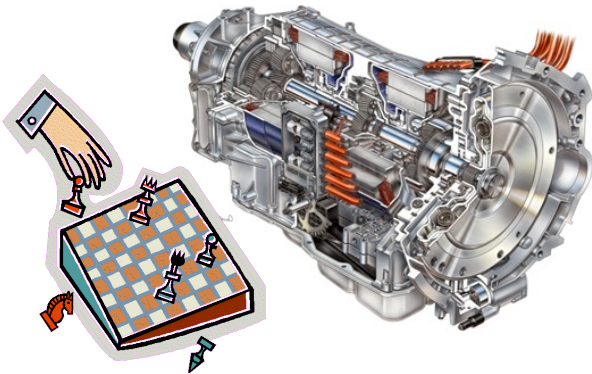




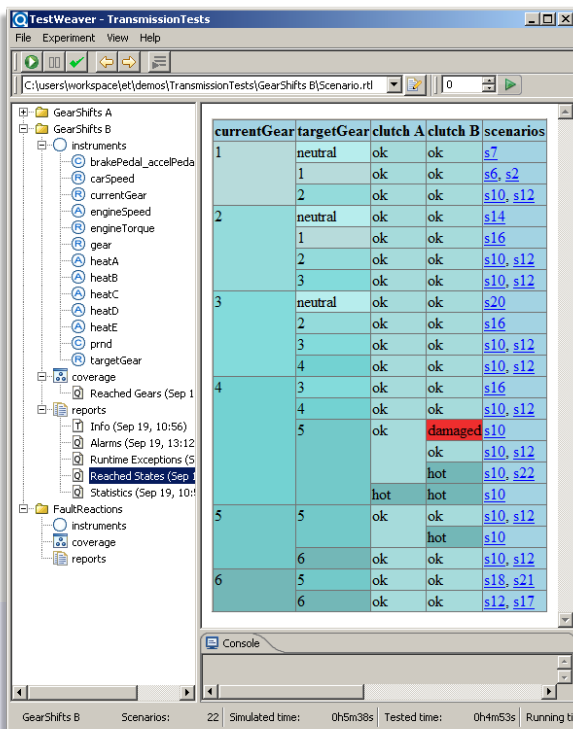
# TestWeaver 2.3

## Hohe Testabdeckung mit wenig Aufwand



TestWeaver ist Software für den automatisierten Test eingebetteter Systeme auf MiL, SiL und HiL Basis.

**TestWeaver benötigt zum Durchführen eines Tests keine Testskripte, sondern generiert selbstständig tausende von Testfällen, führt sie per Simulation aus und bewertet die Systemantwort.** Testfälle werden dabei mittels eines einmaligen Analyseverfahrens intelligent und automatisch erzeugt.



### Feature

- Automatisches Erzeugen und Optimieren von Testfällen  
Ergänzend: skriptbasierte Testautomatisierung
  - interaktives Aufnehmen und Abspielen von Testfällen
  - manuelle Skripterstellung, z.B. mit Python
- Reaktive Testausführung mit MiL, SiL oder HiL
- Automatische Auswertung und Berichterstellung
- Unterstützung von positiven und negativen Tests
- Test mit Simulation von Bauteilfehlern
- Worst-case Analyse
- Durchgängige Verwendung der Tests in MiL, SiL, HiL
- 100% Reproduzierbarkeit von Tests in MiL und SiL
- Bericht über erreichte Testabdeckung im Zustandsraum
- Bericht über Code Coverage, z.B. mit CTC++
- Debugging für C/C++ mit MS Visual Studio

Unterstützung von Automobilanwendungen via Silver:

- ASAP2/A2L, CAN, MDF, XCP, FMI, ISO 26262
- Flashen von Steuergerät Parametern in die Simulation
- Range Überwachung für alle Steuergerät Signale
- Stack- und Laufzeit Überwachung für Funktionscode

Probleme, die TestWeaver entdeckt und meldet:

- Division durch Null, Access Violation, Endlosschleifen, nicht-Determinismus, Range Violation von Signalen
- Oszillation von diskreten and kontinuierlichen Signalen
- Algorithmische Fehler, z.B. Zustandschätzung signifikant falsch für mehrere Zyklen
- Probleme aus Systemebene: schlechte Funktionsqualität, Überhitzung von Bauteilen, falsche Fehlerreaktion

Kodierungsfehler können oft mit überraschend einfachen Simulationsmodellen gefunden werden, während der Test der Funktionsqualität in der Regel gut kalibrierte Simulationsmodelle erfordert.

Unterstützte Modellierungs- und Entwicklungswerkzeuge:

- C/C++, zum Beispiel Microsoft Visual Studio
- MATLAB/Simulink, Real Time Workshop, Dymola
- via Silver: AMESim, SimulationX, Python
- Testwell CTC++ zur Messung der Codeabdeckung

TestWeaver läuft unter Windows auf Standard PCs.

### Nutzen

- schneller entwickeln: durch frühe Fehlererkennung
- höhere Testabdeckung: tausende von Testfällen
- weniger Aufwand: mehr Automatisierung, keine Skripte

**QTronic GmbH**  
Alt-Moabit 92  
D-10559 Berlin

info@QTronic.de  
www.QTronic.de  
+49 30 3512 1067

## Einsatz von TestWeaver in vier Schritten

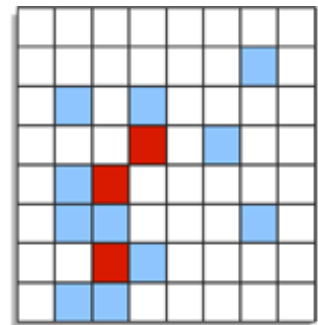
1

Startpunkt ist ein ausführbares Modell des zu testenden Systems. Das Modell enthält typischerweise die Steuerungssoftware des Systems und ein Streckenmodell. Solche Modelle werden im modellbasierten Entwicklungsprozess sowieso erstellt. Für die Entwicklung des Systemmodells können fast beliebige Entwicklungswerkzeuge eingesetzt werden, siehe Rückseite  
TestWeaver benötigt keinen Zugriff auf den Quellcode des Modells, denn TestWeaver testet kompilierte Modelle.

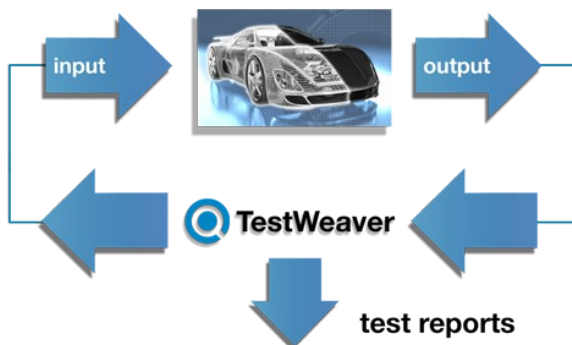


2

Wähle die Schlüssel Inputs und Outputs des Modells. Typische Inputs sind Gas- und Bremspedal, Lenkwinkel, Strassen-eigenschaften und Variablen zur Steuerung von Bauteilfehlern, z. B. Sensorfehlern. Typische Outputs sind Zustandsvariablen der Steuerungssoftware und wichtige Variablen des Fahrzeugmodells, z. B. Ist- und Zielgang, Motormoment. Klassifiziere die möglichen Werte aller Outputs, um TestWeaver die Unterscheidung zwischen 'gutem' und 'schlechtem' Verhalten des getesteten System zu ermöglichen. Klassifiziere ebenso die Werte aller Inputs, um nominale Inputs von solchen zu unterscheiden, die Bauteilfehler steuern. Für kontinuierliche Inputs oder Outputs ist Klassifikation eine Zerlegung der Zahlenachse in Intervalle und die Zuweisung der obigen Eigenschaften (z.B. 'gut', 'schlecht') an jedes solche Intervall.



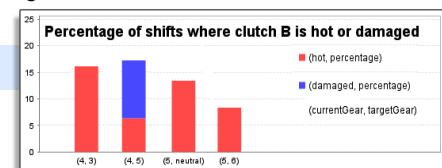
3



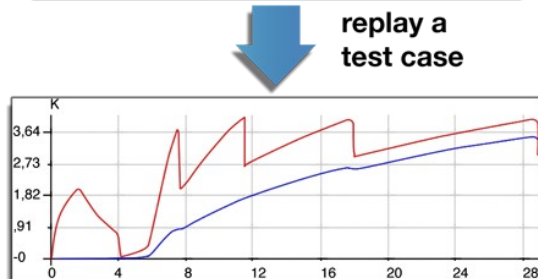
Verbinde die Inputs und Outputs des Systemmodells mit TestWeaver. Dazu gibt es je nach Modell mehrere Möglichkeiten, z. B. ein Simulink Blockset, Pythonskripte, Modelica, Silver oder eine C library.

Starte TestWeaver. TestWeaver treibt das Systemmodell dann selbstständig über dessen Inputs und beobachtet die resultierende Systemantwort an den Outputs. TestWeaver versucht dabei aktiv, das System in unerwünschte Zustände zu treiben und die Menge der dabei eingenommenen Systemzustände zu maximieren, also jeden diskreten Zustand möglichst einmal zu erreichen. Der Zustandsraum wird dabei von den Inputs und Outputs aufgespannt, die in Schritt 2 ausgewählt und diskretisiert worden sind.

alarms	currentGear	targetGear	scenarios
Array index overflow	4	neutral	s13, s9
Access violation	5	neutral	s21, s37
Division by 0	3	neutral	s53, s75
clutchB overheating	4	5	s4, s8
targetGear oscillations	2	3	s62



4



Analysiere die von TestWeaver gefundenen Probleme. TestWeaver berichtet Probleme und die erreichte Testabdeckung tabellarisch und über Histogramme. Jedes gefundene Problem kann in der jeweiligen Simulations- oder Entwicklungsumgebung reproduziert werden, um es zu analysieren oder zu debuggen. Interessante Testfälle können in Testdatenbanken gesammelt werden, z. B. für Regressionstests. Tests können auch für andere Werkzeuge exportiert werden, z. B. für den HiL Test.

### Unsere Dienstleistungen

- Beratung zu Test und Validierung von Systemen
- Unterstützung beim Entwickeln von Systemmodellen
- Einsatz von TestWeaver in Entwicklungsprojekten