Verifikation im Bereich Funktionale Sicherheit

Gudrun Neumann, 07.07.2016

SGS TÜV SAAR

WHEN YOU NEED TO BE SURE

SGS

AGENDA

- Grundlagen der Verifikation
- Verifikation im Softwarelebenszyklus
- Statische Test bzw. Analyse Methoden
- Dynamische Test Methoden

AGENDA

- Grundlagen der Verifikation
- Verifikation im Softwarelebenszyklus
- Statische Test bzw. Analyse Methoden
- Dynamische Test Methoden

MOTIVATION

- Die Verifikation von Software
  - trägt durch Identifizierung und Beseitigung von Fehlern zur Steigerung der Softwarequalität bei.
  - verringert die Risiken beim Einsatz der Software, da mögliche Fehler während des Testens aufgedeckt werden.
- Die Standards zur Funktionalen Sicherheit empfehlen ausgewählte Techniken und Methoden zur Verifikation von Software.
- Im folgenden werden Methoden entsprechend IEC 61508:2010 und ISO 26262:2011 vorgestellt.

DURCHFÜHRUNG

Die Verifikation erfordert unabhängig von der Methodenauswahl folgende Schritte:

- Planung der Ressourcen und der Strategie
- Festlegung der zu verwendenden Verifikationsverfahren , Tools und Testumgebung
- Im Falle von Verifikation durch Testen:
  - Auswahl geeigneter Testfälle mit Festlegung von PASS / FAIL Kriterien
- Durchführung der Verifikation
- Dokumentation der Verifikationsergebnisse, inklusive Abweichungen

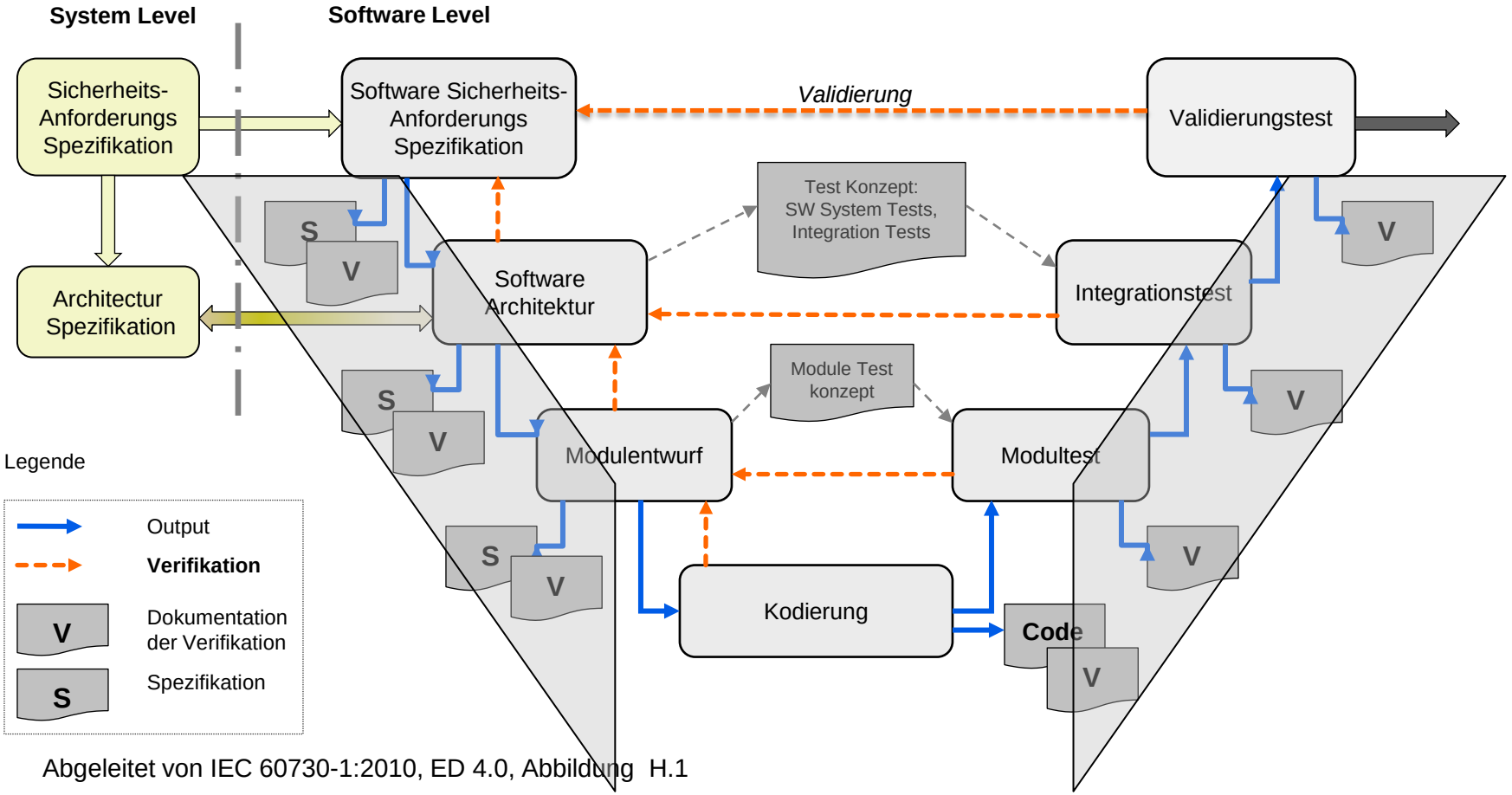
ALLGEMEIN

- Zur Verifikation können folgende Methoden verwendet werden:
  - Testen
  - Analysieren
  - Simulieren
- Für eine Software-Validierung wird als Methode Testen empfohlen

AGENDA

- Grundlagen der Verifikation
- Verifikation im Softwarelebenszyklus
- Statische Test bzw. Analyse Methoden
- Dynamische Test Methoden

BEISPIEL EINES V-MODEL



Abgeleitet von IEC 60730-1:2010, ED 4.0, Abbildung H.1

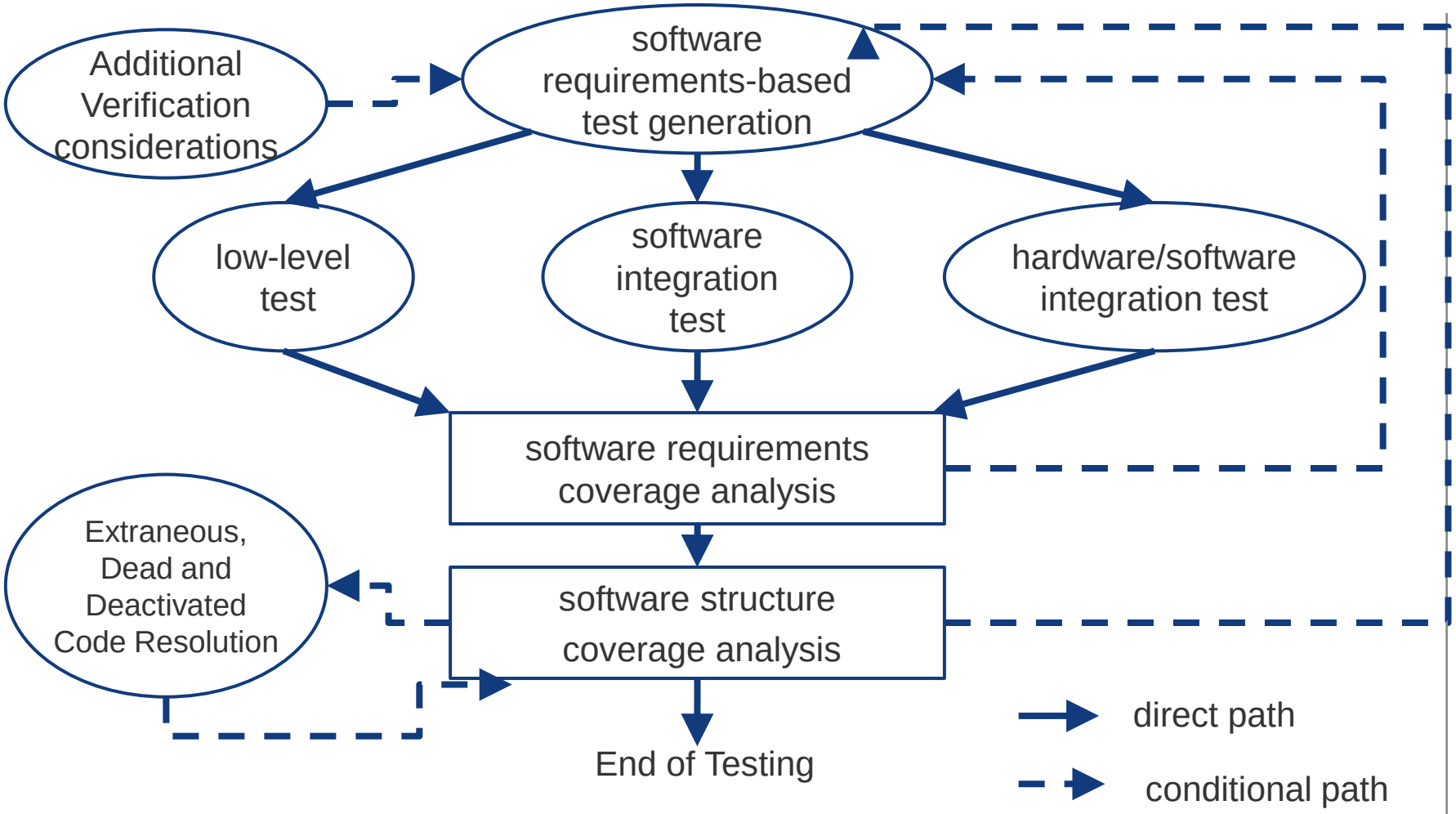
FESTLEGEN DER EINZELNEN INTEGRATIONSSCHRITTE

- Integration in einem oder mehreren Schritten
- Die verschiedenen Ebenen, wie Modultest, Integrationstest und Validierungstest, müssen erkennbar sein
- „Big Bang“ Strategie kann unübersichtlich sein
- Schrittweises Vorgehen ist zu empfehlen

Schrittweises Vorgehen

Big Bang

SOFTWARE TESTING PROCESS (DO-178-C)



AGENDA

- Grundlagen der Verifikation
- Verifikation im Softwarelebenszyklus
- Statische Test bzw. Analyse Methoden
- Dynamische Test Methoden

STATISCHER TEST / ANALYSE

- Das Ziel der statischen Analyse ist die Aufdeckung vorhandener Fehler in einem Dokument / Source Code.
- Der zu prüfende Source Code wird nicht ausgeführt.
- Alle statischen Analysen können prinzipiell ohne Werkzeugunterstützung durchgeführt werden.
- Eine Werkzeugunterstützung ist sinnvoll, Ausnahmen sind Inspektions- und Review-Techniken.
- Statische Tests sind White Box Tests, d.h. der Source Code muss bekannt sein.

STATISCHE PRÜFUNGEN

- Methoden:
  - Strukturierte Gruppenprüfungen (Reviews)
  - Semantische Code Analyse (Semantic code analysis)
  - Grenzwertanalyse (Boundary value analysis)
  - Fehlererwartung (Error guessing)
  - Kontrollflussanalyse (Control flow analysis)
  - Datenflussanalyse (Data flow analysis)
  - Checklisten (Checklists)
  - Statische Code Analyse (Static code analysis)
  - Symbolische Ausführung (Symbolic execution)
  - Formale Verifikation (Formal verification)
  - Semi-formale Verifikation (Semi-formal verification)

SEMANTISCHE CODE ANALYSE

- Ziele:
  - Aufdecken von Fehlern und fehlerträchtigen Codeteilen in der SW
- Überprüfungsmethode:
  - manuelle Prüfung durch Code-Review (inhaltliche Bewertung kann ein Tool nur teilweise leisten).
- Ausführungszeitpunkt:
  - noch bevor die entsprechende Software (z. B. im Modultest) ausgeführt wird.

Methode, ISO 26262-6, Table 9	ASIL A	ASIL B	ASIL C	ASIL D
Semantic code analysis	+	+	+	+

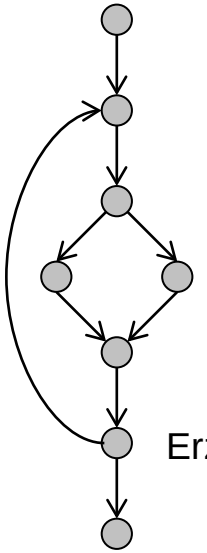
KONTROLLFLUSSANALYSE (CONTROL FLOW ANALYSIS)

- Ziel:
  - Erkennung schlechter und möglicherweise inkorrekter Programmstrukturen.
- Die Kontrollflussanalyse ist ein statisches Testverfahren zur Aufdeckung verdächtiger Codebereiche, die schlechter Programmierpraxis folgen.
- Das Programm wird analysiert, um einen gerichteten Graphen zu erhalten, der weiter analysiert werden kann nach:
  - unzugänglichem Code, zum Beispiel bedingungslose Sprünge, die Blöcke von nicht erreichbarem Code hinterlassen;
  - verknotetem Code. Gut strukturierter Code hat einen Steuergraphen, der schrittweise auf einen einzelnen Knoten reduzierbar ist. Im Gegensatz dazu kann ein schlecht strukturierter Code nur auf eine Struktur reduziert werden, die aus mehreren Knoten zusammengesetzt ist. Siehe Beispiel auf der nächsten Seite.

Measure, IEC 61508-3, Table B.8	SIL1	SIL 2	SIL 3	SIL 4
Control flow analysis	R	HR	HR	HR

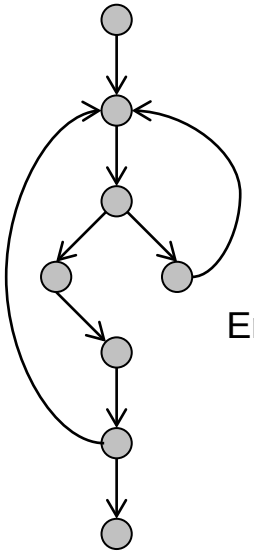
KONTROLLFLUSSANALYSE, BEISPIEL

Gut-strukturierter Code



Erzeugt z.B. durch
do … while

Schlecht-strukturierter Code



Erzeugt z.B. durch
continue,
break oder
goto

Legende:

- Kreis = Knoten = Anweisung
- → Pfeil = Kante = Pfad = Kontrollfluss

DATENFLUSSANALYSE

- Ziel:
  - Erkennung schlechter und möglicherweise inkorrekter Programmstrukturen.
- Die Datenflussanalyse ist ein statisches Testverfahren, das die durch die Kontrollflussanalyse gewonnenen Erkenntnisse/ Ergebnisse nutz.
- Die Analyse kann z.B. entdecken:
  - Variablen, die gelesen werden können, bevor ihnen ein Wert zugewiesen wird.
  - Variablen, die mehrfach beschrieben werden, ohne gelesen zu werden. Dies könnte auf übersprungenen Code hindeuten;
  - Variablen, die beschrieben werden, aber nie gelesen werden. Dies könnte auf redundanten Code hindeuten.

DATENFLUSSANALYSE, BEISPIEL

```
void  MinMax  (int &min, int &max)
{
  int hilf;
  if (min > max)
  {
    max = hilf;
    max = min;
    hilf = min;
  }
}
```

Fehler: Variable „hilf“ ist nicht initialisiert.

Warnung: Variable „max“ wurde mehrmals beschrieben aber dazwischen nicht gelesen.

Warnung: Variable „hilf“ wurde beschrieben aber wird nicht mehr gelesen.

CHECKLISTEN (CHECKLISTS)

- Ziel:
  - Zusammenstellung von Fragen, mit denen versucht wird, alle Problemfelder des IST-Zustandes zu behandeln und systematisch Schwachstellen zu finden.
- Bei der Zusammenstellung der Fragen spielen die logische Betrachtung des Prüfobjektes als auch die Erfahrungen aus der Praxis eine relevante Rolle.
- Beispiele:
  - Wird das firmenspezifische Template benutzt?
  - Wurde das Änderungsverzeichnis aktualisiert?
  - Gibt es veraltete / falsche Kommentare im Source Code?

Measure, IEC 61508-3, Table B.8	SIL1	SIL 2	SIL 3	SIL 4
Checklists	R	R	R	R

STATISCHE CODE ANALYSE

- Ziel:
  - Formale Überprüfung des Quelltextes um bestimmte Arten von Fehlern aufzudecken, z.B. gemäß MISRA-C Regeln.
- Durchführung sinnvoll, noch bevor die entsprechende Software (z. B. im Modultest) ausgeführt wird.
- Überprüfung erfolgt
  - manuell per Review
  - automatisch durch ein Programm
  - Einfache Analysen sind häufig bereits im Compiler (Übersetzer) einer Programmiersprache integriert, z. B. die Prüfung auf Initialisierung einer Variablen.

Methode, ISO 26262-6, Table 9	ASIL A	ASIL B	ASIL C	ASIL D
Static code analysis	+	++	++	++

SYMBOLISCHE AUSFÜHRUNG

- Ziel:
  - Darlegung der Übereinstimmung zwischen dem Source-Code und der Spezifikation.
- Die symbolische Ausführung entspricht einem Pfadüberdeckungstest mit symbolischen Werten.
- Einschränkung:
  - bei größeren Programmen kann die Anzahl der Programmpfade explodieren und damit der symbolische Test zu komplex werden.
- Anwendungsfall:
  - Ein Testdatengenerator kann aus der symbolischen Analyse eines Programmes Testfälle für den dynamischen Test erzeugen.

Measure, IEC 61508-3, Table B.8	SIL1	SIL 2	SIL 3	SIL 4
Symbolic execution	--	--	R	R

SYMBOLISCHE AUSFÜHRUNG, BEISPIEL

- Fragestellung: Wann wird fail() ausgeführt, d.h. fail() == true?

```
void  Checkinput  (void)
{
  y = read();
  y = 2 * y;
  if (y == 12)
    fail();
  else
    print(„OK“);
}
```

y	Bedingung
S	
2*S	
2*S	2*S==12
2*S	2*S <> 12

Ergebnis folgende symbolische Ausdrücke:

fail()=true ∧ S=6	**fail()=false ∧ S<>6**

SEMI-FORMALE VERIFIKATION

- Ziel:
  - Kombination der Vorteile von formalen Methoden mit nicht-formalen Methoden.
- Die semi-formale Verifikation ist eine Verifikation, die auf einer Beschreibung in semi-formaler Notation basiert.
- Beispiel:
  - Generieren von Testdaten basierend auf einem semi-formalen Modell, um so das Systemverhalten gegenüber der Spezifikation zu verifizieren.

Methode, ISO 26262-6, Table 9	ASIL A	ASIL B	ASIL C	ASIL D
Semi-formal verification	+	+	++	++

AGENDA

- Grundlagen der Verifikation
- Verifikation im Softwarelebenszyklus
- Statische Test bzw. Analyse Methoden
- Dynamische Test Methoden

DYNAMISCHER TEST

- Die ausführbare SW wird mit konkreten Eingabewerten versehen und ausgeführt (Testfall).
- Die vom Testlauf erzeugten Ausgabedaten werden mit den erwarteten Daten verglichen. Bei Abweichung liegt ein Fehler vor.
- Eine wesentliche Aufgabe ist die Festlegung geeigneter Testfälle für den Test der Software.
- Es kann in der realen Betriebsumgebung getestet werden.
- Dynamische Tests sind Stichprobenverfahren, d.h. Restfehler können nicht ausgeschlossen werden.
- Dynamische Tests können Black-Box Tests sein.

DYNAMISCHER TESTMETHODEN

- Methoden:
  - Grenzwertanalyse (Boundary Value Analysis)
  - Fehlererwartung (Error Guessing)
  - Fehlereinpflanzung (Error seeding / Fault Injection)
  - Modelbasiertes Testen, inkl. automatische Testfallgenerierung (model-based test case generation)
  - Back-to-back Test
  - Äquivalenzklassen und Test von Partitionen des Eingangsbereichs (Equivalence classes and input partition testing)
  - Ressourcen Gebrauch Test (Resource usage test)
  - Modellierung der Leistungsfähigkeit (Performance modeling)
  - Anforderungsbasiertes Testen (Requirements-based Test)
  - Schnittstellenprüfung (Interface Test)
  - Strukturelle Überdeckung (Structural test coverage)

FEHLERERWARTUNG (ERROR GUESSING)

- Ziel:
  - Testfallerzeugung unter Anwendung schon vorliegender Erfahrungen.
- Diese Methode ist keine unmittelbare Ableitung aus den Anforderungen.
- Beruht auf den Erfahrungen der Testingenieure, also Expertenwissen.
- Testfälle können hier auch durch die Verwendung der „lessons learned“ Dokumentationen oder von Kundenbeschwerden abgeleitet werden.
- Diese Methode empfiehlt sich, wenn schon Erfahrung mit ähnlichen Testobjekten vorliegt.

Measure, IEC 61508-3, Table B.8	SIL1	SIL 2	SIL 3	SIL 4
Error guessing	R	R	R	R

FEHLEREINPFLANZUNG (ERROR SEEDING)

- Ziel:
  - Feststellen, ob eine Auswahl von Testfällen angemessen ist.
- Einige bekannte Arten von Fehlern werden in das Programm eingebracht (gesetzt). Das Programm wird mit den Testfällen unter Testbedingungen ausgeführt. Wenn nur einige der eingebrachten Fehler aufgedeckt werden, ist die Auswahl der Testfälle nicht angemessen.
- Diese Testmethode eignet sich, um Testsysteme zu testen.

Measure, IEC 61508-3, Table B.2	SIL1	SIL 2	SIL 3	SIL 4
Test case execution from error seeding	--	R	R	R

FEHLEREINPFLANZUNG (FAULT INJECTION)

- Ziel:
  - Nachweis von Fehlerbeherrschungsmaßnahmen.
- Testen von Code, der sonst nur schwer testbar / erreichbar ist (z.B. Fehler Handling Code).
- Einbringen von Fehlern in das Testobjekt. Dies geschieht unter Verwendung von speziellen Schnittstellen (Software) oder / und spezifisch präparierter Hardware.
- Auswertung der Reaktion des Testobjektes unter Berücksichtigung der geforderten Antwortzeiten.

Methode, ISO 26262-6, Table 10	ASIL A	ASIL B	ASIL C	ASIL D
fault injection test	+	+	+	++

Methode, ISO 26262-6, Table 13	ASIL A	ASIL B	ASIL C	ASIL D
fault injection test	+	+	++	++

Strukturelle Überdeckung (Structural test coverage)

- Ziel:
  - Nachweis, dass die Anzahl der Testfälle ausreichend ist und es keine unbeabsichtigte Funktionalität gibt.
- Struktuelle Überdeckung beschreibt zu welchem Prozentsatz alle Zweige bzw. Abläufe des Source Codes einer Software-Unit / -Komponente getestet wurden.

Measure, IEC 61508-3, Table B.2	SIL1	SIL 2	SIL 3	SIL 4
Structural test coverage (entry points) 100%	HR	HR	HR	HR
Structural test coverage (statements) 100%	R	HR	HR	HR
Structural test coverage (branches) 100%	R	R	HR	HR
Structural test coverage (conditions, MC/DC) 100%	R	R	R	HR

ZUM SCHLUSS

- Ziel des Testfalls beachten
- PASS/FAIL Kriterien dokumentieren
- Geeignete qualifizierte Tool verwenden
- Testergebnisse archivieren
- Nachvollziehbarkeit sicherstellen
  (Anforderung –
  Spezifikation –
  Implementierung –
  Verifikation –
  Validierung)

Danke für Ihre Aufmerksamkeit

YOUR PARTNER FOR FUNCTIONAL SAFETY

SGS-TÜV Saar GmbH

Functional Safety

Hofmannstrasse 50

D-81379 Muenchen

Germany

www.sgs-tuev-saar.com/fs

Gudrun Neumann

Product Manager Functional Safety Software

Industrial Functional Safety Expert

Automotive Functional Safety Expert

E-mail: gudrun.neumann@sgs.com

Telephone: +49 89 787 475 -216

Fax: +49 89 787 475 -217