

# Modell- und Software-Verifikation vereinfacht

Zur grafischen modellbasierten Entwicklung von Funktions-Software wird in vielen Bereichen Matlab/Simulink verwendet. Mit Hilfe automatischer Code-Generierung können diese Modelle schnell in echtzeitfähige Software überführt werden. Die frühzeitige Simulation der Modelle ermöglicht es, in jedem Entwicklungsschritt die Eigenschaften der Software systematisch zu überprüfen und sicherzustellen. Die dabei eingesetzten Testwerkzeuge verwendeten statische Signalverläufe, die rückwirkungsfrei das zu testende System stimulieren. Die anschließende Testauswertung erfolgte durch manuelle Analyse der sich ergebenden Signalverläufe oder durch eine entwicklungsbegleitende Regressionsanalyse. Diese Vorgehensweise weist eine Reihe von Nachteilen auf:

- ▶ Komplexe Funktionsalgorithmen erfordern aufwendige, komplexe Signalverläufe zu ihrer Absicherung. Diese sind wenig intuitiv und nur schwer verständlich.
- ▶ Durch die statischen, vom Systemzustand unabhängigen Stimuli sind Tests reaktiver Funktionen zeitintensiv und oft nur iterativ ermittelbar.
- ▶ Die Ableitung von Referenzsignalverläufen erfolgt durch Freigabe von manuell als richtig erkannten Testobjektausgaben.
- ▶ Die Differenzbildung zwischen Testergebnissen und Referenzsignalen ist empfindlich gegenüber Änderungen des Testobjekts.

Diese Faktoren adressiert Time Partition Testing (TPT), das ursprünglich von der Forschung der Daimler AG entwickelt wurde und heute in Form eines kommerziellen Software-Werkzeugs durch die PikeTec GmbH weiterentwickelt wird, in dreifacher Hinsicht:

- ▶ Testdefinition mittels hierarchischer Automaten: Das Testsystem ist ein-

## Zeitkritische Software-Anforderungen effizient testen

Testfälle für Software-Funktionen werden typischerweise durch Skripte oder Signalverläufe implementiert. Sie sind kompliziert zu erstellen, aufwendig zu warten und nur durch visuelle Inspektion zu bewerten. Außerdem können sie in anschließenden Stufen des Entwicklungsprozesses nur selten wieder verwendet werden. Intuitive Methoden und Werkzeuge zur grafischen, regelbasierten Implementierung komplexer Testabläufe waren bisher nicht verfügbar. In diese Lücke zielt die Methode „Time Partition Testing“ (TPT).

Von Menno Mennenga, Christian Dziobek und Iyad Bahous

fach parametrier- und wartbar; es wird auch für Nicht-Programmierer verständlich.

▶ Testauswertung durch eine besondere Auswertesprache: Zeitliches und Funktions-Verhalten des Testobjekts kann nicht nur strikt quantitativ, sondern auch qualitativ einfacher beurteilt werden. Die visuelle Inspektion nach jedem Testdurchlauf entfällt.

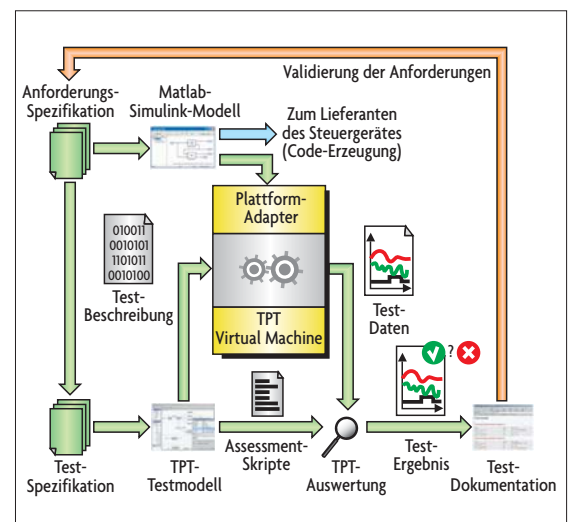
▶ Verkopplung von Testsystem und Testobjekt (Closed-Loop-Test): Während der Testdurchführung wird auf das Systemverhalten in Echtzeit reagiert. Rückgekoppelte Systeme wie Regelkreise können in einer Iteration getestet werden.

Die Methode Time Partition Testing wird seit einigen Jahren erfolgreich bei der Daimler AG in einer Reihe von Projekten im Rahmen der modellbasierten Funktionsentwicklung verwendet. So wird TPT z.B. bei der Entwicklung von Komfort- und Innenraumfunktionen für die aktuelle C-Klasse von Mercedes-Benz eingesetzt. Bei der MB-technology GmbH wird TPT in unterschiedlichen Projekten eingeführt und angewendet. Ein Beispiel ist die modellbasierte

Software-Entwicklung der Power Control Unit eines Diesel-Hybrid-Linienbusses.

## Modellbasierte Funktionsentwicklung mit Matlab und TPT

Bei diesen Projekten geht die Entwicklung eines Matlab/Simulink-Modells der eigentlichen Software-Implementierung voraus, um die Anforder-



! Bild 1. Entwicklungsprozess für Steuergeräte-Software.

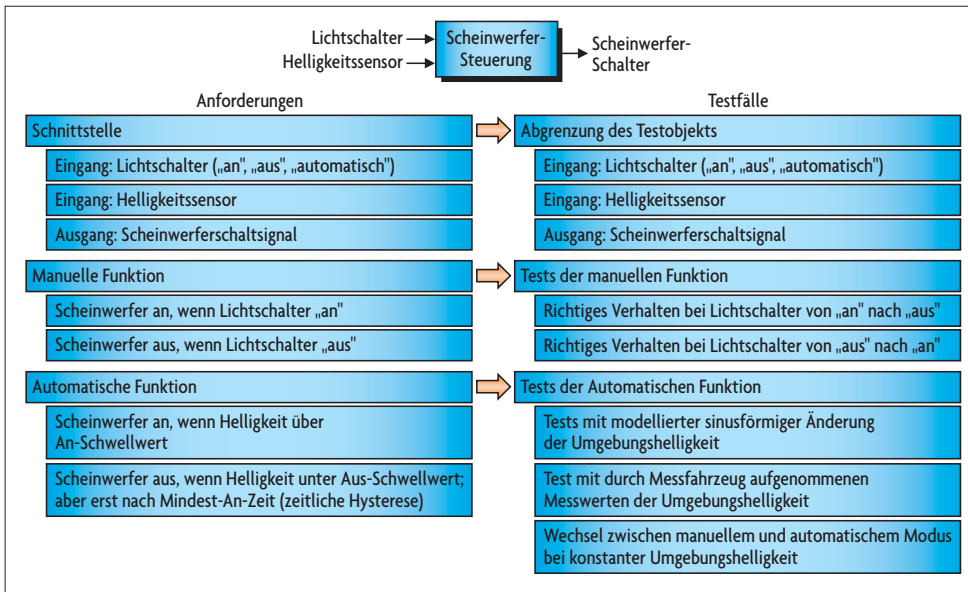


Bild 2. Anforderungen an die Scheinwerfersteuerung.

runngsspezifikation möglichst früh validieren zu können. Dieser Ansatz bietet folgende Vorteile:

- ▶ Zu einem frühen Zeitpunkt kann beurteilt werden, ob die gestellten Anforderungen den angestrebten Fahrzeugcharakter verwirklichen.
- ▶ Vor Weitergabe der Spezifikation an den Zulieferer werden Lücken aufgedeckt. Dadurch werden Nachspezifi-

kationen vermieden und das Risiko erhöhter Entwicklungskosten durch ungeplante Umfänge gesenkt.

- ▶ Im Modell aufgedeckte Fehler führen zu einer geringeren Fehlerzahl in der Implementierung und somit zu einer höheren Software-Qualität.

Bild 1 verdeutlicht den Entwicklungsprozess. Die Erstellung der Steuergeräte-Software beginnt mit der Entwicklung eines Matlab/Simulink-Modells auf Basis der Anforderungsspezifikation. Daraus wird die Testspezifikation abgeleitet und ein Modell-Testsystem auf Basis von TPT erstellt. Fertig gestellte Funktionsmodelle werden an den Zulieferer übergeben. Dieser erzeugt aus den Funktionsmodellen mit Hilfe automatischer Code-Generatoren produktionsstauglichen C-Code und sorgt für dessen Implementierung auf dem Steuergerät. Die Implementierung wird beim Zulieferer mit TPT verifiziert (Software-in-the-Loop), um mögliche Implementierungsfehler aufzudecken. Über mehrere Zyklen wird schließlich ein stabiler Release-Stand erreicht.

Im Vergleich zu früheren Software-Projekten ist durch die Modellierung einer ausführbaren Spezifikation ein zusätzlicher Entwicklungsschritt hinzugekommen. Dafür ist der Aufwand für die Implementierung gesunken, da die Code-Generierung automatisch aus dem Modell heraus erfolgen kann. Jedoch ist auch der Beitrag von TPT zur

Produktivität des Teams nicht von der Hand zu weisen: Während der Implementierung gewonnene Erkenntnisse führen wiederholt zu Änderungen der Anforderungsspezifikation. Diese erfordern Anpassungen der Testspezifikation und des Testsystems. Mit Hilfe von TPT kann der dafür notwendige Aufwand im Vergleich zu früher verwendeten Werkzeugen reduziert werden. TPT erlaubt es, Änderungen einfach in das Testsystem einzupflegen. Erreicht wird dies durch eine übersichtliche Stimulusgenerierung sowie eine effiziente Testauswertung.

### TPT im praktischen Einsatz

Im Folgenden wird anhand eines Beispiels aus der C-Klasse-Entwicklung der praktische Einsatz von TPT erläutert. Hierbei handelt es sich um eine Scheinwerfersteuerung, die wahlweise manuell oder automatisch (gesteuert durch einen Helligkeitssensor) arbeiten kann. Bild 2 zeigt eine Auswahl der Anforderungen an die Steuerung und die daraus abgeleiteten Testfälle. Ein Testingenieur, der die Testfälle implementiert, würde die folgenden Schritte durchführen.

### Deklaration

Ein TPT-Testprojekt beginnt mit der Deklaration von Parametern, Konstanten und Variablen unter Verwendung des Deklarationseditors (im Verlaufe des Projektes können weitere hinzugefügt werden). Neben Parametern und Konstanten werden vier Kanäle definiert. Sie dienen als interne Variablen im Testsystem. Deklarationen bestehen aus einem Namen und einem Typ wie Byte, Integer und Double.

### Schnittstelle zum Testobjekt

Die Schnittstelle zum Testobjekt wird über die Signatur des Projekts festgelegt. Mit dem Signatureditor werden die im Deklarationseditor festgelegten Kanäle zu Ein- und Ausgängen bestimmt. Sie dienen bei der Testausführung zur Kommunikation zwischen TPT und dem Testobjekt durch die Virtual Machine (VM). Die VM ist durch ihre Implementierung in ANSI-C auf verschiedene Testsysteme portier-

Testlet „Initialize Light Switch“			
Signatur	Eingang	Local	Ausgang
	–	–	Light_Switch
Szenarien	Name	Definition	
	On	Light_Switch(t) := LIGHT_ON	
	Off	Light_Switch(t) := LIGHT_OFF	
	Automatic	Light_Switch(t) := LIGHT_AUTO	
Testlet „Constant Light Intensity“			
Signatur	Eingang	Local	Ausgang
	–	Oscillation	Light_Intensity
Szenarien	Name	Definition	
	Light	Light_Intensity(t) := 100	
	Dark	Light_Intensity(t) := 0	
Testlet „Changing Light Intensity“			
Signatur	Eingang	Local	Ausgang
	–	Oscillation	Light_Intensity
Szenarien	Name	Definition	
	Variation at Threshold	Oscillation(t) := 25 × sin(t/10)	
	Created Intensity	Light_Intensity(t) := 50 + oscillation(t)	
	Driving Cycle	siehe Bild 3	

Tabelle 1. Signatur und Szenarien der drei Testlets

bar. Sie ermöglicht – zusammen mit den entsprechenden Treibern – sowohl den Test eines Matlab-Modells auf dem PC als auch den Test eines Steuergerätes auf einem HiL-Teststand.

### Stimulusgenerierung

Nach Abschluss der initialen Deklaration wird das Testprojekt aus Testlets aufgebaut. Testlets sind gekennzeichnet durch

- ▶ eine Signatur (Eingänge, Ausgänge, und interne Variablen (Locals) zur Speicherung von Zwischenwerten oder Zuständen),
- ▶ einen optionalen Parametersatz,
- ▶ die Verhaltensbeschreibung mittels Szenarien (minimal eines) zur Testvektorgenerierung,
- ▶ eine Testauswertung (durch Assessment-Skripte).

Im vorliegenden Beispiel werden drei Testlets auf der Arbeitsfläche

platziert: „Initialize Light Switch“, „Constant Light Intensity“ und „Changing Light Intensity“. Das erste Testlet übernimmt das Stimulieren des Lichtschalteneingangs der Scheinwerfersteuerung, die beiden anderen werden zeitlich parallel zur Stimulierung des Helligkeitseingangs verwendet. In „Initialize Light Switch“ werden später auch die Assessment-Skripte zur Testauswertung implementiert.

Im Signatoreditor wird jedem Testlet eine Auswahl der im Deklarationseditor festgelegten Signale als Eingänge, Ausgänge oder Hilfsvariablen (Locals) zugeordnet (**Tabelle 1**; der Kanal Oscillation wird als interne Hilfsvariable verwendet). Ein Doppelklick auf ein Testlet öffnet ein Auswahlfenster, in welchem die Methodik der Verhaltensbeschreibung abgefragt wird. Zur Verfügung stehen:

- ▶ Direct Definition (Signalerzeugung),
- ▶ Time Partitioning (hierarchischer Aufbau aus anderen Testlets)

- ▶ und Reference (Ableitung aus einem bestehenden Testlet).

Für alle Testlets wird Direct Definition gewählt. Solche Testlets bilden im hierarchischen Aufbau eines Testprojekts die unterste Ebene. Sie wirken direkt auf die Eingänge des Testobjekts. Die Festlegung der Signalverläufe erfolgt mittels Szenarien durch manuell erstellte Signalverläufe, importierte Daten, Konstanten oder Formeln. **Tabelle 1** zeigt Signatur und Szenarien der drei Testlets.

Auf Basis der Testlets erfolgt der Aufbau komplexerer Strukturen mittels Time Partitioning, wie in **Bild 3** dargestellt. „Constant Light Intensity“ und „Changing Light Intensity“ werden durch Zustandsübergänge (Transitions) miteinander verbunden. Ein Übergang kann mehrere alternative Übergangsbedingungen enthalten. Die Ableitung von TP-Szenarien (Time Partitioning) erfolgt in den folgenden zwei Schritten:

- ▶ Pfadauswahl: Aktivieren von Transaktionen durch Anklicken. Von aktivierten Transitionen berührte Testlets werden automatisch aktiviert.
- ▶ Parametrierung der Übergänge und Testlets: Für jeden aktivierten Zustandsübergang wird durch Rechtsklicken eine Übergangsbedingung ausgewählt, und für jedes aktivierte Testlet wird durch Rechtsklicken eines der zur Verfügung stehenden Szenarien ausgewählt.

Bild 3 zeigt, dass sich aus dem Testlet ein Zustandsautomat ergibt, welcher die Steuerung des Signals Light\_Intensity übernimmt: Die Ausführung der verwendeten Szenarien wird zur Testlaufzeit nacheinander aktiviert und deaktiviert.

Während das ursprüngliche, im oberen Bereich von Bild 3 gezeigte Testlet mehrere Anfangszustände haben konnte, ist das für die Szenarien nicht mehr möglich. Wird eine Anfangsbedingung aktiviert, führt die Aktivierung einer anderen automatisch zu ihrer Deaktivierung.

Mit Hilfe von TPT werden so einfach hierarchisch aufgebaute, konfigurierbare, parallele Zustandsautomaten geschaffen.

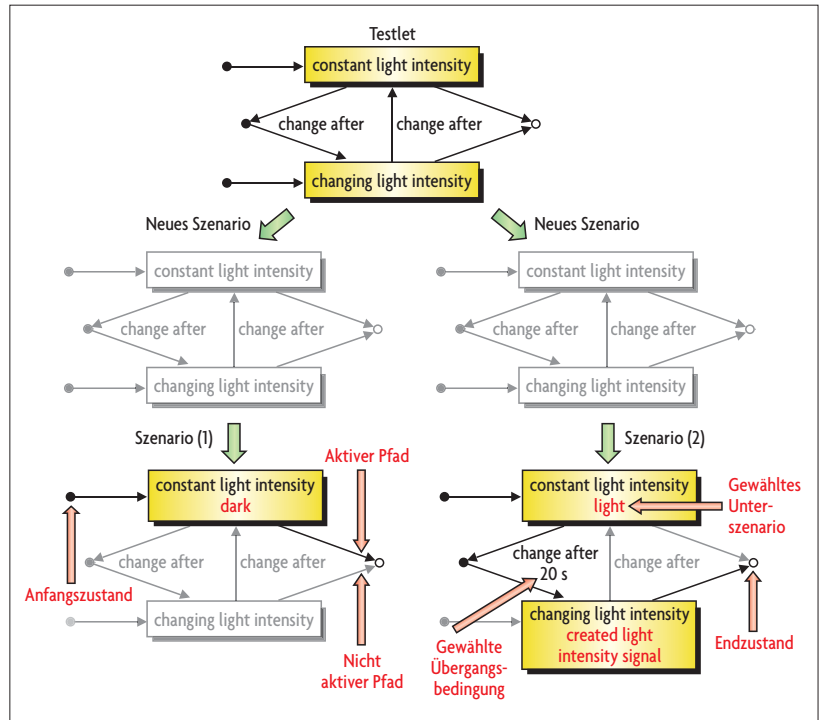


Bild 3. Erstellen von TP-Szenarien.

### Testauswertung auf Basis von Python

Die Beurteilung des Testobjektverhaltens erfolgt durch Auswertungs- oder Assessment-Skripte. Globale Assessment-Skripte werden während der gesamten Laufzeit eines Tests ausgeführt. Lokale Skripte sind einem bestimmten Testlet zugeordnet (Ausführung nur bei Aktivierung des entsprechenden Testlets bzw. eines seiner Szenarien). So kann das Testobjektverhalten abhängig von der Betriebsart, z.B. Normalbetrieb, Übertemperatur oder Notbetrieb, unterschiedlich beurteilt werden.

Assessment-Skripte werden in TPT auf der Basis von Python erstellt. Python wurde von PikeTec um Sprachelemente erweitert (Schlüsselwörter, Typen und Methoden), welche die Testauswertung vereinfachen. Das Listing zeigt einen Ausschnitt aus dem Assessment-Skript des Szenarios Automatic („Initialize Light Switch“) zum Test des Automatikmodus.

Das Schlüsselwort *during* (Zeile 2) erlaubt es ähnlich einer *if*-Anweisung, Code-Abschnitte nur bei Eintritt bestimmter Bedingungen auszuführen. Dabei werden neben herkömmlichen auch zeitliche reguläre Ausdrücke akzeptiert. Ein Beispiel liefert das Argument von *during* in Zeile 2: Der Ausdruck ist wahr, wenn der Helligkeitswert zum aktuellen Zeitpunkt *t* kleiner als der Anschaltenschwellwert ist. Innerhalb des von *during* eingefassten Codes ermöglicht TPT den

Assessment-Skripte werden in TPT auf der Basis von Python erstellt. Python wurde von PikeTec um Sprachelemente erweitert (Schlüsselwörter, Typen und Methoden), welche die Testauswertung vereinfachen. Das Listing zeigt einen Ausschnitt aus dem Assessment-Skript des Szenarios Automatic („Initialize Light Switch“) zum Test des Automatikmodus.

```

1: # Test des Hystereseverhalten unter sinkender Umgebungshelligkeit
2: during TPT.regexp([light_intensity(t) < INTENSITY_LIMIT_LIGHT_ON]):
3:     # Scheinwerfer noch aus im aktuellen Zeitintervall
4:     if headlight(this.getStartime()) == false:
5:         # Hysteresezeit NICHT verstrichen
6:         during TPT.regexp([(t) < HYSTERESE_TIME_ON]):
7:             if TPT.always(headlight(t) == false):
8:                 # Verdikt: richtig
9:                 automatic_ok := true;
10:            else:
11:                # Verdikt: falsch
12:                automatic_ok := false;
13:            # Hysteresezeit IST verstrichen
14:            during TPT.regexp([(t) > HYSTERESE_TIME_ON]):
15:                if TPT.always(headlight(t) == true):
16:                    # Verdikt: richtig
17:                    automatic_ok := true;
18:                else:
19:                    # Verdikt: falsch
20:                    automatic_ok := false;
21:            # Scheinwerfer eingeschaltet im aktuellen Zeitintervall
22:        else:
23:            if TPT.always(headlight(t) == true):
24:                # Verdikt: richtig
25:                automatic_ok := true;
26:            else:
27:                # Verdikt: falsch
28:                automatic_ok := false;

```

Ausschnitte aus dem Assessmentskript für den Test des Automatikmodus

Zugriff auf den Zeitabschnitt, in welchem die Bedingung erfüllt ist. Der Operator *this* liefert dazu die Instanz des aktuellen Zeitintervalls, des so genannten Kontextintervalls. So wird in Zeile 4 mittels *this.getStartTime()* der Zustand der Scheinwerfer zu dem Zeitpunkt geprüft, als die Umgebungshelligkeit unter den Einschaltenschwellwert sank. War die Bedingung zwischenzeitlich nicht erfüllt und wird sie anschließend wieder wahr, liefert *this* eine neue Zeitintervallinstanz. Für den Testingenieur entfällt auf diese Weise das Programmieren globaler oder ereignisabhängiger Zähler. Trotzdem können Tests auf die Richtigkeit von Ereignissen hinsichtlich der absoluten Zeit oder einer relativen zeitliche Abfolge übersichtlich implementiert werden.

In den Zeilen 9, 12, 17, 20, 25 und 28 wird die Variable *automatic\_ok* je nach Testerfolg auf wahr oder falsch gesetzt. Dazu wird ein weiteres Schlüsselwort (*always*) verwendet, welches dann den Wert *true* liefert, wenn die entsprechende Bedingung für jeden Zeitpunkt des Kontextintervalls erfüllt ist. Bei *automatic\_ok* handelt es sich um eine Assessment-Variable. Dies ist ein Array, welches bei jeder neuen Zuweisung ein Element speichert, das den zugewiesenen Wert und Start- und Endzeit des aktuellen Kontextintervalls enthält (Beispiel für einen Arrayelement: *{Startzeit: 2,2 s, Endzeit: 2,4 s, Wert: false}*). Durch das Auswerten von Assessment-Variablen

Zeitintervalle	
float iv.getStartTime()	Startzeit eines Intervalls
float iv.getEndTime()	Endzeit eines Intervalls
float iv.getLength()	Dauer eines Intervalls
bool iv.intersects(interval otheriv)	Prüft, ob sich zwei Zeitintervalle überschneiden
Assessmentvariablen	
int av.getSize()	Liefert Anzahl der gespeicherten Feldelemente (Zeitintervall und assoziierte Werte)
interval av[int index]	Zugriff auf ein Feldelement (Zeitintervall und assoziierte Werte)
TPT-Kontrollfluss	
bool TPT.always(t-bool expr)	Auswertung eines Ausdrucks über alle Werte eines Zeitintervalls; wahr, wenn Ausdruck stets zutreffend
bool TPT.never(t-bool expr)	Auswertung eines Ausdrucks über alle Werte eines Zeitintervalls; wahr, wenn Ausdruck stets nicht zutreffend
Signalverarbeitung (Wertberechnungen)	
double TPT.average(t-float expr)	Berechnet den Durchschnitt aller Werte im aktuellen Kontextintervall
float TPT.max(t-float expr)	Liefert das Maximum aller Werte im aktuellen Kontextintervall
float TPT.min(t-float expr)	Liefert das Minimum aller Werte im aktuellen Kontextintervall
Signalverarbeitung (Signalberechnungen)	
t-float TPT.deviation (t-float expr1, t-float expr2, float x)	Liefert die Abweichung zweier Signale abzüglich der Toleranz x (oder Null, wenn Abweichung kleiner als die Toleranz)
t-float TPT.integrate (t-float expr, int type)	Integral eines Signal (verfügbare Typen: EULER_FORWARD, EULER_BACHWARD, TUSTIN)
t-float TPT.dt (t-float expr)	Liefert den Differentialquotienten
t-bool TPT.monotony (t-float expr, int type)	Prüft das monotone Verhalten eines Signals (verfügbare Typen: INCREASING, DECREASING, STRICTLY_INCREASING, STRICTLY_DECREASING)

**Tabelle 2. Auswahl an Vergleichsmethoden und Kontrollflussroutinen**

kann der Zeitpunkt eines falschen Verhaltens bestimmt und ein Fehler schnell eingegrenzt werden.

Andere verfügbare Erweiterungen umfassen Methoden zur Verarbeitung von Zeitintervallen und Assessment-Variablen, zur Signalverarbeitung, umfangreiche Vergleichsmethoden sowie spezielle Kontrollflussrou-

tin. Eine Auswahl ist in **Tabelle 2** aufgelistet.

Nach Erstellen der Testlets, Szenarien und Assessment-Skripte werden die Tests ausgeführt (wahlweise im Debug- oder im Normal-Modus). Nach Abschluss der Testausführung generiert TPT Logfiles oder HTML-Reports. sj



**Dipl.-Ing. Menno Mennenga**

studierte Elektrotechnik an der TU Dresden und an der Texas A&M University. Er ist bei der Mindteck Germany GmbH für die Geschäftsentwicklung in Deutschland zuständig.  
[menno.mennenga@mindteck.com](mailto:menno.mennenga@mindteck.com)



**Dipl.-Ing. Christian Dziobek**

studierte Allgemeine Elektrotechnik an der RWTH Aachen. Seit 1998 arbeitet er bei der Daimler AG und beschäftigt sich mit der Einführung von Methoden und Tools zur modellbasierten Funktionsentwicklung im Bereich E/E der Pkw-Serienentwicklung.  
[christian.dziobek@daimler.com](mailto:christian.dziobek@daimler.com)



**Dipl.-Ing. Iyad Bahous**

studierte an der Cologne University of Applied Sciences Fahrzeugtechnik mit den Schwerpunkten Fahrwerk und Simulation. Er ist Leiter Software Verification & Diagnostics Management bei der MB-technology GmbH in Sindelfingen.  
[iyad.bahous@mbtech-group.com](mailto:iyad.bahous@mbtech-group.com)