Durchgängiges, automatisiertes Testen bei der Entwicklung von Automobilelektronik

Beitrag zur Tagung

Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik

Berlin, 30 September und 01. Oktober 2003

Autor

Dr.-Ing. Dipl.-Math. Klaus Lamberg

Produktmanager Test- und Experimentiersoftware

dSPACE GmbH

Technologiepark 25, D-33100 Paderborn

Tel.: 05251/1638-0, Fax: 05251/66529

E-Mail: klamberg@dspace.de

1 Testen als zentrale Maßnahme der Qualitätssicherung

Der zunehmende Stellenwert der Elektronik im Kraftfahrzeug wird durch den wachsenden Anteil der Elektrik/Elektronik an den Herstellkosten deutlich, der mittlerweile bei ca. 30% liegt. Gleichzeitig stellt die Elektrik/Elektronik mit ca. 30 % jedoch auch den Hauptverursacher von Fahrzeugausfällen im Feld dar (Quellen u.a. ADAC). Die durch die wachsende Zahl von Steuergeräten entstehenden Probleme wurden zwar erkannt. Doch auch wenn es mittlerweile Bestrebungen gibt, die Zahl der Steuergeräte bewusst zu reduzieren, so heißt das nicht, dass der Funktionsumfang insgesamt abnimmt. Die funktionale Integration in den Steuergeräten steigt weiter, ebenso der Grad der Vernetzung und somit die Gesamtkomplexität. Auf Prozessebene nimmt die Komplexität ebenfalls zu. Verteilte Prozesse mit mindestens einem Automobilhersteller und in der Regel mehreren Zulieferern sind typisch. Dies führt zu verteilter Verantwortung. Der Nachweis der Qualität wird immer wichtiger, denn durch Qualitätsmängel drohen Haftungsrisiken, Imageprobleme und Kosten für Rückrufaktionen und Nachbesserungen. Dahinter steht die Erkenntnis: „Gute Qualität ist teuer, schlechte noch viel mehr.“ Und die Einführung sicherheitskritischer Funktionen (Stichwort X-by-Wire) verschärft das Problem in Zukunft noch mehr.

Bei der immensen Entwicklungsgeschwindigkeit mit der neue Technologien und Funktionen entstehen, darf die Qualität nicht auf der Strecke bleiben. Qualität wird zum wettbewerbsbestimmenden Faktor, Qualitätssicherung wird zu einer der wichtigsten Aufgaben und zur Kernkompetenz. Und eine wesentliche Maßnahme zur Qualitätssicherung ist das Testen.

Um Testen innerhalb des gesamten Entwicklungsprozesses zu ermöglichen, sind leistungsstarke und effiziente Mittel zur Testentwicklung und -beschreibung notwendig. Gleichzeitig müssen dabei die verschiedenen Anforderungen, die sich aus den Testaufgaben in den unterschiedlichen Entwicklungsphasen ergeben, berücksichtigt werden. Basierend auf einem modellbasierten Testprozess stellt dieser Beitrag einen integrierten Ansatz für die moderne Testentwicklung in den verschiedenen Entwicklungsphasen und das Testmanagement im Gesamtprozess vor.

2 Entwicklung von Automobilelektronik

Die Entwicklung von Automobilelektronik lässt sich in zwei Aufgabenbereiche unterteilen: die Funktions- und Steuergeräteentwicklung einerseits und der Test von Steuergeräten andererseits.

2.1 Modellbasierte Funktions- und Steuergeräteentwicklung

Der Einsatz von Modellierungs-, Simulations- und Code-Generierungswerkzeugen bei der Entwicklung von Fahrzeugfunktionen ist heute üblich. Der Funktionsentwurf erfolgt typischerweise als Beschreibung von Steuerungs- und Regelungsfunktionen und -algorithmen in einer entsprechenden Simulationsumgebung, wie z.B. MATLAB/Simulink/Stateflow. Mithilfe von Rapid-Control-Prototyping-(RCP-) Systemen lassen sich diese Funktionen im realen Fahrzeug oder am Prüfstand erproben. Dazu wird aus dem Funktionsmodell mittels automatischer Codegenerierung C-Code erzeugt, welcher auf leistungsfähigen Echtzeitsystemen zur Anwendung kommt. Über geeignetete I/O wird das RCP-System mit der realen Regelstrecke verbunden. Änderungen lassen sich ohne weiteres am ursprünglichen Funktionsmodell durchführen und mittels erneuter Codegenerierung am realen System erfahren.

Die Implementierung der Funktion auf einem Seriensteuergerät erfolgt ebenfalls zunehmend per automatischer Codegenerierung. Allerdings sind die Anforderungen an Seriencode-Generatoren sehr viel höher als beim RCP. Der erzeugte Code muss hocheffizient, fehlerfrei, reproduzierbar und gut dokumentiert sein [1]. Ein Beispiel für einen Seriencode-Generator ist

TargetLink [2]. Der generierte Funktionscode wird in die gesamte Steuergerätesoftware integriert. Dies umfasst im wesentlichen die Integration mit weiteren Funktionsmodulen, mit dem Betriebssystem und mit der I/O des Steuergeräts (SG).

2.2 Testen von Steuergeräten

Für den Test von Steuergeräten hat sich die Hardware-in-the-Loop-Simulation (HIL-Simulation) etabliert und bewährt. Dabei werden zu entwickelnde und zu testende elektronische Steuergeräte nicht im realen Fahrzeug verbaut, sondern an ein Simulationssystem angeschlossen, das die mechanische, elektrische, hydraulische und elektronische Umgebung des zu entwickelnden Steuergeräts in Echtzeit simuliert [3]. Beispiele für die frühen Anwendungen der HIL-Simulation sind Motor-, Fahrdynamik- und auch Klimaregelungen. Mittlerweile werden jedoch sämtliche Steuergeräte einer Fahrzeugbaureihe mithilfe der HIL-Simulation getestet, weil der Gesamtverbund der Elektroniksysteme im Fahrzeug eine kaum noch zu beherrschende Gesamtkomplexität erreicht [4], [5].

Für ein einzelnes Steuergerät oder eine Fahrzeugfunktion stellen hunderte oder gar tausende von implementierten Tests eine mittlerweile übliche Größenordnung dar, die mittels HIL-Simulation automatisiert durchgeführt werden [6]. Diese Tests laufen i.d.R. zwischen einigen Stunden und einigen Tagen automatisch ab, häufig nachts.

Mithilfe der HIL-Simulation lassen sich bis zu 90% aller im Fahrversuch auftretenden Fehler nachbilden. Daher stellt sich das Kosten-Nutzen-Verhältnis bei der HIL-Simulation trotz einer auf den ersten Blick nicht gerade geringen Investition insgesamt sehr positiv dar, findet doch eine Amortisierung erfahrungsgemäß schon innerhalb von drei bis sechs Monaten statt.

2.3 Heutige Probleme beim Testen von Automobilelektronik

Testen umfasst heute 30% bis 40% des Gesamtaufwands eines Softwareprojekts [7]. Dennoch besteht Testen heute vielfach aus „Ausprobieren“. Das heutige Vorgehen, insb. die Erprobung von Funktionen direkt am realen System, erfolgt überwiegend experimentell. Stellt die Testautomatisierung im Zusammenhang mit der HIL-Simulation bereits heute den Stand der Technik dar, so spielt das systematische und automatisierte Testen in den frühen frühen Phasen der Funktionsentwicklung (s. Abschnitt 2.1) bisher nur eine untergeordnete Rolle. Darüber hinaus fehlt es an Testwerkzeugen, welche bestimmte, auf die jeweiligen Prozessphasen zugeschnittene Vorgehensmethodiken bereitstellen, und gleichzeitig den Gesamtprozess unterstützen. Mit dem im Folgenden vorgestellten modellbasierten Testprozess und der Unterstützung der unterschiedlichen, resultierenden Testaktivitäten wird bewusst das modellbasierte und automatisierte Testen im gesamten Entwicklungsprozess adressiert.

3 Der modellbasierte Testprozess

Der hier vorgestellte modellbasierte Testprozess (Abbildung 1) stellt die verschiedenen Aufgaben innerhalb der Funktions- und Steuergeräteentwicklung aus Sicht des Testens dar. Gleichzeitig legt einen wesentlichen Fokus auf das automatisierte Testen insb. in frühen Entwicklungsphasen. Gerade im Zusammenhang mit der Funktions- und Softwareentwicklung wird das Experimentieren durch den dargestellten Prozess um das systematische und automatisierte Testen ergänzt. Darüber hinaus enthält der Prozess die bereits heute üblichen Aktivitäten in Bezug auf den Test realer Steuergeräte auf verschiedenen Ebenen.

Der modellbasierte Testprozess besteht im wesentlichen aus den folgenden Testschritten:

- Der „Test des Funktionsmodells“ umfasst das systematische und automatisierte Testen eines ausführbaren Modells der zu entwickelnden Funktion bzw. des Reglers gemäß Abschnitt 2.1. Dieses Modell stellt das Testobjekt oder die „Unit under Test“ (UUT) dar. Der Test kann Open-Loop oder gegen ein Modell der entsprechenden Regelstrecke erfolgen

(Model-in-the-Loop, MIL). Solche Tests können sowohl beim Automobilhersteller als auch beim Zulieferer sinnvoll sein.

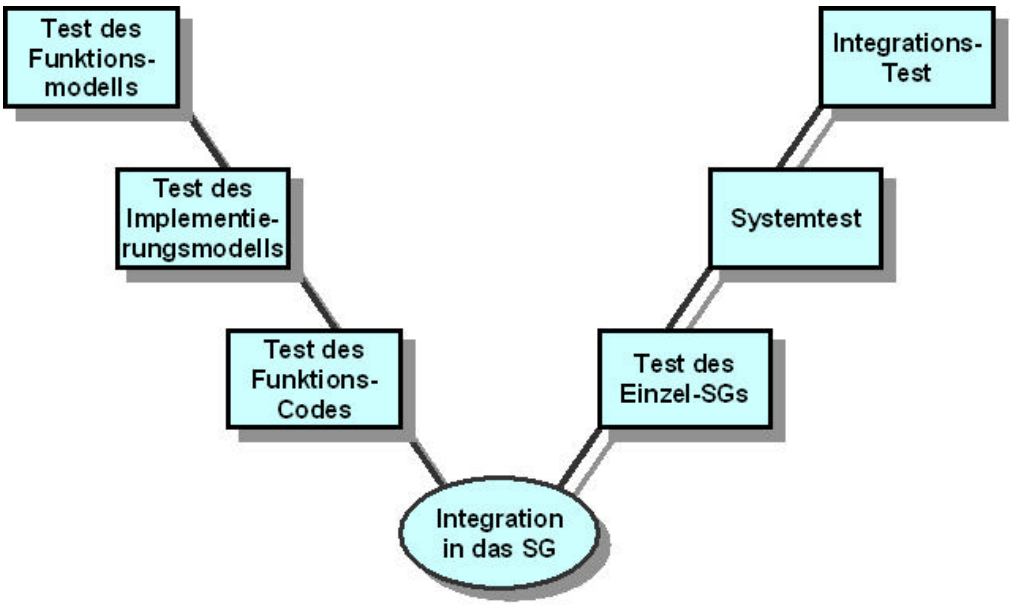


Abbildung 1: Der modellbasierte Testprozess

- Beim „Test des Implementierungsmodells“ handelt es sich bei der UUT um das Funktions- bzw. Reglermodell, jedoch in der für die anschließende Seriencodegenerierung vorliegenden Form. Dazu werden zusätzlich „Design-Informationen“ in das Funktionsmodell integriert. Während das Funktionsmodell i.d.R. als Floating-Point-Modell vorliegt, erfolgt die Implementierung in C häufig mit Fixed-Point-Arithmetik (inkl. Skalierung, Datentyp, LSB und Offset für jedes Signal und jeden Parameter), abhängig vom Zielprozessor im Steuergerät. Diese Fixed-Point-Darstellung wird durch das Implementierungsmodell repräsentiert. Das Verhalten des Implementierungsmodells muss insb. gegen das Verhalten des ursprünglichen Funktionsmodells getestet werden. Abweichungen sind i.d.R. eine Folge entsprechender Quantisierungseffekte. Auch das Implementierungsmodell kann in einer geschlossenen Regelschleife getestet werden (MIL). Da die Codierung von Steuergeräten i.d.R. beim Zulieferer erfolgt, wird diese Art des Tests auch meistens dort durchgeführt.
- Die darauf folgende Repräsentationsform, in der die Funktion getestet werden muss, ist der Funktions-Code selber („Test des Funktions-Codes“). Dieser kann entweder auf dem Host-PC ausgeführt werden (Software-in-the-Loop, SIL) oder auf dem Zielprozessor (Processor-in-the-Loop, PIL). Auch diese Art des Testens ist vornehmlich Aufgabe des Zulieferers.
- Der „Test des Einzel-SGs“ erfolgt i.d.R. mittels HIL-Simulation. Es handelt sich weitgehend um Black-Box-Tests, wobei fehlende Busteilnehmer simuliert werden (Restbussimulation). Ziel ist der Test der einzelnen Steuergerätefunktionen. Es ist auch möglich, dass die UUT kein reales Steuergerät, sondern ein RCP-System ist. Dies kann z.B. sinnvoll sein, bevor sicherheitskritische Funktionen in einem RCP-System im realen Fahrzeug erprobt werden. Der Test des einzelnen Steuergeräts erfolgt überwiegend beim Zulieferer, vereinzelt auch beim OEM.
- Ziel des Systemtests ist es, das Steuergerät in seiner unmittelbaren Umgebung mithilfe der HIL-Simulation zu testen. Dazu wird es partiell mit anderen Steuergeräten integriert, ansonsten kann ebenfalls eine Restbussimulation erfolgen. Der Systemtest wird i.d.R. beim OEM durchgeführt.

- Abschließend werden sämtliche Steuergeräte eines Bussystems oder des gesamten Fahrzeugs zum Gesamtsystem integriert und der Gesamtverbund getestet. Dies wird als „Integrationstest“ bezeichnet. Auch hier kommt die HIL-Simulation zunehmend zur Anwendung.

Die konkreten Testaufgaben in den verschiedenen Phasen des dargestellten Prozesses sind sehr unterschiedlich. Dies hat verschiedene Gründe:

Zum einen handelt es sich um einen auf einen Automobilhersteller und mehrere Zulieferer verteilten Prozess mit unterschiedlichen Verantwortlichkeiten. Während beispielsweise der Automobilhersteller die generelle Übereinstimmung eines Funktionsmodells mit den Anforderungen sicherstellen muss, interessiert den Zulieferer die Übereinstimmung der Testergebnisse von Implementierungsmodell und Funktionscode. Und während der Zulieferer testet, ob die Endstufen eines Steuergeräteprototypen funktionieren, überprüft der Automobilhersteller, ob sämtliche Diagnosefunktionen bereits vollständig und korrekt implementiert sind.

Ein weiterer Unterschied zwischen den verschiedenen Testphasen ergibt sich durch die verschiedenen Schnittstellen zum Testobjekt. Während beim Test des Funktionsmodells jedes einzelne Modul identifizier-, separier- und instrumentierbar ist, sind innerhalb des Steuergeräts nicht mehr alle Funktionsschnittstellen zugreifbar. Eine entsprechende Instrumentierung des Steuergerätecodes kann zu einem veränderten Laufzeit- und Ressourcenverhalten führen. Daher lassen sich Tests im linken Ast des Testprozesses häufig nicht 1:1 im rechten Ast wiederverwenden. Es muss dazu erst die Schnittstelle, auf welcher der Test aufsetzt, neu identifiziert und zugewiesen werden.

Diese Unterschiede bzgl. der Rollenverteilung, der Testaktivitäten und -ziele führen zu unterschiedlichen Anforderungen an Testwerkzeuge für die einzelnen Testphasen. Die Aufgaben im Zusammenhang mit der HIL-Simulation sind sehr vielfältig und erfordern daher flexible und vielseitige Möglichkeiten der Testbeschreibung. Um die Komplexität im Spannungsfeld Steuergeräteentwicklung – HIL-Simulation zu beherrschen, existiert häufig eine klar getrennte Aufgabenverteilung zwischen Testentwickler, Testanwender und Testsystemexperten [8]. Beim Testen in frühen Entwicklungsphasen ist diese Rollenverteilung nicht so stark ausgeprägt. Dort testet der Funktionsentwickler überwiegend selber. Er braucht daher auf seine Aufgabenstellung zugeschnittene und weniger flexible Testwerkzeuge. Im Zusammenhang mit dem Test von Funktionsmodell, Implementierungsmodell und Funktionscode ist es daher wichtig, dass ein Testwerkzeug dem Entwickler ein geeignetes Vorgehensmodell bereitstellt, das ihn durch seine speziellen Testaufgaben führt.

4 Lösungen für das durchgängige, modellbasierte Testen

Im folgenden werden Lösungen für die speziellen Testaufgaben in den verschiedenen Phasen des modellbasierten Testprozesses dargestellt. Dies umfasst sowohl das systematische Testen innerhalb der Funktions- und Softwareentwicklung (Abschnitt 4.1), wie auch flexible und effiziente Möglichkeiten zur Testbeschreibung für den Test realer Steuergeräte (Abschnitte 4.2 und 4.3). Schließlich wird gezeigt, wie diese Lösungen in einem einzigen Testwerkzeug integriert werden (Abschnitt 0).

4.1 Systematisches Black-Box-Testen mit MTest

Ein Werkzeug, welches das systematische, modellbasierte Testen speziell von SW- und Funktionsmodulen in Verbindung mit Simulink und TargetLink unterstützt, ist MTest. MTest ergänzt die modellbasierte SW- und Funktionsentwicklung um die Klassifikationsbaummethode. Ausgangspunkt der MTest-Methodik ist ein Modell der zu testenden Funktion bzw. des zu testenden Reglers in Simulink oder TargetLink. Ausgehend von der Schnittstelle des Modells und unter Verwendung der Klassifikationsbaummethode [9] kann der Anwender systematisch Testszenarien definieren und grafisch unterstützt beschreiben. Die Klassifikations-

baummethode wurde bereits Anfang der neunziger Jahre entwickelt [10] und fand bisher vornehmlich Anwendung im Zusammenhang mit dem Test reinen C-Codes, weitgehend unabhängig von der modellbasierten Entwicklung.

4.1.1 Die Klassifikationsbaummethode

Die Klassifikationsbaummethode ist eine Black-Box-Methode, bei welcher der Eingaberaum des Testobjekts (Menge der Inputwerte und -kombinationen) unter bestimmten Aspekten zerlegt wird. Die entstehenden Partitionen, die sogenannten Klassifizierungen, werden wiederum in Äquivalenzklassen unterteilt. Ziel der Partitionierung ist es, die einzelnen Klassen so zu wählen, dass sie sich einheitlich (uniform) in Bezug auf die Aufdeckung potenzieller Fehler verhalten. Das heißt, die UUT verhält sich für alle Werte einer Klasse entweder korrekt oder fehlerhaft („Uniformitätshypothese“). Schließlich werden Kombinationen von Klassen ausgewählt und darauf basierend werden Testsequenzen definiert.

Der Aufbau eines Klassifikationsbaums soll anhand eines Beispiels aus [11] gezeigt werden. Abbildung 2 zeigt die Schnittstelle eines Fahrdynamikreglers, Abbildung 3 zeigt einen zugehörigen Klassifikationsbaum.

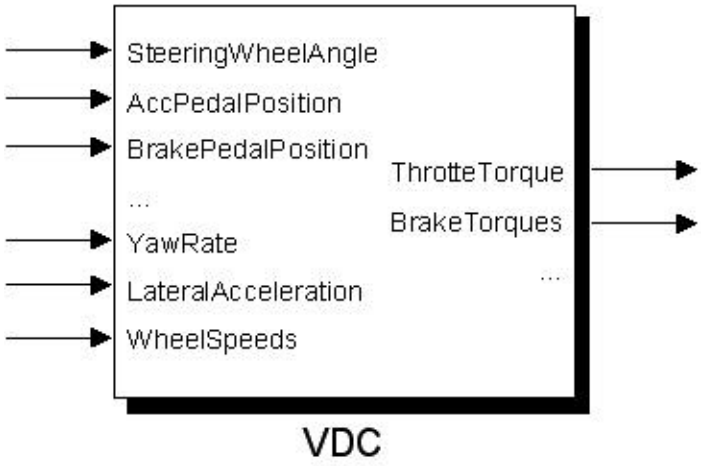


Abbildung 2: Schnittstelle des Fahrdynamikreglers

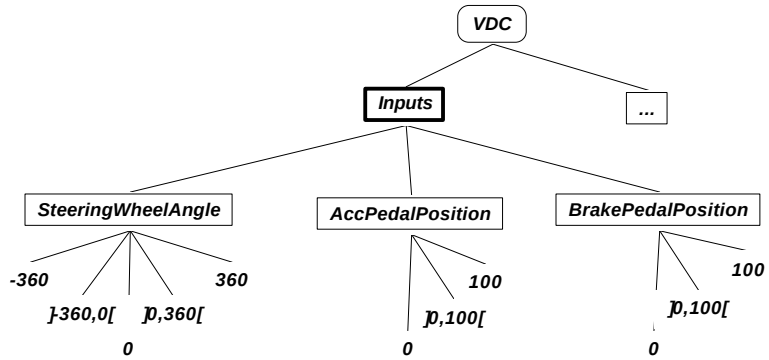


Abbildung 3: Klassifikationsbaum für den Regler

Der Name der UUT („VDC“) bildet die Wurzel des Baums, die Inputsignale definieren die Klassifikationen unterhalb des Wurzelknotens. Darunter finden sich die Äquivalenzklassen. Der Benutzer kann weitere Klassifikationen und Klassen definieren, bestehende löschen oder modifizieren usw..

Basierend auf der so erfolgten Partitionierung des Eingaberaums lassen sich nun Testsequenzen definieren. Die Testsequenzen ergeben sich durch gezielte Kombination einzelner Klassen: jede Zeile der Kombinationstabelle beschreibt einen Testschritt innerhalb der Testsequenz. Um einen solchen Testschritt zu definieren, werden dort einige der senkrecht darüber stehenden Klassen des Klassifikationsbaums markiert.

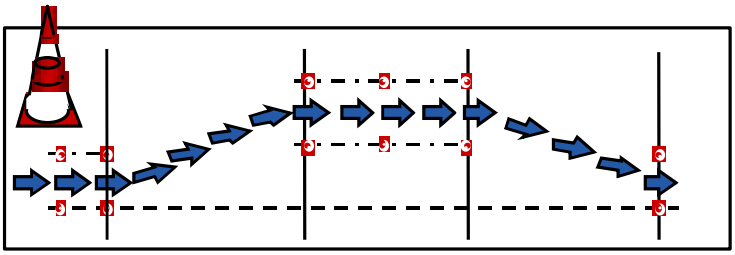


Abbildung 4: Fahrmanöver Spurwechseltest

Eine Darstellung des in Abbildung 4 abgebildeten Spurwechseltests als Testsequenz in der Kombinationstabelle zeigt Abbildung 5 unten. Nach einer Beschleunigungsphase erfolgt zunächst ein Lenkradeinschlag um 90° nach links (auf -90°), anschließend eine Lenkbewegung um 180° nach rechts (auf +90°) und wieder zurück in die Ausgangslage. Nach einer Haltephase wird um 90° nach rechts und wieder um 180° in die entgegengesetzte Richtung gelenkt (auf -90°) und anschließend wieder zurück in die Nulllage.

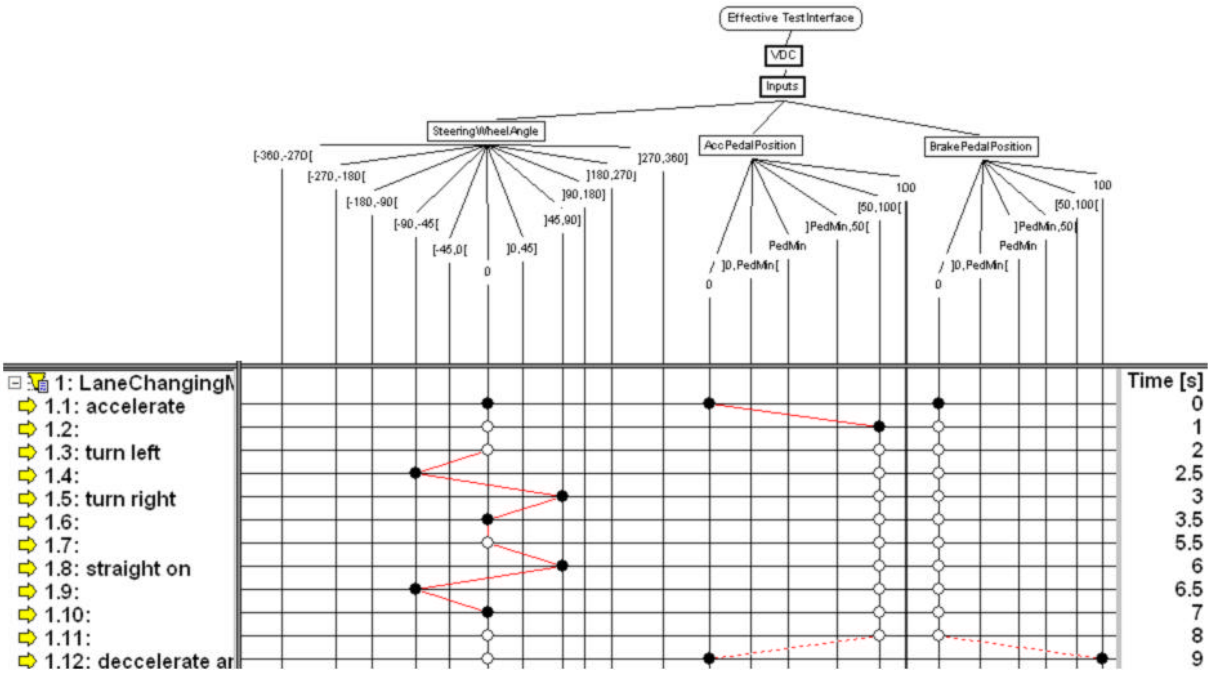


Abbildung 5: Klassifikationsbaum mit Testsequenzen

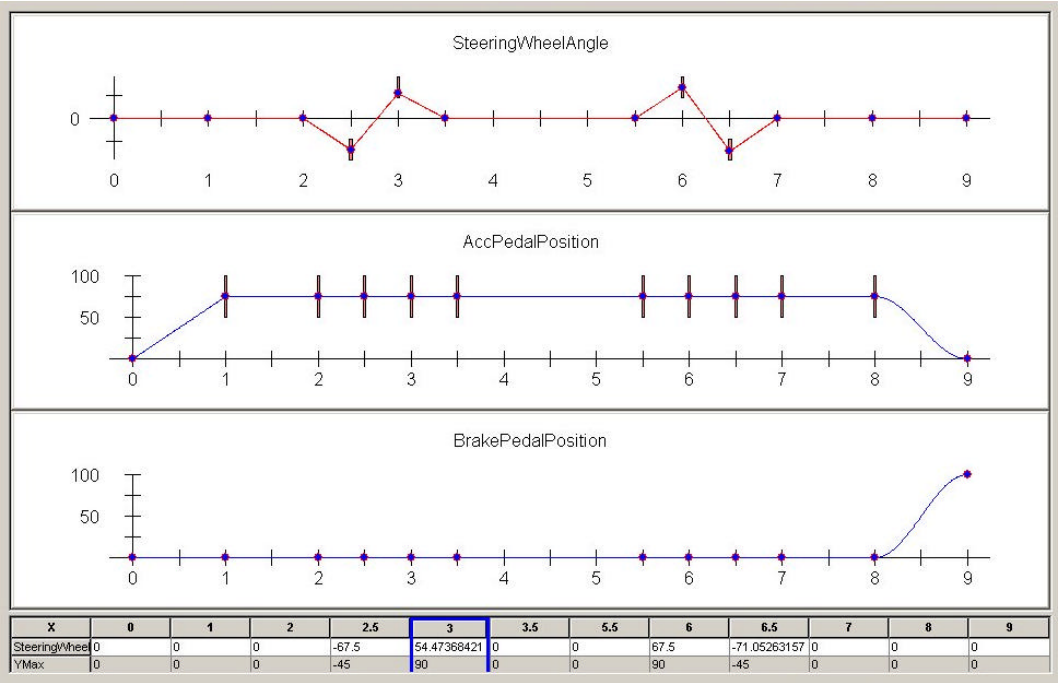
Eine durchgehende Linie als Transitionstyp bedeutet dabei eine rampenförmige Änderung des Signalwertes, keine (sichtbare) Transition bedeutet einen Sprung des Signalwertes am Intervallende. Das Gaspedal wird zu Beginn der Beschleunigungsphase leicht betätigt. Die eingestellte Pedalposition wird gehalten und am Ende der Testsequenz wird das Pedal wieder gelöst. Gleichzeitig erfolgt das Betätigen der Bremse. Eine gestrichelte Linie als Transitionstyp bedeutet einen Wechsel des Signalwertes in Form einer Sinus-Halbwelle. Auf die

beschriebene Art und Weise können unterhalb des Klassifikationsbaumes weitere Testsequenzen beschrieben werden.

4.1.2 Definition von Testdaten

Die mithilfe der Klassifikationsbaummethode entwickelten Testszenarien enthalten lediglich eine abstrahierte Form von Stimulusinformationen für die UUT, da nur Äquivalenzklassen und keine konkreten Werte zugrunde gelegt wurden. Daher werden die zu verwendenden Testdaten in einem zweiten Schritt konkretisiert. Die im Klassifikationsbaum definierten Vorgaben können dabei als Signalkorridore interpretiert werden. Innerhalb dieser Korridore dürfen die tatsächlichen Stimulusdaten variieren.

Die Konkretisierung der Testdaten erfolgt mit dem in Abbildung 6 dargestellten Signaleditor. Die Grenzen der Äquivalenzklassen bilden die Begrenzungen der Wertebereiche an den entsprechenden Stützstellen. MTest verwendet standardmäßig die Mittelwerte der durch den Klassifikationsbaum vorgegebenen Korridore. An der Stelle 3 Sekunden wurde der Standardwert im Rahmen der Grenzen manuell editiert (vergl. markierte Zellen im unteren Bereich der Darstellung).



X	0	1	2	2.5	3	3.5	5.5	6	6.5	7	8	9
SteeringWheel	0	0	0	-67.5	54.47368421	0	0	67.5	-71.05263157	0	0	0
YMax	0	0	0	-45	90	0	0	90	-45	0	0	0

Abbildung 6: Signaleditor mit Signalkorridoren

Alternativ zur Verwendung der Klassifikationsbaummethode bietet MTest die Möglichkeit, bereits vorhandene Daten als Testdaten zu verwenden. Dieses wird als „direktes Testen“ bezeichnet. Beim direkten Testen ist es möglich, z.B. Messdaten aus dem realen Fahr- oder Prüfstandsversuch zu importieren und als Stimulusdaten zu verwenden. Auf diese Weise kann die UUT mit den realen Messdaten getestet werden.

4.1.3 Anwendung vom Modell bis zum C-Code

Die mittels Klassifikationsbaummethode oder Datenimport beim direkten Testen entstandenen Testszenarien können auf sämtliche Repräsentationsformen der UUT angewendet werden: auf das Funktionsmodell, auf das Implementierungsmodell und auf den Funktionscode. Dazu unterstützt MTest die Ausführung der Tests in Simulink und in den verschiedenen TargetLink-Modi. Für die Ausführung der Tests wird in Simulink automatisch ein Testrahmen

generiert (s. Abbildung 7), in welchen das Simulink- oder TargetLink-Modell der UUT hineinkopiert wird.

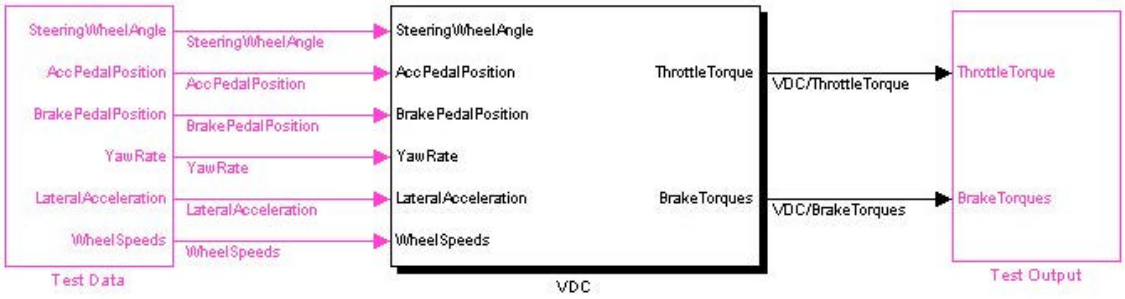


Abbildung 7: Automatisch generierter Testrahmen in Simulink

MTest kann den entsprechenden TargetLink Simulationsmodus – Floating-Point-Simulation auf dem Host-PC („TargetLink MIL“), Seriencode-Simulation auf dem Host-PC („TargetLink SIL“) und Seriencode-Simulation auf dem Zielprozessor („TargetLink PIL“) – aktivieren und ggfs. auch die C-Code-Generierung mit TargetLink anstoßen. Abbildung 8 zeigt im linken Teil, wie die verschiedenen Modi, in denen eine Testsequenz ausgeführt wird, ausgewählt werden können. Der rechte Teil der Abbildung zeigt, wie die einzelnen Modi als Unterknoten der Testsequenz „LaneChangingManeuver“ im MTest-Projektbaum dargestellt werden.

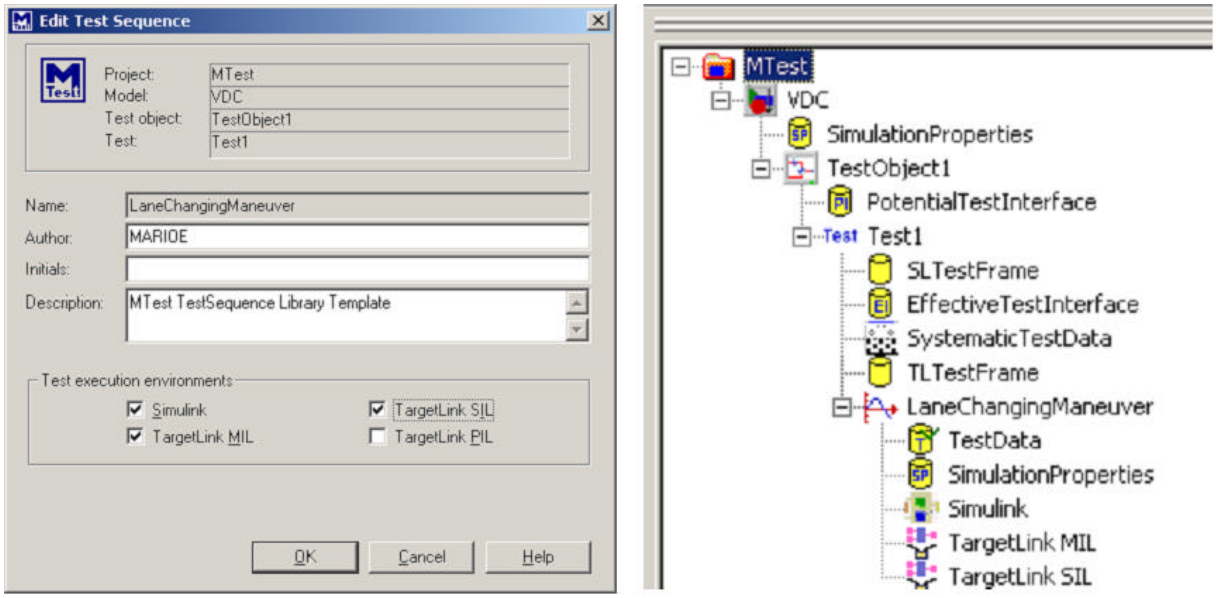


Abbildung 8: Testsequenz „LaneChangingManeuver“ in den verschiedenen Simulationsmodi

Darüber hinaus ist es möglich, Referenzdaten vorzugeben, mit denen das Ausgangsverhalten der entsprechenden UUT-Repräsentation verglichen wird. Schließlich können das Ausgangsverhalten der einen Repräsentationsform als Referenzvorgabe für die nächste verwendet und Abweichungen automatisch erkannt und dokumentiert werden.

4.2 Grafische Testbeschreibung für den Steuergerätetest

Im rechten Ast des Testprozesses kommt überwiegend die HIL-Simulation als Plattform für das automatisierte Testen zum Einsatz. Der Test von Steuergeräten mit HIL-Simulation erfordert die Unterstützung unterschiedlichster Anwendungsfälle, vom Diagnosetest über OBD-II-Tests für Motorsteuergeräte bis hin zu Fahrmanövern für den Test von Fahrdynamiksteuergeräten. Darüber hinaus enthalten HIL-Systeme verschiedene, automatisierbare Schnittstellen, z.B. für den Modellzugriff, die Ansteuerung der elektrischen Fehlersimulation und die Anbindung von Kalibrier- und Diagnosetools [8]. Aus diesen Gründen verlangt der Test mithilfe der HIL-Simulation eine möglichst große Flexibilität und eine übersichtliche und effiziente Unterstützung bei der Testentwicklung.

Testprogramme wurden bisher in der Regel mithilfe von Skriptsprachen geschrieben. Die Verwendung von Skiptsprachen ist äußerst flexibel. Fortgeschrittene Nutzer können ihre eigenen Bibliotheken anlegen und somit Funktionen und Tests wiederverwenden. Andererseits erfordert die manuelle Programmierung zunächst das Erlernen der Skriptsprache. Für weniger geübte Anwender kann die Testerstellung in einer Skriptsprache umständlich und fehleranfällig sein. Daher ist es von Vorteil, wenn Tests auf einer höheren Abstraktionsebene beschrieben werden können. Dies kann durch die grafische Testentwicklung erreicht werden. Ein grafischer Testeditor ermöglicht eine kurze und steile Lernkurve durch den Anwender. Die Testentwicklung wird schneller und somit effizienter, mehr Tests können in kürzerer Zeit entwickelt werden. Zusätzlich vereinfacht eine grafische Testrepräsentation das Navigieren in komplexen Testsequenzen mit mehreren Hierarchieebenen.

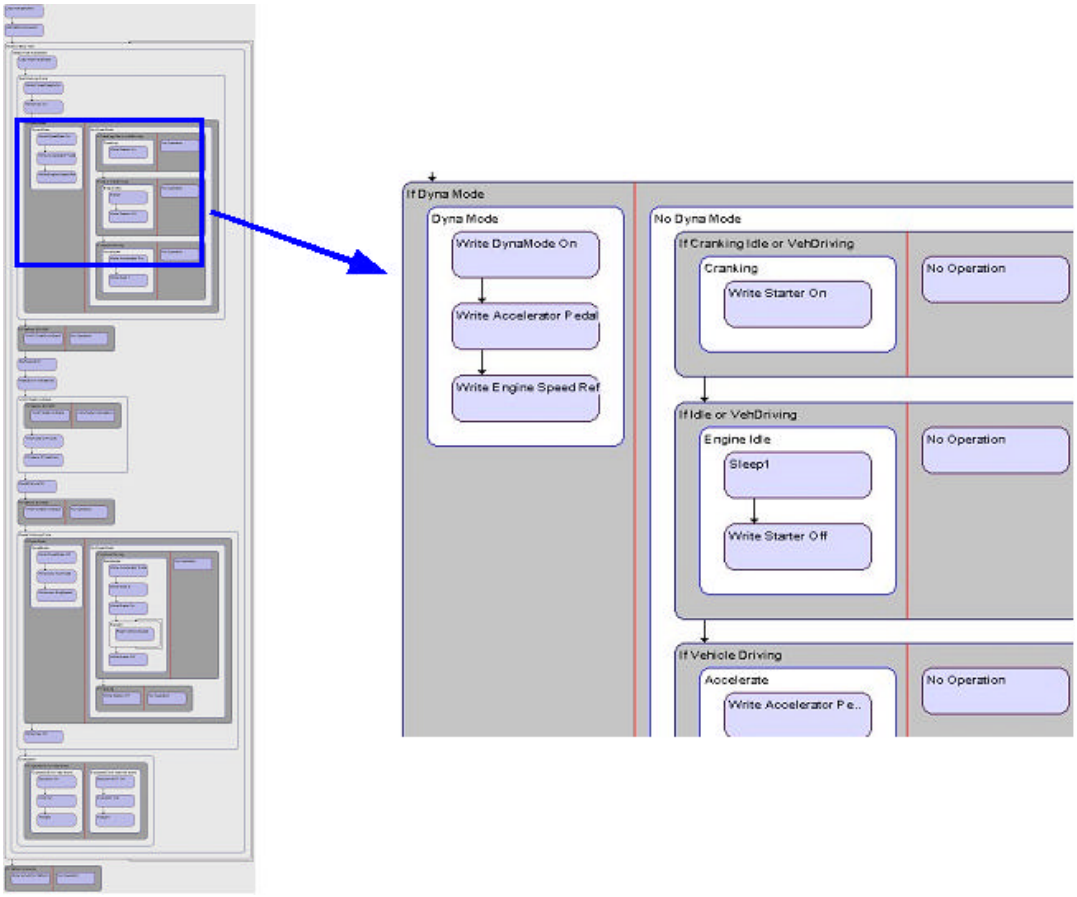


Abbildung 9: Grafische Darstellung eines Testablaufs

Abbildung 9 zeigt links eine Darstellung eines Diagnosetests, wie er an einem HIL-Simulator verwendet wird. Das blaue Kästchen markiert den Ausschnitt, der im rechten Teil der Abbildung dargestellt wird. Die verwendete Semantik lehnt sich an UML-Aktivitätsdiagramme an. Die Bedeutung der UML (Unified Modeling Language, [12]) in diesem Zusammenhang und insb. im automotiven Umfeld steigt immer mehr, sowohl für die Beschreibung von Software-Architekturen, als auch für die Testspezifikation. Um Testbeschreibungen zukünftig zwischen verschiedenen Testwerkzeugen austauschen zu können, ist eine Standardisierung anzustreben. Die UML bietet hierfür eine mögliche Grundlage.

4.3 Wiederverwendbarkeit durch Bibliotheken

Die Effektivität eines Automatisierungskonzepts wird grundlegend durch die verfügbaren Automatisierungsfunktionen bestimmt. Diese Automatisierungsfunktionen definieren den Funktionsumfang, den ein Anwender aus einer Testsequenz heraus nutzen kann. Typische Beispiele dafür sind Funktionen für den Zugriff auf das Simulationsmodell, Funktionen für die Verwaltung und Bedienung von Fehlersimulations-Hardware (z.B. Relais-Karten in einem HIL-Simulator) und Funktionen für den Zugriff auf den Diagnosespeicher des zu testenden Steuergeräts. Weitere Funktionsumfänge, die beim automatisierten Testen häufig benötigt werden, sind Schnittstellen zu anderen Geräten und Programmen, z.B. zur Auswertung von Tests und Schnittstellen zur automatischen Report-Generierung. Innerhalb eines Tests werden diese Funktionen aufgerufen und verwendet. Sinnvoll ist es, wenn die Funktionen für die grafische Testentwicklung in einer Bibliothek bereitstehen. Der Anwender kann sie dann sehr einfach in seine Testsequenz einbauen, indem er sie unter Verwendung der Maus einfach aus der Bibliothek in die grafische Testdarstellung zieht.

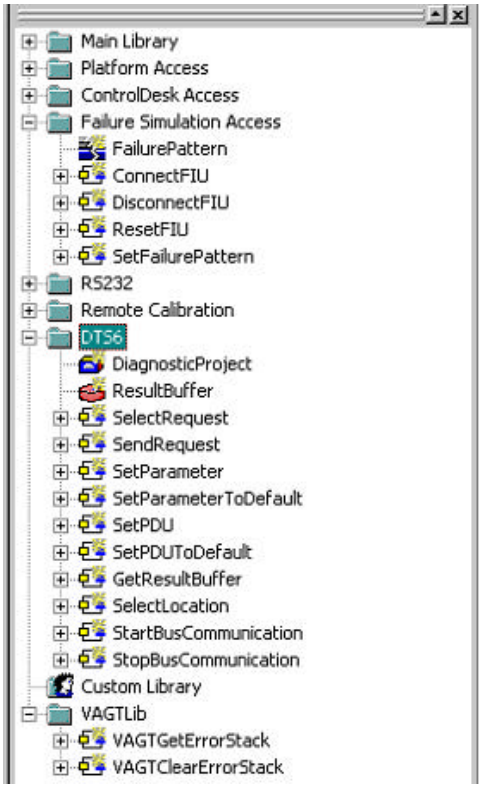


Abbildung 10: Bibliotheken in AutomationDesk

Generell lassen sich zwei Arten von Bibliotheken unterscheiden:

- “Built-in”-Bibliotheken stellen grundlegende Funktionalität bereit, um daraus neue Sequenzen und Tests zusammenzustellen. Beispiele dafür sind Kontrollstrukturen (sequen-

tielle oder nebenläufige Strukturen), Bedingungen und Schleifen wie z.B. If-Else- oder While-Konstrukte, algebraische und logische Operatoren und Schreib- und Lesefunktionen für den Zugriff auf das zugrundeliegende Simulationsmodell.

- Kundenspezifische Bibliotheken enthalten Erweiterungen, die durch den Anwender gemacht werden. Typischerweise handelt es sich um generische Sub-Sequenzen, wie z.B. das Anfahren eines bestimmten Betriebspunkts, die in verschiedenen Testsequenzen benötigt und wiederverwendet werden sollen.

Abbildung 10 zeigt verschiedene Bibliotheken, die typischerweise in HIL-Anwendungen verwendet werden. Ein typisches Anwendungsbeispiel ist der Test von Diagnosefunktionen. Dabei wird für jeden Pin des Steuergeräts und für jeden möglichen elektrischen Fehler der gleiche Testablauf durchfahren. Das Ziel ist es zu testen, ob das Steuergerät sämtliche elektrischen Fehler richtig erkennt. Die Prüfzeiten lassen sich durch die Automatisierung derartiger Testabläufe auf ein Zehntel reduzieren [13]. In Abbildung 10 werden entsprechend die „Failure Simulation Access“-Bibliothek und zwei Bibliotheken für die Steuergätediagnose dargestellt. Die „Failure Simulation Access“-Bibliothek dient zur Ansteuerung von Relaiskarten und zur elektrischen Fehlersimulation aus entsprechenden Testsequenzen heraus. Die DTS6-Bibliothek stellt Funktionsblöcke bereit, mit denen das Diagnose-Tool „Diagnostic Tool Set“ der Firma Softing aus einem HIL-Simulator heraus angesteuert werden kann. Alternativ kommen auch andere Diagnose-Werkzeuge zum Einsatz, z.B. der sogenannte VAG-Tester. Über die VAGTLib lässt auch dieser sich aus einem entsprechenden Automatisierungsszenario heraus fernbedienen.

Solch ein Bibliothekskonzept hat einen entscheidenden Vorteil: ein Testentwickler kann nicht nur vordefinierte Automatisierungsschritte zu Sequenzen höherer Funktionalität zusammenfügen. Er kann diese Aggregationen auch zurück in die Bibliothek stellen, um sie in anderen Testsequenzen oder Projekten wiederzuverwenden. Nach und nach wachsen die verfügbaren Bibliotheksumfänge, die Testentwicklung wird schneller und somit effizienter.

4.4 Testprojektmanagement

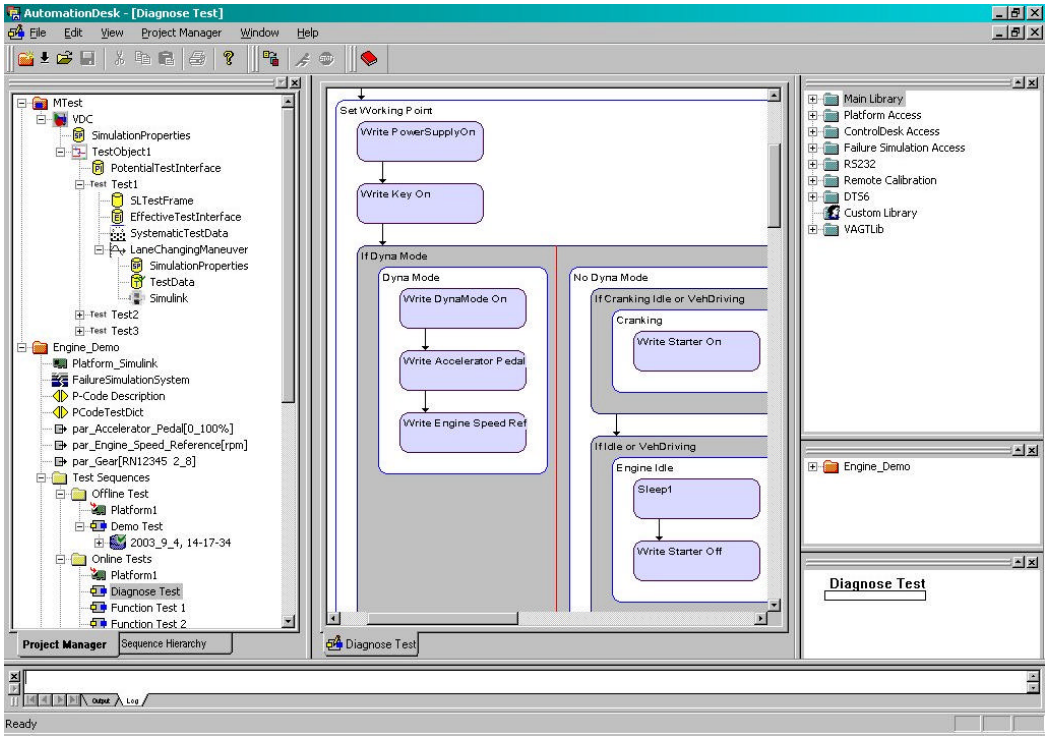


Abbildung 11: Integrierte Testumgebung AutomationDesk

So unterschiedlich und vielfältig die Testaufgaben und -aktivitäten in den verschiedenen Phasen des modellbasierten Testprozesses auch sind – eine Testumgebung für den gesamten modellbasierten Testprozess muss alle Ansätze miteinander unter einem gemeinsamen Dach vereinen. Dies wird erreicht, wenn sämtliche notwendigen Elemente des Prozesses innerhalb einer Testumgebung verfügaus sind. Ein Beispiel für eine solche Integrationsumgebung, das Testwerkzeug AutomationDesk [14], zeigt Abbildung 11.

Die immense Vielzahl von Tests muss nicht nur entwickelt und ausgeführt werden. Die Tests müssen auch gespeichert und reproduzierbar verwaltet werden. Die große Zahl von Testergebnissen – jeder Testlauf erzeugt neue Ergebnisse – muss ebenfalls administriert werden. Basierend auf den Ergebnissen werden automatisiert Testreports erzeugt. Die Speicherung, Verwaltung und Administration der großen Zahl von Tests, Testdaten und Testergebnissen stellen zentrale Aufgaben dar, die allen Testphasen gemeinsam ist. Sie erfordern Möglichkeiten der Strukturierung und des Managements von Testprojekten.

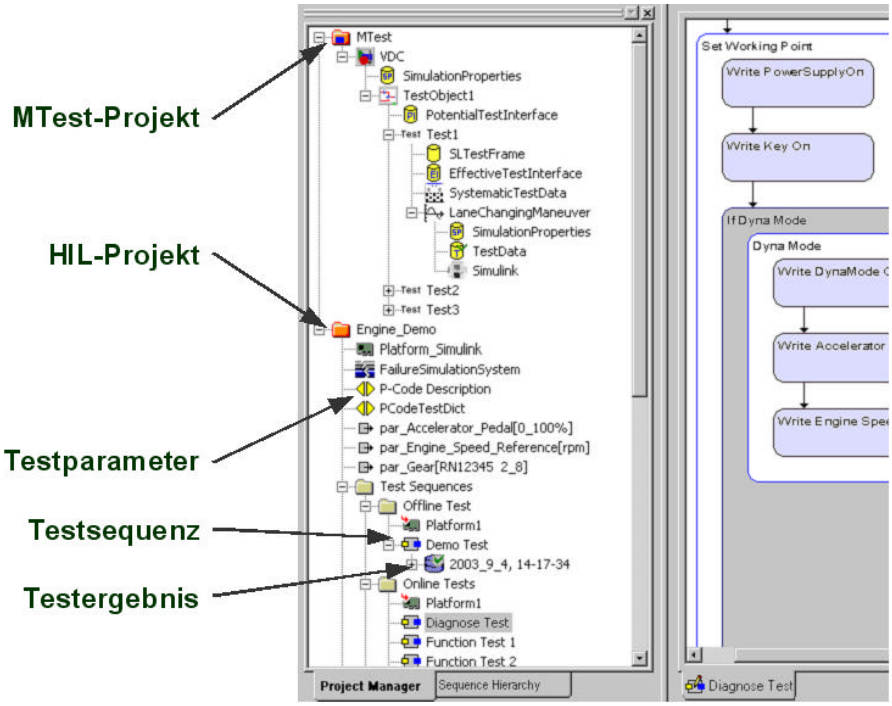


Abbildung 12: Beispiel einer Testprojekt-Struktur

Abbildung 12 zeigt ein Beispiel für eine Testprojekt-Struktur in AutomationDesk. Die Testdaten und Testergebnisse werden gemeinsam mit den Testsequenzen in Projektbäumen dargestellt. Im oberen Teil ist ein MTest-Projekt dargestellt, im unteren ein typisches HIL-Projekt. Es wird deutlich, dass durch die Verwaltung innerhalb eines einzigen Werkzeugs Testdaten, die z.B. mit MTest für den Test von Funktionsmodellen erstellt wurden, in der HIL-Simulation wieder verwendet werden können, auch wenn die Schnittstelle zum Testobjekt sich verändert hat.

5 Zusammenfassung

Das Testen von Software und Steuergeräten spielt eine zentrale Rolle bei der Entwicklung von Automobilelektronik. In den späten Entwicklungsphasen, insb. für den Test von Steuergeräten mithilfe der HIL-Simulation hat sich das automatisierte Testen etabliert. In den früheren Entwicklungsphasen besteht jedoch ein großes Potenzial, das überwiegend experimentelle Entwicklungsvorgehen um Methoden des modellbasierten, automatisierten Testens zu ergänzen.

Der modellbasierte Testprozess beschreibt die wesentlichen Testaktivitäten im Gesamtprozess. Es werden für die speziellen Testaufgaben spezifische Lösungen angeboten. Für die frühen Testphasen wird mit MTest eine Methodik zur systematischen Testfallentwicklung basierend auf der Klassifikationsbaummethode bereitgestellt. Für spätere Testphasen, insb. im Zusammenhang mit der HIL-Simulation, stehen grafische Testbeschreibungsmöglichkeiten und mächtige Bibliothekskonzepte bereit. Diese unterschiedlichen Ansätze werden innerhalb von AutomationDesk zu einer gesamtheitlichen Lösung für die moderne Testentwicklung und das Testmanagement vereint.

Testen bezieht sich immer auf die Anforderungen, die an das Testobjekt gestellt werden. Testen kann daher nur erfolgreich sein, wenn die Anforderungen bekannt und formuliert sind, und wenn sie bei der Testspezifikation und -entwicklung berücksichtigt werden. Ansonsten kann Testen nur durch Zufall zum Erfolg führen. Dies wird umso deutlicher, wenn man berücksichtigt, dass bis zu 40% der gefundenen Steuergerätefehler durch unzureichende und unvollständige bzw. unklare Spezifikationen verursacht wird [1]. Testen und Anforderungsanalyse hängen daher sehr eng zusammen. Aus diesem Grund wird Testen in Zukunft weniger als letzter und lästiger Schritt innerhalb der Entwicklungsketten verstanden werden, sondern zunehmend als integraler Bestandteil im gesamten Entwicklungsprozess.

Literatur

1. Hanselmann, H.: Vom Modell zum Seriencode. Electronic Automotive III/2003
2. dSPACE GmbH: Produktinformationen zu TargetLink: http://www.dspace.de.
3. Lamberg, K.; Wältermann, P.: Einsatz der HIL-Simulation zum Test von Mechatronik-Komponenten in der Fahrzeugtechnik. 2. Tagung Mechatronik im Automobil, HdT, München, 2000.
4. Lemp, D.: Opel Vectra Heading for its World Premiere. dSPACE News 1/2002, Paderborn
5. Wältermann, P.; Diekstall, K.; Schütte, H.: Hardware-in-the-Loop-Test verteilter Kfz-Elektroniksysteme. Beitrag zur Tagung Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik. Berlin 2003
6. Hartmann, N.: Automation des Tests eingebetteter Systeme am Beispiel der Kraftfahrzeugelektronik. Diss. Universität Karlsruhe, 2001
7. Ebert, C.; Dumke, R.: Softwaremetriken in der Praxis. Springer Verlag
8. Lamberg, K., Richert, J.; Rasche, R.: A New Environment for Integrated Development and Management of ECU Tests. SAE 2003-01-1024, 2003.
9. Conrad, M.: Beschreibung von Testszenarien für Steuergeräte-Software - Vergleichskriterien und deren Anwendung. 10. Internationaler Kongress Elektronik im Kraftfahrzeug, Baden-Baden (D), 2001
10. Grochtmann, M.; Grimm, K.: Classification Trees for Partition Testing. Software Testing, Verification and Reliability, 3, 63-82, 1993
11. v. Zanten, A.; Erhardt, R.; Landesfeind, K.; Pfaff, G.: Stability Control. In: R. K. Jurgen (Ed.): Automotive Electronics Handbook. 2nd edition, Mc Graw-Hill, 1999
12. OMG: http://www.uml.org.
13. Gühmann, C.; Riese, J.: Testautomatisierung in der Hardware-in-the-Loop Simulation. VDI-Berichte Nr. 1672, 2002.
14. dSPACE GmbH: Produktinformationen zu AutomationDesk und MTest: http://www.dspace.de.