

# **Automatische Testfallgenerierung aus einer formalen Funktionsbeschreibung**

## *Automatic testcase generation based on a formal functional specification*

Dipl.-Ing. **Michael Brost**, FKFS, Stuttgart,

Prof. Dr.-Ing. **Hans-Christian Reuss**, FKFS, Stuttgart

Dr.-Ing. **Rolf Zöller**, Dr. Ing. h.c. F. Porsche AG, Stuttgart

### **Zusammenfassung**

Im Zuge einer verstärkten Qualitätssicherung sowohl bei OEMs als auch Zulieferern gewinnt die Verifikation und Validierung von Steuergerätefunktionen in den letzten Jahren immer mehr an Bedeutung. Wie das Rapid-Prototyping setzt auch der Test durch Automatisierung auf eine Steigerung der Effizienz und Qualität.

Die eigentliche Durchführung von Steuergerätetests hat mittlerweile einen hohen Automatisierungsgrad erreicht. Es haben sich verschiedene Anbieter auf dem Markt etabliert, die sowohl die notwendige Hardware als auch Software als komplette Testwerkzeuge liefern.

Die Erzeugung der notwendigen Testfälle, die Testplanung und die Auswertung der Testergebnisse erfolgen nach wie vor überwiegend manuell. Es ist daher aus Gründen der Steigerung von Effizienz und Qualität wünschenswert, auch in den Phasen vor dem eigentlichen Test, dem Preprocessing, und nach dem Test, dem Postprocessing, den Grad der Automatisierung zu steigern.

Im Rahmen dieses Beitrags soll in der Phase vor dem Test die Erzeugung von Testfällen beleuchtet werden. Es soll ein Weg beschrieben werden, wie aus einer formalen Beschreibung einer Funktion die notwendigen Testfälle automatisiert abgeleitet und ausführbare Testskripte erzeugt werden. Beispielhaft soll dies an einfachen Funktionen aus dem Komfortbereich erläutert werden.

### **Abstract**

In the course of an ever intensifying quality control at both the manufacturer and supplier level the verification and validation of ECU functions have gained more and more importance over the last several years. Like rapid-prototyping testing focuses on automation to further increase efficiency and quality.

The actual execution of ECU tests has already achieved a high degree of automation. Different suppliers have established a market which offers all the necessary hardware and software as complete automatic testing solutions.

The generation of the necessary test cases, the planning of tests and the analysis of the test results are still predominantly carried out manually. It is therefore desirable in order to increase efficiency and quality to also strengthen the degree of automation in the stages before the actual testing – the pre-processing – and after the actual testing – the post-processing.

In the context of this paper the generation of test cases in the stage before the actual test is discussed. A mechanism is shown how the necessary test cases and executable test scripts are automatically generated based on a formal description of ECU functions. Using simple function examples of body electronics the described mechanism is explained.

## 1 Einführung

Im Rahmen eines Kooperationsprojektes zwischen der Dr. Ing. h.c. F. Porsche AG und dem Forschungsinstitut für Kraftfahrwesen und Fahrzeugmotoren Stuttgart (FKFS) soll ein Prozess zur automatisierten Generierung von Testfällen für Kraftfahrzeugsteuergeräte auf Basis einer formalen Spezifikation bzw. Funktionsbeschreibung definiert, umgesetzt und getestet werden. Der Fokus liegt zunächst auf Steuergeräten aus dem Komfortsegment. Ziel ist es, den Testprozess durch eine Ausweitung der Automatisierung in Effizienz und Qualität zu verbessern. Auf den folgenden Seiten wird dieser Prozess skizziert und erläutert.

Wie in [1] und [2] beschrieben und Abb. 1 dargestellt, kann der Testprozess in ein fünfstufiges, sequentielles Phasenmodell unterteilt werden. Der Testdurchführung vorangestellt ist eine sog. Preprocessing-Phase, die sich wiederum in die zwei Teilphasen Testvorbereitung und Testspezifikation aufgliedert. An die eigentliche Testdurchführung gliedert sich eine Postprocessing- bzw. Testabschlussphase.

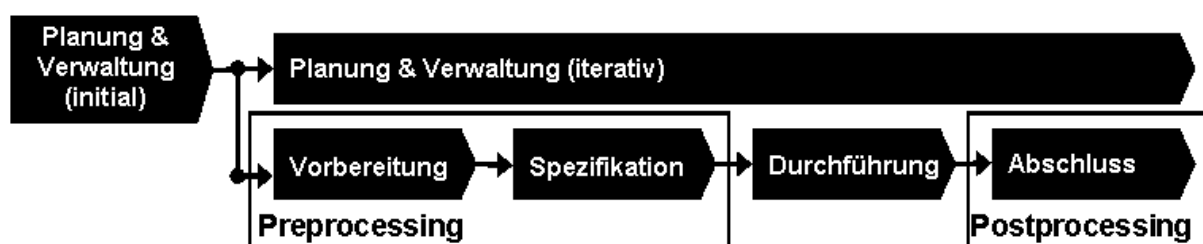


Abb. 1: *Überblick über den fünfphasigen Testprozess mit initialer und begleitender iterativer Planung und Verwaltung. Die hier beschriebene Generierung von Testfällen ist Teil der Spezifikation.*

Die Testvorbereitung umfasst die Beantwortung der Fragen, was getestet werden soll als Priorisierung der Testumfänge, wann getestet werden soll und die Auswahl von Testmethode und Testebene. Die Testspezifikation umfasst neben der Bereitstellung der erforderlichen Ressourcen für die Testausführung und der Definition von Zielgrößen, die in dieser Arbeit und diesem Projekt behandelte Erzeugung bzw. Generierung der Testfälle. Im Postprocessing bzw. in der Testabschlussphase

werden die erzielten Testergebnisse mit dem erwarteten Sollverhalten verglichen und mögliche Fehler am Prüfling aufgezeigt.

Der Testprozess ist iterativ und umfasst alle Ebenen des Absicherungspfades im V-Modell (vergleiche [1], [2], [3] und [4]). Im Zentrum der Betrachtungen stehen in dieser Arbeit die Ebenen 3 und 4 in Abb. 2, d.h. der Test von Steuergeräten und funktionalen Verbänden von Steuergeräten wie dem Komfortsegment. Es handelt sich nach [5] um sog. funktionsorientiertes Testen, bei dem nicht die Implementierung Basis der Testfallgenerierung ist sondern die Spezifikation. Der funktionsorientierte Test ist nicht mit dem Black-Box-Test gleich zu setzen. Es handelt sich nach [6] bei Funktionstests um eine Teilmenge der Black-Box-Tests. Zufallstests sind Black-Box-Tests, die nicht funktionsorientiert sind. An dieser Stelle werden aber beide Begriffe synonym verwendet.



Abb. 2: Die sieben Testebenen des Absicherungspfades im V-Modell.

Da jede Form von automatischer, rechnergestützter Datenverarbeitung nur bei Kenntnis über Format und Inhalt der verarbeiteten Daten erfolgen kann, muss eine automatisierte Testfallgenerierung ebenfalls auf einer formalen Grundlage in Form einer maschinenlesbaren Spezifikation basieren. Diese Grundlage soll eine formale Funktionsbeschreibung auf Basis der Unified Modeling Language (UML) sein.

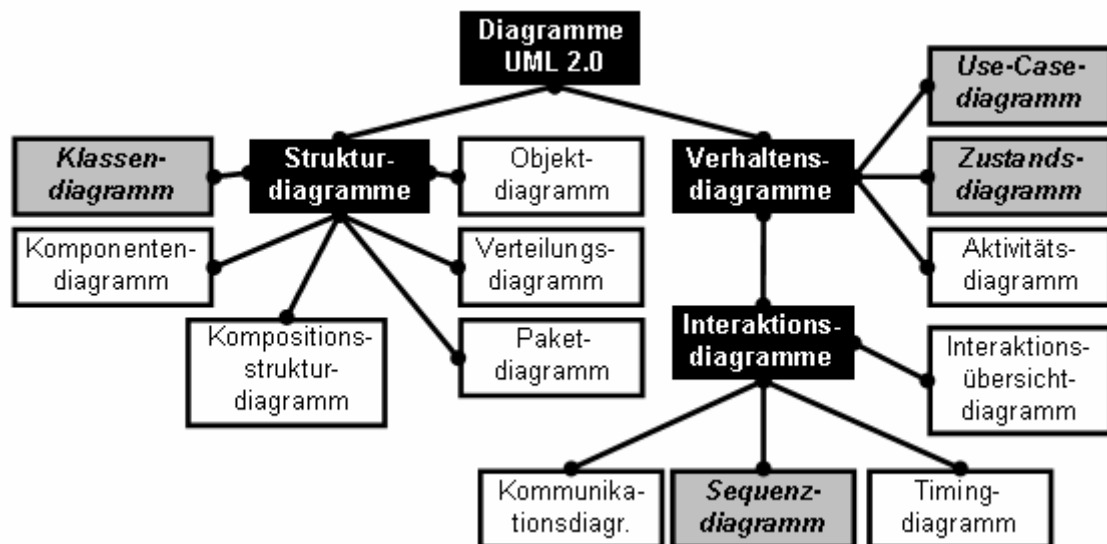
## 2 UML für formale Funktionsbeschreibungen

Die Modellierung eines Tests und einer Funktion sind nicht identisch. Aus einer Funktion folgt mindestens ein Testfall und i.d.R. sind es mehrere. Weiterhin enthält die Modellierung eines Testfalls explizit die Überprüfung von Ausgangs- und

eventuell Eingangsgrößen des zu testenden Systems. Funktionsmodellierungen enthalten diese Überprüfungen – wenn überhaupt – implizit.

Nach eingehender Untersuchung der Möglichkeiten zur formalen Funktionsbeschreibung und unter Berücksichtigung der Randbedingungen des Projekts wurde die UML als Basis der formalen Funktionsbeschreibung ausgewählt.

Die UML in ihrer aktuellen Version 2.0 beschreibt 13 Diagrammtypen (siehe [7]). Nicht alle dieser Diagrammtypen sind für eine formale Funktionsbeschreibung sinnvoll und zielführend. Erste Erfahrungen haben gezeigt, dass die Fülle der Darstellungsformen, die die UML erlaubt, Anwender zunächst überfordert. Eine starke Reduzierung der verwendeten Diagrammtypen ist daher zu empfehlen. Eine erste an [8], [9] und [10] angelehnte Eingrenzung umfasst die in *Abb. 3* kursiv dargestellten und grau hinterlegten Diagrammtypen. Aus den Strukturdiagrammen sind dies nur die Klassendiagramme. Für die Verhaltensmodellierung werden Use-Case-Diagramme mit nachgelagerten Zustands- und Sequenzdiagrammen verwendet.



*Abb. 3: Die 13 Diagrammtypen der UML 2.0, gruppiert in Strukturdiagramme und Verhaltensdiagramme mit der Teilgruppierung der Interaktionsdiagramme.*

Für die Verwendung der UML sprechen ihr Verbreitungsgrad (vergleiche auch [11] und [12]) und die damit einhergehende Werkzeugvielfalt, die Ausdrucksmächtigkeit und ihre inhärente Möglichkeit, durch Profile domänen-spezifische Anpassungen vorzunehmen.

Die Tatsache, dass die UML nicht domänenspezifisch ist, kann als ihr größter Nachteil interpretiert werden. In [12] und [13] werden weitere Nachteile der UML im Zusammenhang mit der Modellierung von Steuergeräten, wie das Thema Echtzeit, detailliert beleuchtet. In der Praxis wird vorwiegend mit an der Domäne angepassten Werkzeugen wie Ascet und Matlab/Simulink gearbeitet. Die UML ist eine Notation und keine Methode (siehe [11]) und es bedarf daher eines domänenspezifischen

Regelwerkes, das festlegt, wie die testgetriebene Modellierung von Steuergeräten bzw. Steuergeräteverbänden auszusehen hat. Ein Teil der weiteren Projektarbeit wird es sein, diese Modellierungsrichtlinien herauszuarbeiten.

Die Modellierung eines Systems wie eines Kraftfahrzeugsteuergerätes basiert auf mindestens zwei Diagrammen. Zum Einen muss die statische Struktur und die angeschlossene Peripherie eines solchen Systems dargestellt werden, d.h. es muss beschrieben werden, an welche Komponenten das System angeschlossen ist, über welche Funktionen diese verfügen und wie diese angeschlossen sind bzw. über welche Schnittstellen das System verfügt. Übertragen auf ein Türsteuergerät könnte das Diagramm beispielsweise die Information enthalten, dass das Steuergerät über einen LIN-Bus an ein Bedienfeld, einen Fensterhebermotor und einen Hall-Sensor angeschlossen ist. Zum Anderen muss das Verhalten des Systems beschrieben werden. Hierfür muss ein Verhaltensdiagramm wie z.B. ein Zustandsdiagramm verwendet werden.

Zur Beschreibung eines Systems auf Makroebene gehören die augenblicklichen Zustandsgrößen  $z(t)$ , die Systemeingänge  $x(t)$  und die Systemausgänge  $y(t)$ . Der funktionsorientierte Test geht davon aus, dass alle Systemzustände unbekannt sind bzw. nicht direkt z.B. durch Messung ermittelt werden können. Der funktionsorientierte Test kann daher nur auf die Ein- und Ausgänge des Systems zugreifen. Es ist bei dieser Betrachtung auch zunächst unwichtig, ob dies Closed-Loop geschieht oder nicht.

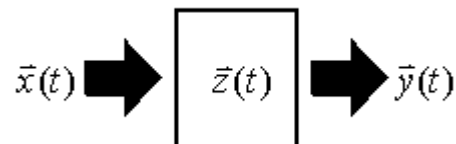


Abb. 4: System mit Zustandsgrößen  $z(t)$  und Systemeingängen  $x(t)$  und Systemausgängen  $y(t)$ .

Eine Funktion wird als Sequenz von definierten Eingangssignalen modelliert. Dies kann als Sequenzdiagramm erfolgen oder als Pfad durch einen Zustandsautomaten, d.h. als eine Folge von Zustandsübergängen. Ausgangspunkt der Funktionsmodellierung ist ein benutzerrelevanter Use-Case (Anwendungsfall), der die Funktion auf einer höheren Abstraktionsebene darstellt. Bei Verwendung von Zustandsautomaten zur Verhaltensmodellierung des Systems muss aus dem gewählten Use-Case der relevante Pfad folgen.

Abb. 5 zeigt das Use-Case Diagramm für die Standardfehlerstimulation jeder Signalleitung eines Steuergerätes. Diese Stimulation besteht aus den drei Fällen „Unterbrechung“, „Kurzschluss“ und „Wackelkontakt“. Bei der „Unterbrechung“ wird die Signalleitung unterbrochen und die erforderliche Reaktion des Steuergerätes geprüft. Erfolgt die Unterbrechung dauerhaft, wird ein statischer Fehler erzeugt. Erfolgt die Unterbrechung nur kurzzeitig, ist das Ergebnis ein sporadischer Fehler.

Der „Kurzschluss“ kann wahlweise gegen Masse oder Batterie durchgeführt werden und sowohl sporadische als auch statische Fehler erzeugen. Der Use-Case „Wackelkontakt“ ist eine Folge von Unterbrechungen mit benutzerdefinierter Frequenz und Anzahl von Wiederholungen und erzeugt einen sporadischen Fehler.

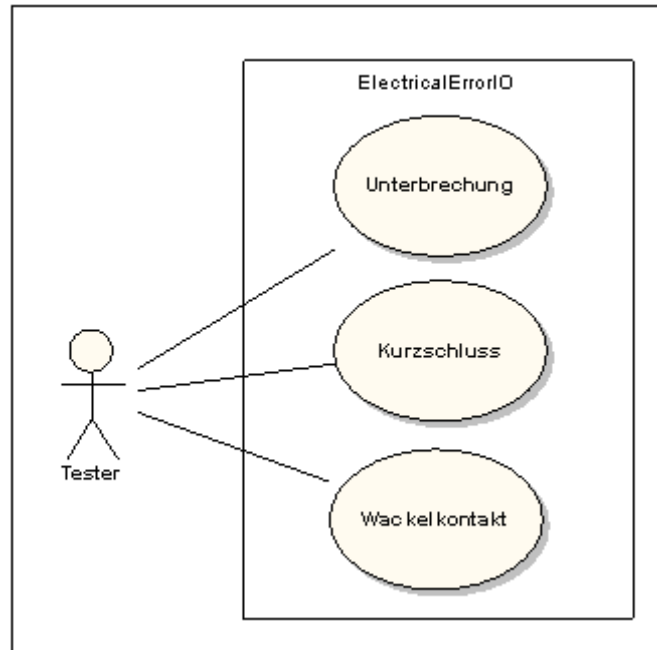


Abb. 5: Use-Case Diagramm für die Standardfehlerstimulation der elektrischen Schnittstelle eines Steuergerätes.

Abb. 6 zeigt den dem Use-Case „Unterbrechung“ hinterlegten Zustandsautomaten. Es ist ein einfacher Zustandsautomat mit drei sequentiellen Zuständen und nur einem möglichen Pfad. Nach der Feststellung, dass die Signalleitung „signal1“ unterbrochen ist, folgt nach spätestens einer Sekunde ein Eintrag im Fehlerspeicher und das Fehlerbit „errorbit1“ auf dem CAN-Bus wird gesetzt. Wie die Unterbrechung der Signalleitung erkannt wird, ist auf dieser Abstraktionsebene unerheblich. Diese Funktion könnte dem Zustand „Leitung unterbrochen“ als neuer Zustandsautomat hinzugefügt werden.

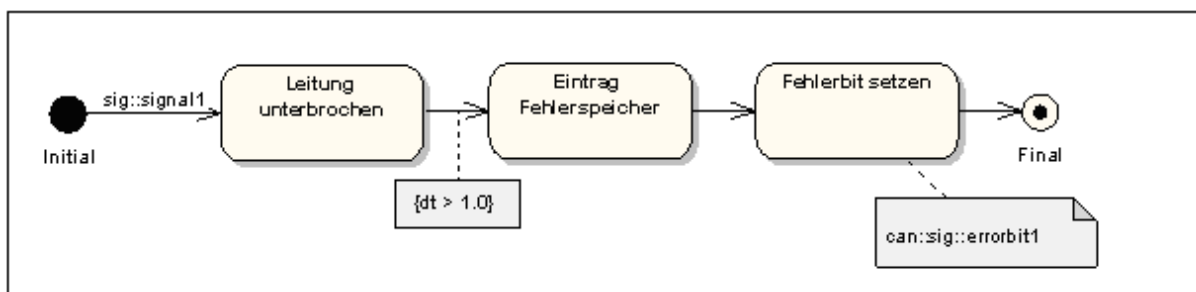


Abb. 6: Zustandsautomat für die Modellierung der Unterbrechung von Signal 1. Der Eintrag in den Fehlerspeicher erfolgt nach spätestens einer Sekunde.

Die oben dargestellte Funktion „Unterbrechung“ ist ein Fall von Misuse und der abgeleitete Testfall prüft beispielsweise die korrekte Implementierung der Diagnosefunktion. Dies ist ein einfacher Fall der Modellierung mittels UML. Die Modellierung komplexer Abläufe ist aufwändiger und wird im Rahmen des Projektes erarbeitet. Allen Anwendungsfällen bzw. Use-Cases ist aber gemeinsam, dass sie sich auf eine Sequenz von Veränderungen der Eingangssignale reduzieren lassen. Dies kann sowohl mit Rückführung der Ausgangssignale als auch ohne erfolgen. Die Rückführung der Ausgänge erfordert ein Modell der Regelstrecke. Im Rahmen dieser Arbeit sind die Unterbrechung von Signalleitungen und andere Manipulationen der Schnittstellen als Veränderung der Eingangssignale definiert.

### 3 Testfallgenerierung

Die Testfallgenerierung setzt auf der Modellierung mittels UML auf. Da es sich bei der UML nur um eine Form der Notation handelt, kann sie nicht direkt maschinell verarbeitet werden. Als formale Schnittstelle dient XMI (**X**ML **M**etadata **I**nterchange), das wie UML ein Standard der OMG (**O**bject **M**anagement **G**roup) ist.

Ausgangspunkte der Testfallgenerierung sind die einzelnen Use-Cases der Funktionsmodellierung. Use-Cases stellen auf sehr abstrakter Ebene Benutzerinteraktionen mit dem System dar. Benutzer können in diesem Fall Personen sein aber auch andere Systeme wie z.B. andere Steuergeräte, die Funktionen des Systems nutzen.

Jeder Use-Case basiert gemäß der Einschränkung aus Abschnitt 2 in diesem Kontext zunächst auf einem Zustandsautomat oder einem Sequenzdiagramm. Eine Verschachtelung dieser Diagramme ist prinzipiell möglich.

In einem ersten Schritt werden die Testfälle erzeugt, die direkt aus den Funktionsmodellierungen folgen, d.h. der Test führt die Funktion gemäß Spezifikation aus. Bei einem Zustandsautomaten wird der Pfad durchlaufen, der durch den Use-Case vorgegeben wird. Sequenzdiagramme werden wie beschrieben ausgeführt.

Die zu entwickelnde Heuristik muss die folgenden Aufgaben übernehmen:

- Die Funktionsmodellierung enthält nur die Stimulation des Systems. Die abgeleiteten Testfälle müssen neben der Stimulation die Ausgaben des Systems überwachen. Die Heuristik muss herleiten, welche Ausgänge wann und in welcher Form gegen welche Referenz geprüft werden sollen.
- Neben der Ausführung der Funktion selbst können auf Basis der Modellierung weitere Testfälle hergeleitet werden. Bei der Verhaltensmodellierung mit einem Zustandsautomaten können nach den folgenden Strategien weitere Testfälle erzeugt werden (siehe auch [14]):
  - „Alle Übergänge“: Durch die Ausführung aller Übergänge eines Zustandsautomaten wird garantiert, dass jeder Zustand erreicht wird.

Diese Strategie zeigt alle fehlerhaften und fehlenden aber keine zusätzlichen Übergänge auf. Sie kann keine fehlerhaften Zustände identifizieren.

- „Alle Round-Trips“: Werden alle Übergangsfolgen, die durch die N+-Strategie ermittelt wurden, erfolgreich durchlaufen, ist die Korrektheit des expliziten Verhaltensmodells nachgewiesen. Es werden alle fehlerhaften und fehlenden Übergänge ermittelt. Fehlerhafte Zustände können nicht eindeutig identifiziert werden.

Die N+-Strategie besteht aus dem Round-Trip-Test und dem Aufspüren von Schleichpfaden, d.h. Zustandsübergänge die nicht ausdrücklich modelliert sind. Zunächst wird von vollständig beschriebenen Zustandsautomaten ausgegangen. Dies macht das Aufspüren von Schleichpfaden überflüssig.

Der Round-Trip-Test erzeugt ausgehend von einem Startzustand einen Übergangsbaum. Jeder Testfall beschreibt eine Übergangsfolge vom Startzustand bis zu jedem Endzustand (Blatt) des Übergangsbaums. Der Testfall prüft am Ende, ob der resultierende Zustand mit dem Übergangsbaum übereinstimmt.

- „M-Längensignatur“: Alle Black-Box-Tests gehen davon aus, dass der zu untersuchende Zustandsautomat opak, d.h. der augenblickliche Zustand nicht einsehbar ist. Die Identifikation eines Zustands erfolgt über eine Sequenz von Ausgabeaktionen des Zustandsautomaten – die Zustandssignatur.

Falls davon ausgegangen wird, dass der Zustandsautomat mit  $n$  Zuständen keine fehlerhaften Zustände hat, kann immer eine Signatur von der Länge  $n+1$  ermittelt werden. Falls der Zustandsautomat aber fehlerhafte Zustände beinhaltet, kann einer dieser Zustände dennoch korrektes Verhalten imitieren. Nach [14] ist es eine Möglichkeit, an dieser Stelle eine Signatur mit der Länge  $M$  zu testen, die länger ist als die Anzahl aller fehlerhaften Zustände. Wie groß  $M$  ist, muss vorher ermittelt und bei der Modellierung berücksichtigt werden.

Für die Herleitung von Testfällen auf Basis von Sequenzdiagrammen sind ebenfalls Methoden herzuleiten und zu untersuchen. Eine Strategie ist der Round-Trip-Test bei dem ein Kontrollflussmodell aus dem Sequenzdiagramm erzeugt wird. Alle möglichen Pfade durch das resultierende Flussdiagramm werden durch eine Sequenz von Eingaben und Zuständen definiert. Diese Sequenz entspricht einem Testfall. Die Strategie wird im Kontext dieses Projektes hinsichtlich ihrer Anwendbarkeit auf den Steuergerätetest untersucht.

Ein Testskript auf Basis der in Abb. 5 und Abb. 6 modellierten Signalunterbrechung und der einfachen „Alle Übergänge“ Strategie für ein fiktives Signal „signal1“ könnte in C-Syntax als Pseudocode wie folgt aussehen:

```
void SignalBreakTest()
    int iRes = 0;
    bool bResult = true;
    init_system();           // Initialisierungsaufgaben

    set_break(signal1);     // signal1 unterbrechen
    wait(1.0);              // 1 Sekunde warten
    iRes = make_dia(0x1234); // Fehlerspeicher auf Eintrag
                           // 0x1234 prüfen
    if (iRes == 0)          // falls Eintrag nicht im
        bResult = false;   // Fehlerspeicher -> Fehler

    iRes = check_can_sig(errorbit1,1); // prüfen ob CAN-Signal
                                       // gesetzt (=1) ist
    if (iRes == 0)          // falls das Fehlerbit nicht
        bResult = false;   // gesetzt ist -> Fehler

    reset_break(signal1);   // Signalunterbrechung beenden
    write2report(bResult);  // Ergebnisse in den Report
return;
```

Das tatsächliche Skript, wie in es in Abschnitt 4 noch erläutert wird, umgesetzt auf ein P.A.T.E.-Testsystem (**P**ersonal **A**utomatic **T**ester for **E**lectronics) und geöffnet im Editor ist in Abb. 7 dargestellt.

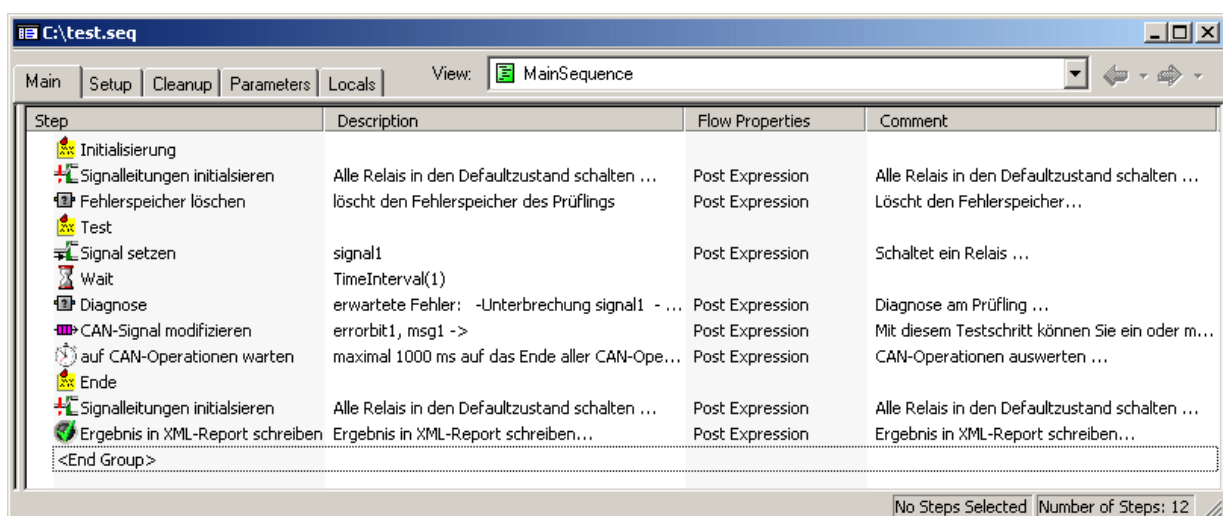


Abb. 7: Darstellung des Testfalls "Signalunterbrechung von signal1" im Editor eines P.A.T.E. Testsystems.

## 4 Anpassung an Testsystem

Damit die generierten Testfälle auch ausgeführt werden können, müssen die Testskripte in ein Format konvertiert werden, das für das jeweilige Testsystem lesbar und ausführbar ist. Zusätzlich zur Konvertierung müssen Parameter, die nicht Teil der Testfallgenerierung sind aber für die Ausführung benötigt werden, hinzugefügt werden.

Erste Zielplattform für die Testdurchführung ist ein vom FKFS entwickeltes P.A.T.E.-System, das als Open-Loop Funktionstester aufgebaut ist. In Abb. 8 wird das System in seiner aktuellen Ausbaustufe gezeigt. In [15] ist die genaue Funktionsweise und der Systemaufbau erläutert. Das System zeichnet sich zum Einen durch die Möglichkeit aus, zwei Steuergeräte parallel zu testen und zum Anderen durch den Aufbau als Universaltester. Durch die strikte Trennung von testsystemspezifischen und steuergerätespezifischen Parametern, erlaubt das Testsystem den Test aller Steuergeräte, für die es aus Hardwaresicht eine Übermenge darstellt. P.A.T.E.-Testsysteme wurden bereits mehrfach für verschiedenste Prüflinge, von ESP-Steuergeräten bis hin zu Bordnetzsteuergeräten, eingesetzt und haben sich während der Serienentwicklung bewährt.



Abb. 8: Das P.A.T.E. Testsystem für den Komfortbereich.

Bei der Konvertierung für das P.A.T.E.-Testsystem werden die auf XML basierenden Testfallbeschreibungen in benutzerdefinierten Gruppen eingelesen. Über den COM-Server (**C**omponent **O**bject **M**odel) der Testausführungsmaschine werden dann ablauffähige Testskripte erzeugt. Zusätzliche Parameter, wie z.B. die textuelle Beschreibung von Fehlerspeichereinträgen, die für die Ausführung notwendig sind, werden über eine **allgemeine Parametrierschnittstelle (AIPas)** zur Verfügung gestellt (siehe auch Übersicht in Abb. 9).

Unter Verwendung eines Profils für das jeweilige Testsystem werden bei der Testskripterzeugung bestimmte Sonderaspekte des jeweiligen Testsystems berücksichtigt. Wird beispielsweise eine Signalunterbrechung in das Testskript eingefügt, kann das Profil festlegen, dass nach dieser, mit einem Relais vollzogenen Unterbrechung eine Pause von n ms eingefügt wird, um die Entprelldauer des Relais zu berücksichtigen. Über eine Plugin-Architektur ist es möglich, weitere Testsysteme zu unterstützen.

## 5 Prozessübersicht

In Abb. 9 ist eine Gesamtübersicht über den an dieser Stelle vorgestellten Prozess zur automatisierten Testfallgenerierung auf Basis formaler Funktionsbeschreibungen in UML dargestellt.

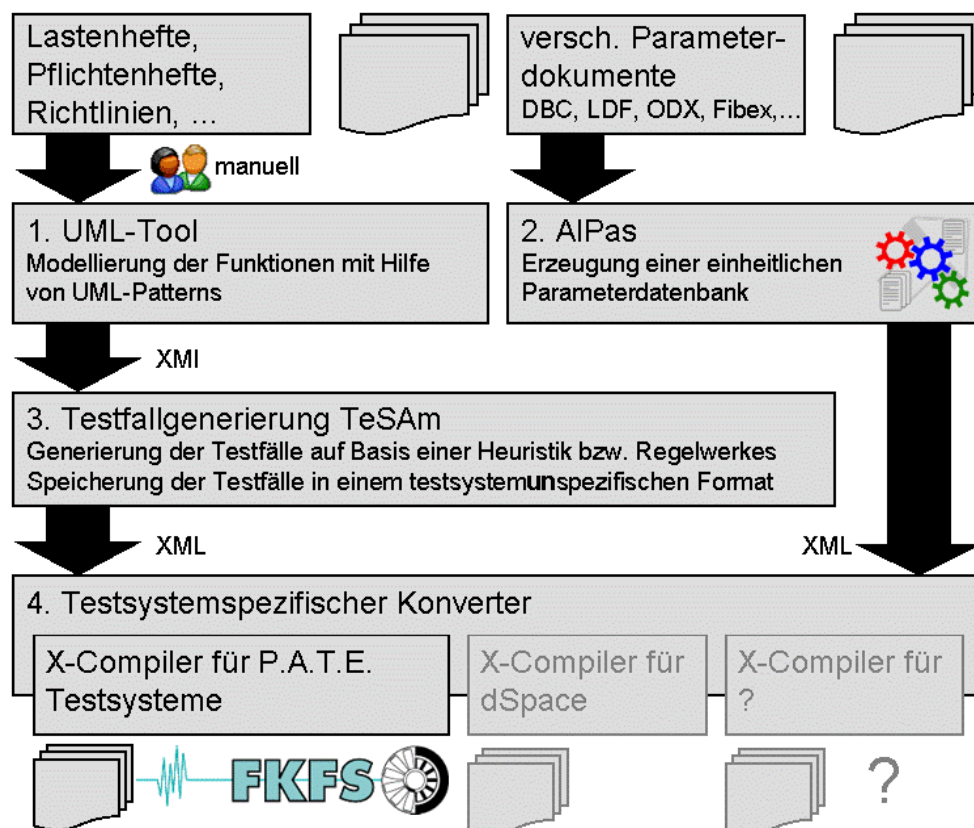


Abb. 9: Gesamtübersicht über den Prozess zur automatisierten Testfallgenerierung auf Basis formaler Funktionsbeschreibungen in UML.

Formale und maschinenlesbare Parameterdokumente wie logische Busbeschreibungen und nicht formale Dokumente wie Lastenhefte und Konzernrichtlinien bilden zunächst die Grundlage der Testfallgenerierung. Die maschinenlesbaren Dokumente werden über das Werkzeug AIPas eingelesen, aufbereitet und dann zusammengefasst als XML-Dokument dem testsystemspezifischen Konverter zur Verfügung gestellt. AIPas verfügt über eine eigene Versionierung und ein Projektmanagement auf Basis von XML.

Um beispielsweise ein bestimmtes CAN-Signal während der Testausführung zu verändern, ist es notwendig, auch dessen Bitposition, Skalierung, Offset usw. zu kennen. Es ist aber nicht sinnvoll und notwendig, all diese Informationen schon bei der Funktionsmodellierung zu berücksichtigen. Ein Identifikationsmerkmal, wie z.B. der Signalname reicht zunächst aus, um die notwendigen Parameter zu einem späteren und aus Gründen der Handhabung günstigeren Zeitpunkt hinzuzufügen.

Für die Funktionsmodellierung in UML kann prinzipiell jedes UML-Tool verwendet werden, das UML 2.0 konform ist und über einen XMI-Export verfügt. Die Modellierung in dieser Arbeit wird zunächst mit dem im Rahmen von Autosar verwendeten Werkzeug Enterprise Architect durchgeführt.

Die eigentliche Testfallgenerierung wird mit dem zu entwickelnden Testskriptautomat TeSAM durchgeführt. Über den eingebauten XMI-Import werden die modellierten Funktionen eingelesen. Das Werkzeug erzeugt anschließend unter Anwendung der in Abschnitt 3 definierten Heuristik Testfälle und legt diese in einem nicht testsystem spezifischen Format ab. Vorlage für dieses Format ist die sich im Standardisierungsprozess der IEEE (Institute of Electrical and Electronics Engineers) befindliche Automatic Test Markup Language ATML (siehe [16] und [17]). Dieser Standard definiert mehrere XML-Schemata, die alle Informationen zur Durchführung eines Testes inklusive der Datenorganisation und Partitionierung enthalten. XML-Daten eignen sich gut für eine automatisierte Verarbeitung.

Aus den erzeugten Testfällen müssen ausführbare Testskripte für eine oder auch mehrere Zieltestsysteme erzeugt werden. Dies geschieht, wie in Abschnitt 4 dargestellt, über einen Konverter oder auch Crosscompiler, der systemspezifische und ablauffähige Testskripte erzeugt. Es wird zunächst eine Unterstützung für P.A.T.E.-Testsysteme implementiert.

## **6 Vorteile**

Die Einführung eines solchen Prozesses bietet die folgenden Verbesserungen der Tests von Steuergeräten.

Die automatisierte Testfallgenerierung kann nach Etablierung des Prozesses einen erheblichen Gewinn an Zeit und Qualität schaffen. Testfälle sind bereits nach Erstellung eines formalen Lastenheftes bzw. einer formalen Funktionsbeschreibung verfügbar. Eine formale Funktionsbeschreibung lässt weniger Spielraum für

Interpretation und steigert somit die Qualität der darauf aufbauenden Testfälle speziell bei Regressionstests.

Ein nicht testsystemspezifisches Format für Testskripte ist Voraussetzung für einen integrativen Testprozess und erlaubt beispielsweise den Austausch von Testskripten zwischen OEM und TIER-1 Supplier. Es ist darüber hinaus Grundlage für die Austauschbarkeit von Testsystemen. Dies ist vor allem bei Gemeinschaftsentwicklungen, bei denen die Heterogenität der Testlandschaft zweier oder mehrerer Unternehmen zu bewältigen ist, von Vorteil.

Im Zuge einer auch im Embedded Bereich zunehmenden Objektorientierung (siehe [18]) in der Softwareentwicklung ist es ein vorstellbares Szenario, dass für die Modellierung zumindest bestimmter Teile der Steuergerätesoftware auf UML zurückgegriffen wird. Dieser Ansatz bietet die Möglichkeit, direkt aus dieser Quelle alle notwendigen Testfälle zu erzeugen und somit den Aufwand für die Testfallerzeugung auf ein absolutes Minimum herunterzufahren.

## **7 Ausblick**

Der hier skizzierte Weg von einer formalen Funktionsbeschreibung zu einer testsystemunabhängigen Darstellung der resultierenden Testfälle hin zu ausführbaren Testskripten wird im Rahmen eines Entwicklungsprojektes mit mehreren Steuergeräten aus dem Komfortsegment verfeinert und ausgebaut.

Neben der Erstellung der formalen Funktionsbeschreibung für Teilumfänge dieser Steuergeräte und den sich ergebenden Modellierungsrichtlinien kommt der Präzisierung der Heuristik zur Testfallgenerierung speziell in Hinblick auf ihre allgemeine Anwendbarkeit besondere Bedeutung zu. Es ist geplant, das Werkzeug TeSAM weiter auszubauen und die definierte Heuristik in automatisierter Form umzusetzen. Die Umsetzung hin zu ausführbaren Testskripten wird durch die Einbindung neuer Zielplattformen ergänzt.

Die operative Implementierung dieser Werkzeug- und Prozesskette wird begleitet von einer Untersuchung hinsichtlich Akzeptanz bei allen am Testprozess beteiligten Personen sowie der zu erzielenden Effizienzsteigerungen.

Weiterhin ist es Ziel, bestehende oder sich entwickelnde Anstrengungen hinsichtlich der Standardisierung in das vorgestellte Prozessmodell zu integrieren. Dies ist für die verschiedensten Formate von Bus- und Diagnosebeschreibungen – seien sie proprietär oder standardisiert – bereits geschehen und wird beispielsweise mit einer genauen Analyse von Autosar hinsichtlich einer Verwertung für den Test fortgeführt. Aspekte, die ebenfalls in der Testfallgenerierung Berücksichtigung finden müssen, sind das Variantenmanagement und die Ermittlung einer funktionalen Testabdeckung.

## 8 Literatur

- [1] Meisenzahl, J.; Vollerthun, A.; de la Cruz, M.: „*Innovation und Zuverlässigkeit durch systematische Testplanung und Testspezifikation*“, VDI-Tagung „Elektronik im Kraftfahrzeug“, Baden-Baden, 2005
- [2] Zöller, R.; Dorn, R.; Koley, A.; Marx, D.: „*Prozess SPRINT – Anpassung und Einbettung etablierter Software-Entwicklungsprozesse in die Gesamtfahrzeug-Entwicklung der Porsche AG*“, VDI-Tagung „Elektronik im Kraftfahrzeug“, Baden-Baden, 2003
- [3] Lange, K.: „*Erreichen einer hohen Zuverlässigkeit in elektronischen Systemen mit Hilfe von strukturiertem Testen*“, Tagung „Der Weg zu optimierten Testprozessen“, Stuttgart, 2004
- [4] Schäuffele, J.; Zurawka, T.: „*Automotive Software Engineering*“, Vieweg Verlag, Wiesbaden, 2. Auflage, 2004
- [5] Spillner, A.; Linz, T.: „*Basiswissen Softwaretest*“, Dpunkt Verlag, Heidelberg, 1. Auflage, 2002
- [6] Liggesmeyer, P.: „*Software-Qualität*“, Spektrum Akademischer Verlag, Heidelberg, 2002
- [7] Jeckle, M.; Rupp, C.; Hahn, J.; Zengler, B.; Queins, S.: „*UML2 glasklar*“, Carl Hanser Verlag, München, 1. Auflage, 2004
- [8] Object Management Group: „*OMG SysML Specification*“, Needham, Massachusetts, 2006
- [9] Gross, H.-G.: „*Testing and the UML – A Perfect Fit*“, IESE-Report No. 110.03/E, Fraunhofer Institut für Experimentelles Software Engineering, Kaiserslautern, 2003
- [10] Rumpe, B.: „*Modellierung mit UML*“, Springer Verlag, Heidelberg, 2004
- [11] v.d. Beeck, M.: „*Eignung der UML 2.0 zur Entwicklung von Bordnetz-architekturen*“, Tagungsband „Modellbasierte Entwicklung eingebetteter Systeme II“, Informatik Bericht TU Braunschweig, 2006
- [12] Götze, M.; Kattaneck, W.: „*Erfahrungen mit der UML beim Entwurf von Kfz-Steuerungen*“, ITG/GI/GMM Workshop „Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen“, Meißen, 2001
- [13] Dimitrow, W.; Sporer, M.; Hardt, W.: „*UML basierte Zeitmodellierung für eingebettete Echtzeitsysteme*“, Chemnitzer Informatik-Berichte, Technische Universität Chemnitz, 2006

- [14] Binder, R.: „*Testing Object-Oriented Systems. Models, Patterns, Tools.*“, Addison Wesley Verlag, München, 1999
- [15] Brost, M.; Baumann, G.; Reuss, H.-C.: „*P.A.T.E. – Eine offene Plattform für automatisierte Steuergeräte-Tests*“, Tagungsband „Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik“, Berlin, 2005, Expert Verlag, Renningen, 2005
- [16] IEEE Standards Coordinating Committee 20, „*Automatic Test Markup Language - Overview*“, IEEE Standards Activities Department, New Jersey, 2004, <http://grouper.ieee.org/groups/scc20/tii/>
- [17] IEEE Standards Coordinating Committee 20, „*Standard for Standard Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML: Exchanging Test Descriptions - Draft*“, IEEE Standards Activities Department, New Jersey, 2005, <http://grouper.ieee.org/groups/scc20/tii/>
- [18] Xcc Software AG, „*Embedded Trends 2005/2006 - Eine Marktumfrage zu aktuellen Themen der Embedded Software-Entwicklung*“, Karlsruhe, 2006