

Modularer HiL-Teststand

Überblick und Test-Konzepte

Johannes Bergsmann

Berater, Trainer, Eigentümer



CONSULTING

- Management Consulting
- Prozesse und Vorgehensmodelle
- Teststrategie und -konzeption
- Requirements und Ausschreibungen
- Architektur und Modellierung
- Code Analyse und Metriken



OPERATIONAL SERVICES & TESTCENTER

- TestCenter
- Requirements Engineering
- Testmanagement und -spezifikation
- Softwareverifikation und -validierung
- Testautomatisierung und -reporting
- Reviews und Code Analyse



ACADEMY

- Requirements und Usability
- Architektur und Modellierung
- Testen und Automatisieren
- Agile Softwareentwicklung
- Projektabwicklung und Vorgehensmodelle
- Lehrgänge und Zertifizierungen



TOOL EXPERTISE

- Tool Evaluation Center
- Tool-Einführung und Pilotprojekte
- Tool-Schnittstellen und -Einbindung
- Softwareprozessautomatisierung
- Test Automation Frameworks
- Tool-Studien und Tool-Expertisen



**software
uality lab**

Innovation Meets Quality

Über Software Quality Lab


Auszug aus der Kundenliste

Energie & Versorgung	Elektronik & Automation	Industrie & Engineering	Dienstleistungen & Handel
       	      	              	    
Medizin & Pharma         	Software & IT          	Finanz & Versicherung          	Mobilität & Kommunikation      

© Software Quality Lab

www.software-quality-lab.com

- 3 -




**software
uality lab**

Innovation Meets Quality

Modularer HIL-Teststand

Inhalt



- Ausgangsüberlegungen
- Features
- Grundkonzept und Grobarchitektur
- Moderne Testtechniken

© Software Quality Lab

www.software-quality-lab.com

- 4 -

Ausgangsüberlegungen

Status Quo bei HIL-Testständen

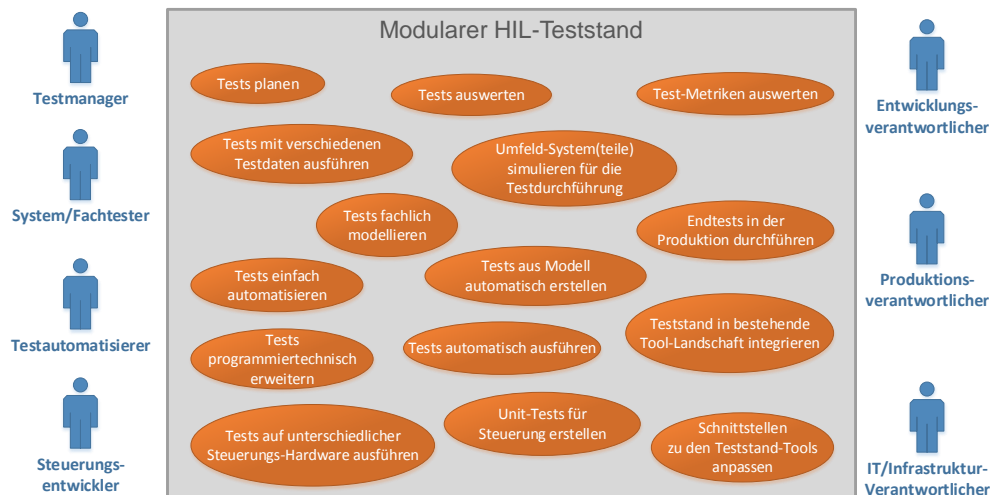


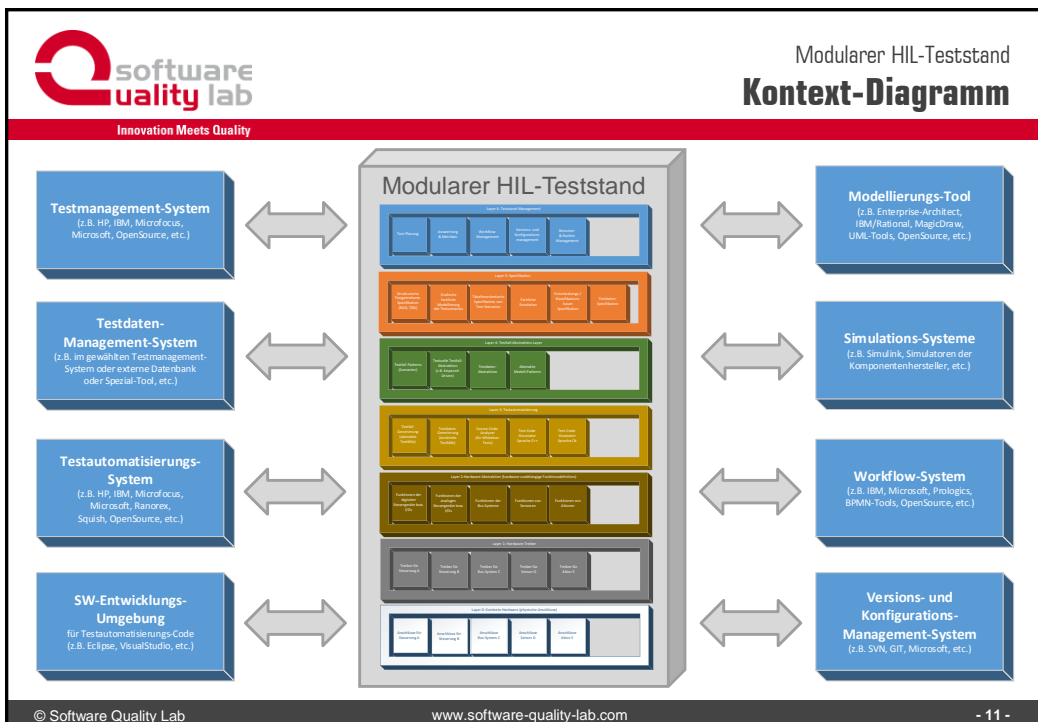
- Viele Firmen in der Industrie benötigen HIL-Teststände
 - für die Entwicklung und
 - für die Fertigungs-Prüfung
- Oft mit viel Aufwand erstellt
 - Entwicklungskosten oft deutlich > 200T. EUR
 - Wartung und Weiterentwicklung zusätzlich
 - Mehrere Entwickler werden tw. für Teststand-entwicklung und Wartung gebunden, statt am eigentlichen Produkt zu entwickeln!
- Tests (Schnittstellen, Testfall-Muster, etc.) sind immer wieder ähnlich / vergleichbar
- Testmethoden oft nicht State of the Art

Features & Architektur im Überblick

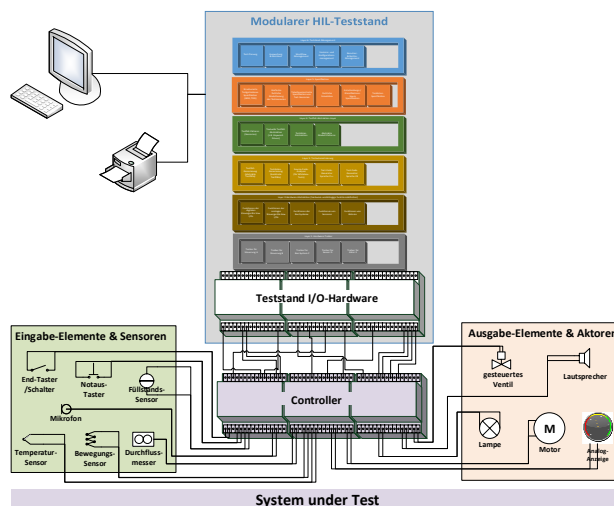
aus der Sicht von Testspezialisten

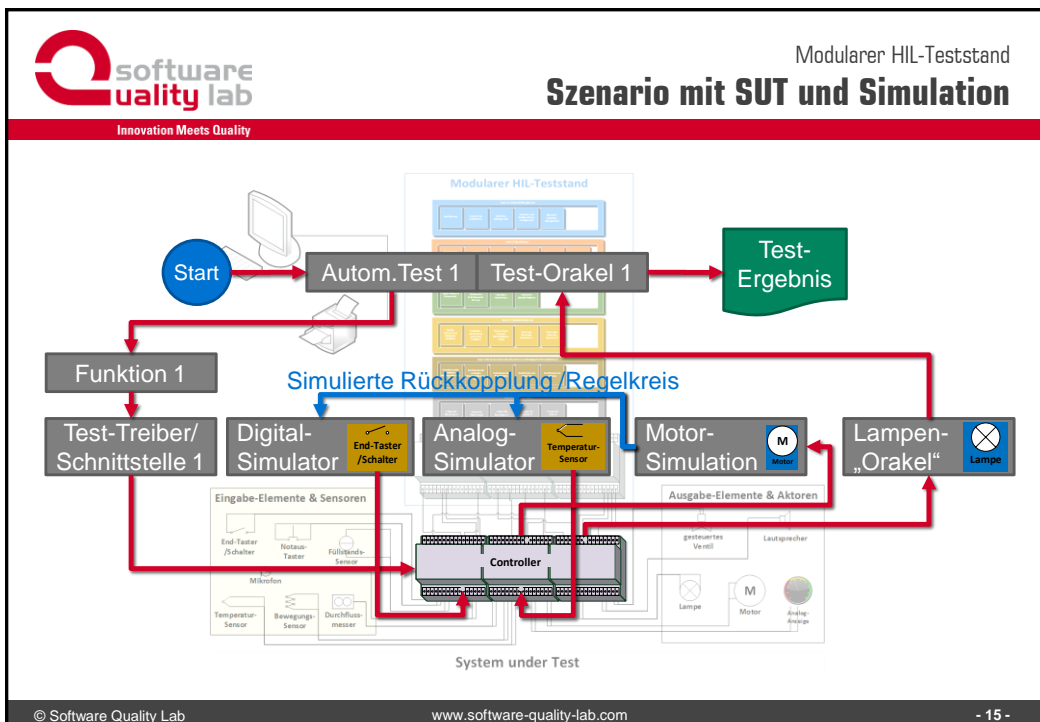
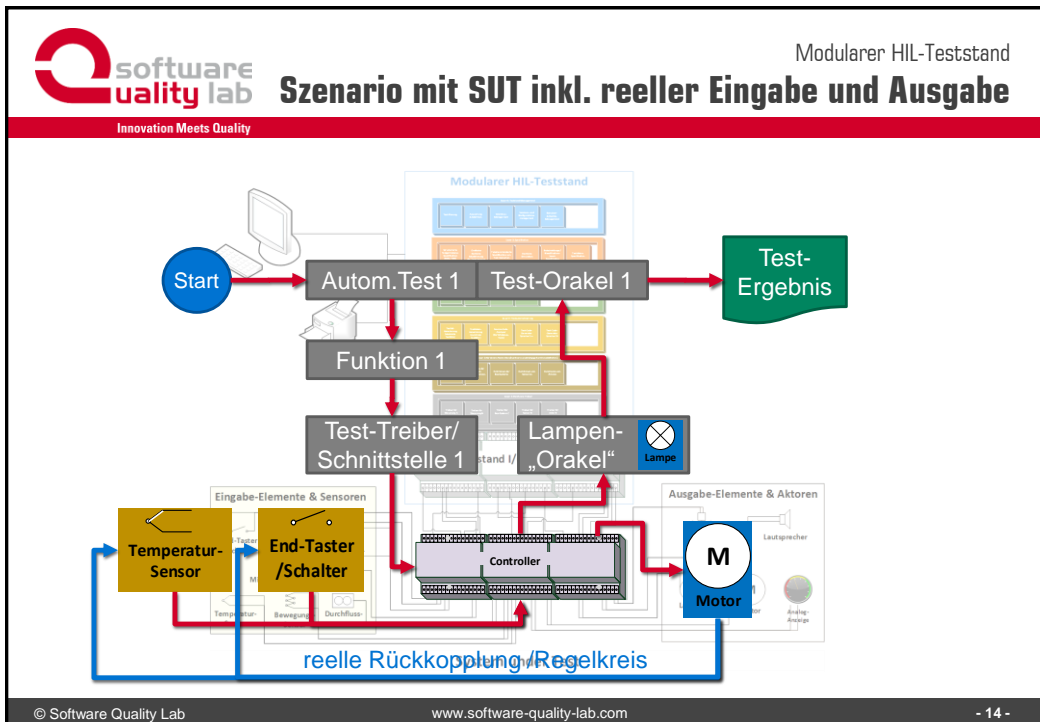
Modularer HiL-Teststand Use-Cases Überblick





- **Unterschiedliche Hardware-Bausteine** zur Automatisierung andockbar (digitale I/O, analoge I/O, Ethernet, CAN-Bus, RS232, RS485, etc.)
- **“Einfache Bedienbarkeit”** auch für “Nicht Soft-Werker” geeignet.
- Soll **auch als Produktionsteststand** konfigurierbar und einsetzbar sein.
- **Reuse von Testfällen und Testszenarien** soll ermöglicht werden (Testfall-Library, Patterns)
- **Automatische Generierung von Testfällen** aus Spezifikationen (z.B. Modellen, Sprach-Patterns, etc.)



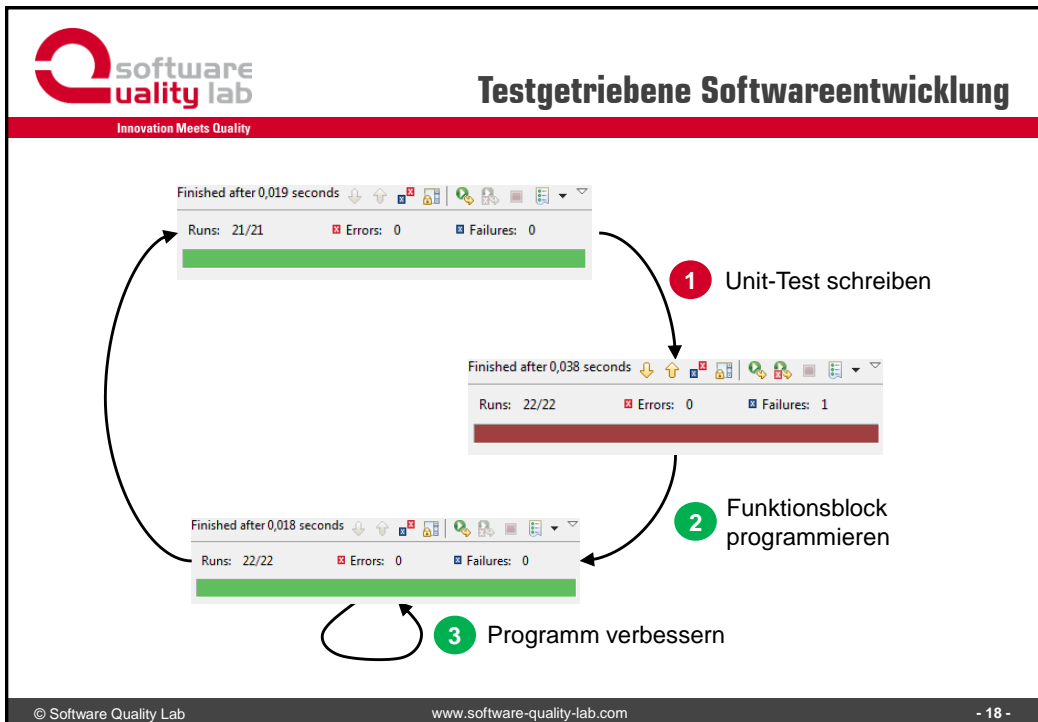


Moderne Testtechniken

für HIL-Test

Testgetriebene Vorgehensweise

Wenn Sie agil entwickeln wollen!



software quality lab
Innovation Meets Quality

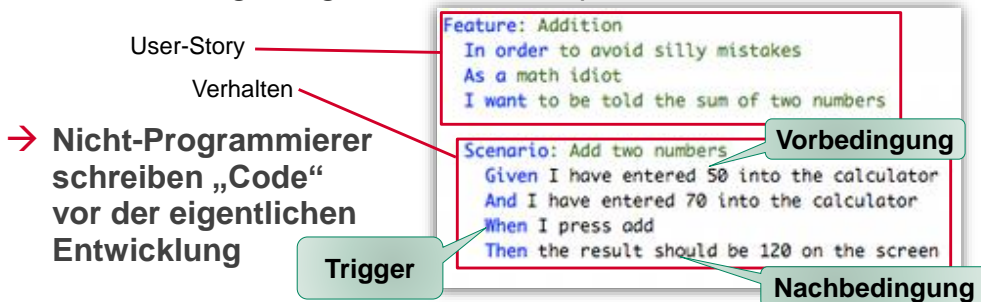
Behaviour Driven Development

Testautomatisierung für Fachtester

© Software Quality Lab www.software-quality-lab.com - 19 -

Behaviour Driven Development (BDD)

- 2003 von Dan North als Antwort auf Test-Driven Development (Unit-Tests sind für Anwender schlecht lesbar)
- erstellt User-Stories und erfasst dazu auch das Verhalten (Behaviour) mit Vor-&Nachbedingungen
- Beschreibung erfolgt in „natürlicher Sprache“:



Behaviour Driven Development (BDD)

Beispiel Cucumber (Tool für BDD, Codebasis: Ruby)

1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

Scenario: Add two numbers
  Given I have entered 50 into the calculator
  And I have entered 70 into the calculator
  When I press add
  Then the result should be 120 on the screen
```

2: Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

3: Run and watch it fail (Class Calculator missing)

```
$ cucumber features/addition.feature
Feature: Addition # Features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # Features/addition.feature
    Given I have entered 50 into the calculator # Features/step_1
    And I have entered 70 into the calculator # Features/step_2
    When I press add # Features/step_3
    Then the result should be 120 on the screen # Features/step_4
```

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition # Features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # Features/addition.feature
    Given I have entered 50 into the calculator # Features/step_1
    And I have entered 70 into the calculator # Features/step_2
    When I press add # Features/step_3
    Then the result should be 120 on the screen # Features/step_4
```

6. Repeat 2-5 until green like a cucumber

```
$ cucumber features/addition.feature
Feature: Addition # Features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers # Features/addition.feature
    Given I have entered 50 into the calculator # Features/step_1
    And I have entered 70 into the calculator # Features/step_2
    When I press add # Features/step_3
    Then the result should be 120 on the screen # Features/step_4
```

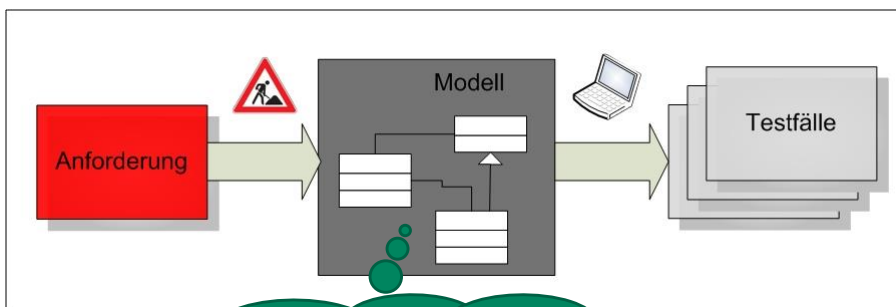
7. Repeat 1-6 until the money runs out



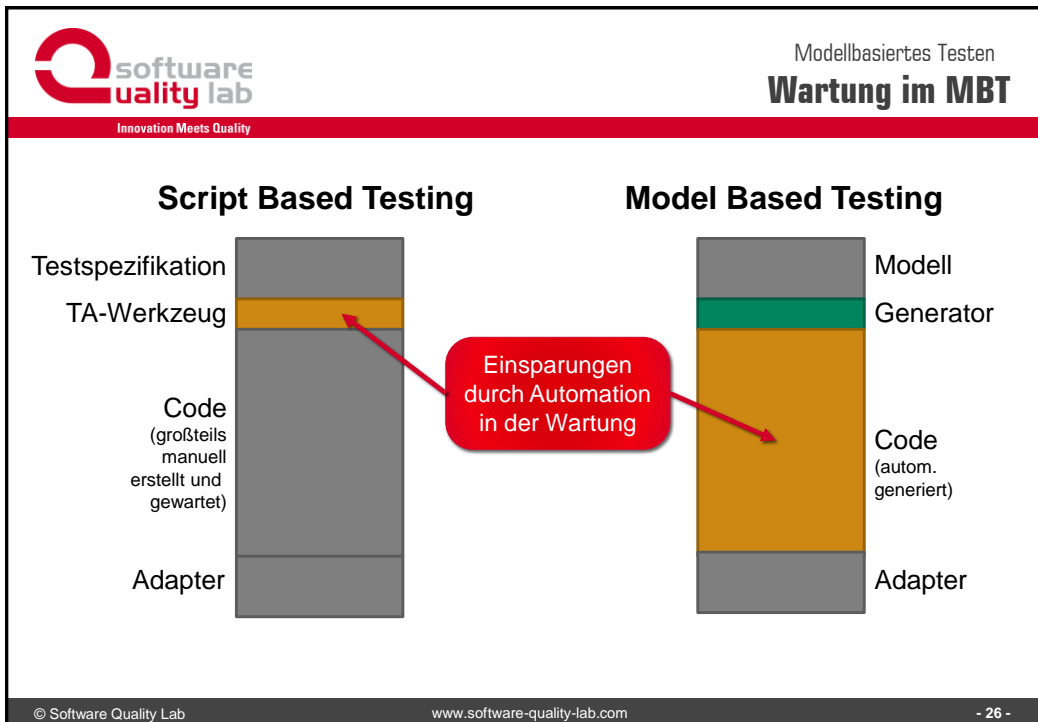
Quelle: <http://www.cukes.info/>


Modellbasiertes Testen

Wenn Sie noch produktiver werden wollen!



Modelle sind z.B. Aktivitätsdiagramme,
Zustandsdiagramme, Sequenz-
Diagramme, Klassenmodell, Datenbank-
Schema, Masken-Definition, etc.





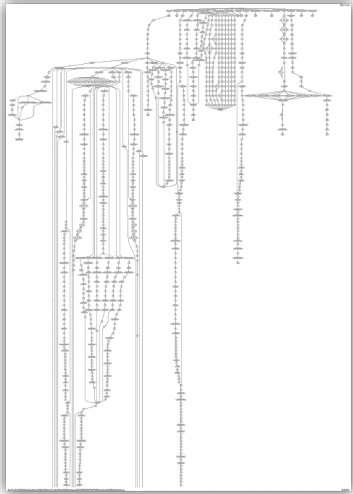
**software
uality lab**

Innovation Meets Quality

Modellbasiertes Testen

Fallbeispiel: Medizinprodukt

- Ausgangslage
 - Ca. 200 spezifizierte manuelle Tests
 - Manuelle Durchführungszeit ca. 4 Wochen
- Ziel: möglichst effektive Automatisierung der Tests und bessere Abdeckung
- Umsetzung
 - Ca. 3 Wochen Aufwand für Modellerstellung (grafisches Modell – ca. A0 groß)
 - Ca. 1 Woche Aufwand für Konfiguration und Feinjustierung der Generierung

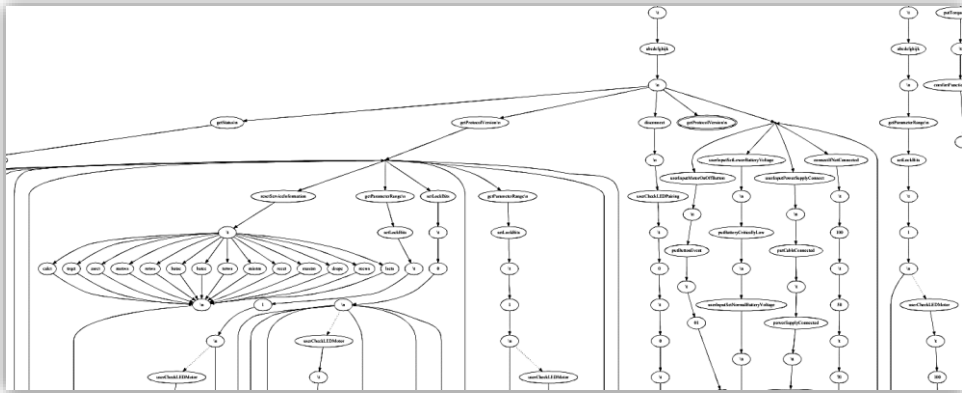


© Software Quality Lab

www.software-quality-lab.com

- 27 -

- Modell – Auszug



© Software Quality Lab

www.software-quality-lab.com

- 28 -

- Ergebnis

- Mehr als 1,8 Mio. (sinnvolle!) Testfälle generiert aus dem Modell !
- Davon ca. 60% sofort automatisch ausführbar
- Ausführungszeit ca. 5-7 Tage für > 1Mio. TF statt 4 Wochen für ca. 200 TF
- Extrem leichte Wartbarkeit der Testfälle (kaum Codierungsaufwand)

→Gewaltige Effizienzsteigerung!

→ Manuell und mit herkömmlicher Automatisierung ist diese Abdeckung und Absicherung NICHT zu erreichen!

→ Hocheffektive Absicherung mit guter Abdeckung!



© Software Quality Lab

www.software-quality-lab.com

- 29 -

- Künftig wird man beim HIL-Testen fragen:

**Programmieren Sie noch
oder
modellieren Sie schon?**



Fragen?



Software Quality Lab

INNOVATION MEETS QUALITY

Academy | Consulting | Operational Services | Tool Expertise