die den notwendigen Kontext für das Modell bereitstellt. Die Laufzeitumgebung führt das Modell aus und simuliert die anderen Systembestandteile, mit denen das Modell interagiert. Das Umgebungsmodell, beispielsweise die Regelstrecke für ein Regler-Modell, ist Teil des Automationskerns (siehe Abschnitt 4.1.2). Je exakter die Umweltsimulation sein soll, desto höher wird der Entwicklungsaufwand für das Umgebungsmodell. Daher konzentriert man sich beim MiL häufig auf den Test der Basisfunktionalität des Modells, arbeitet mit einem vereinfachten Umgebungsmodell und überlässt tiefergehende Aspekte späteren Teststufen.