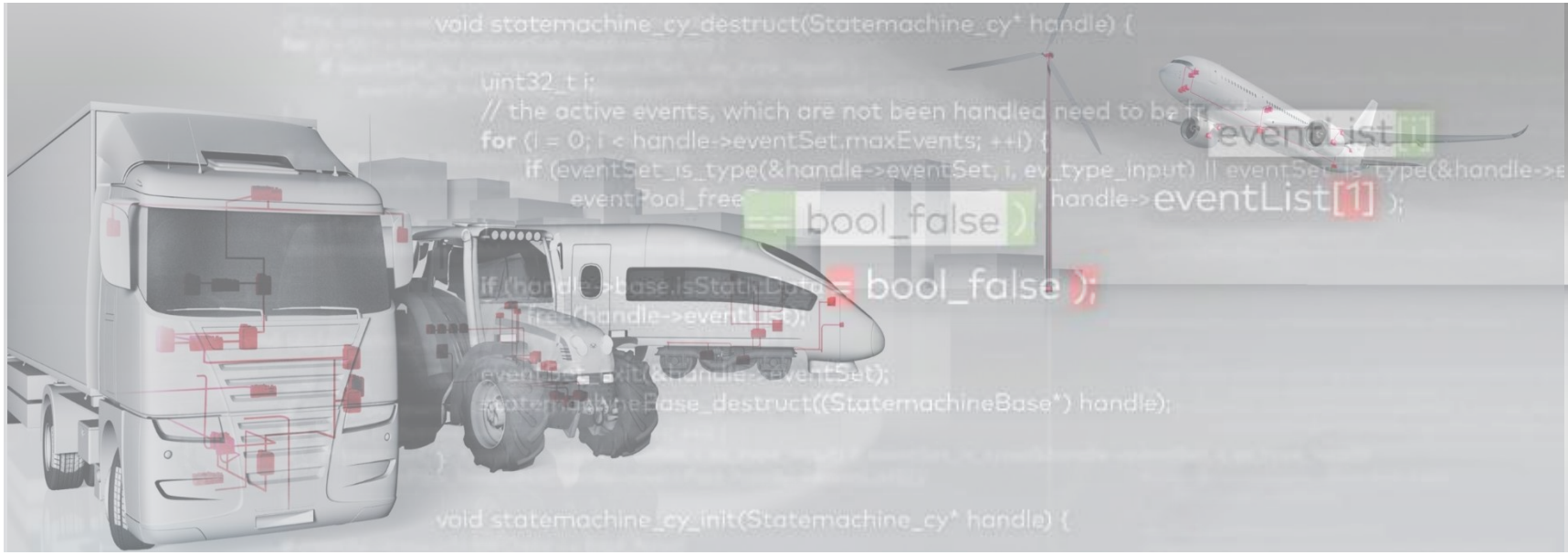
VECTOR



Wie White Box Testen die Code Qualität steigert

Embedded Testing, Dornbach, 4. Juli 2019

Agenda

- **Wer wir sind**
- Whitebox Testing
- Code Coverage Grundlagen
- 100% Code Coverage
- Weitere Vorteile des Whitebox-Testens
- Demo

Vector auf einen Blick

Mitarbeiter:
> 2.500
Vectorianer

Umsatz:
654 Mio. €
in 2018

Vereinigungen:
Mitarbeit in
15 Gremien

Kunden:
> 7.500 Firmen
in 72 Ländern

Niederlassungen:
26 Standorte in
12 Ländern

Verbundene Unternehmen:
GiN | CSM | BASELABS

- Vector Stiftung
  - 60% des Kapitals
  - Dauerhafte Sicherung der Unabhängigkeit der Vector Gruppe
  - Unterstützung von Forschung, Bildung, karitative Einrichtungen

Geschäftsfelder und Produktbeispiele

Entwicklung verteilter Systeme
PREEvision

Embedded Software und Systeme
MICROSAR, CANbedded, VC ECU, Kundenprojekte

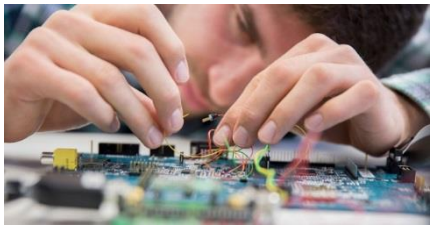
Testing
CANoe, CANalyzer, vTESTstudio, VT System, VectorCAST

Diagnose
CANdelaStudio, Indigo, vFlash, CANoe.DiVa

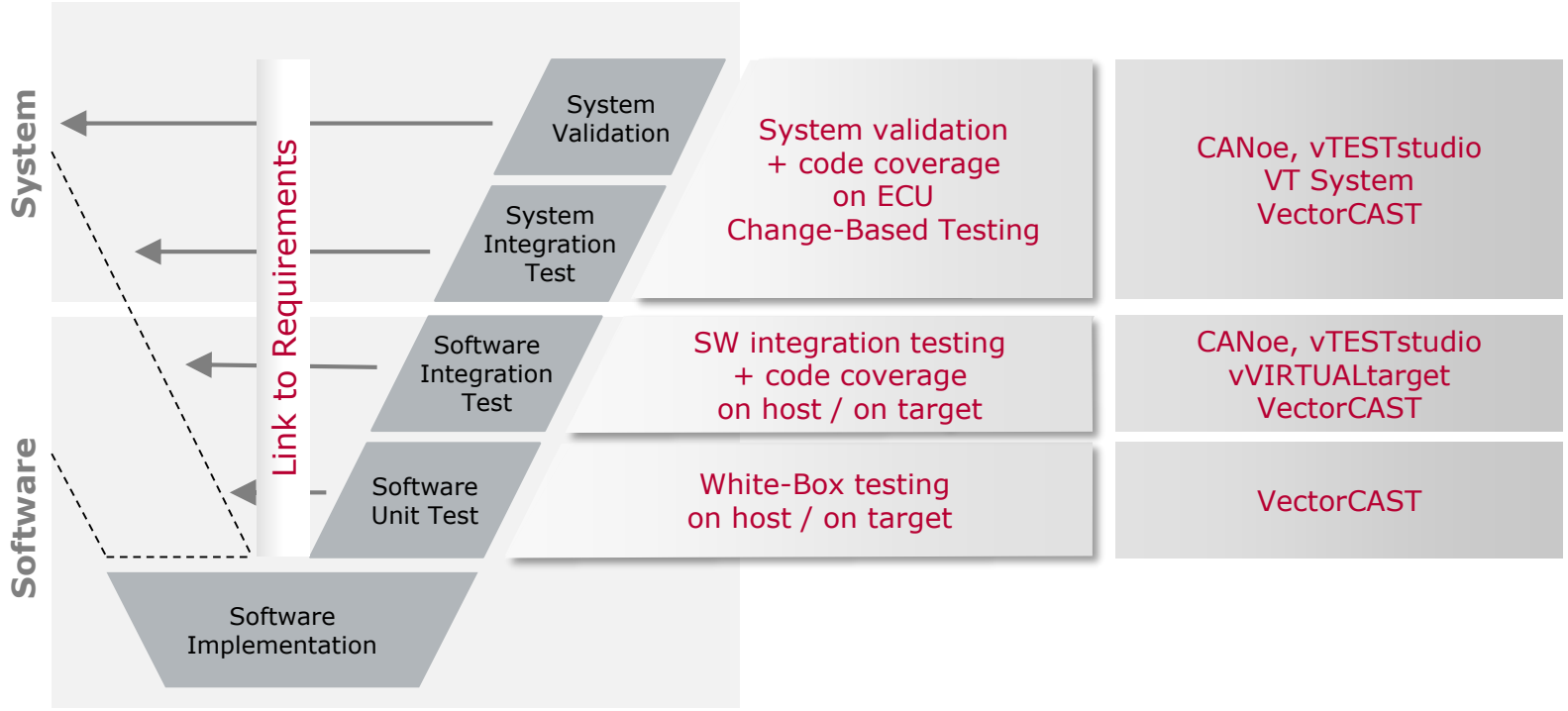
Steuergeräte-Kalibrierung
CANape, VX1000, vCDM, vADASdeveloper, ASAP2 Tool-Set

Messtechnik
vMeasure exp, vSignalyzer, vMDM, Logger, Analogmesstechnik

Beratung
Consulting Services, Engineering Services



Vector Testing Solution – Test Tools for all Test Phases



Agenda

- Wer wir sind
- **Whitebox Testing**
- Code Coverage Grundlagen
- 100% Code Coverage
- Weitere Vorteile des Whitebox-Testens
- Demo

VECTOR

Was ist „Blackbox Testing“?

- Keine Kenntnisse über die innere Struktur
  - Code-Agnostisch
  - Algorithmen-Agnostisch
- Üblicherweise Anforderungs-basiert
  - Funktionale Tests
  - Stabilitäts-Tests
  - Laufzeit-Tests
- Vorteile
  - Tester ist unvoreingenommen
- Nachteile
  - Keine Kenntnisse über Vollständigkeit

VECTOR

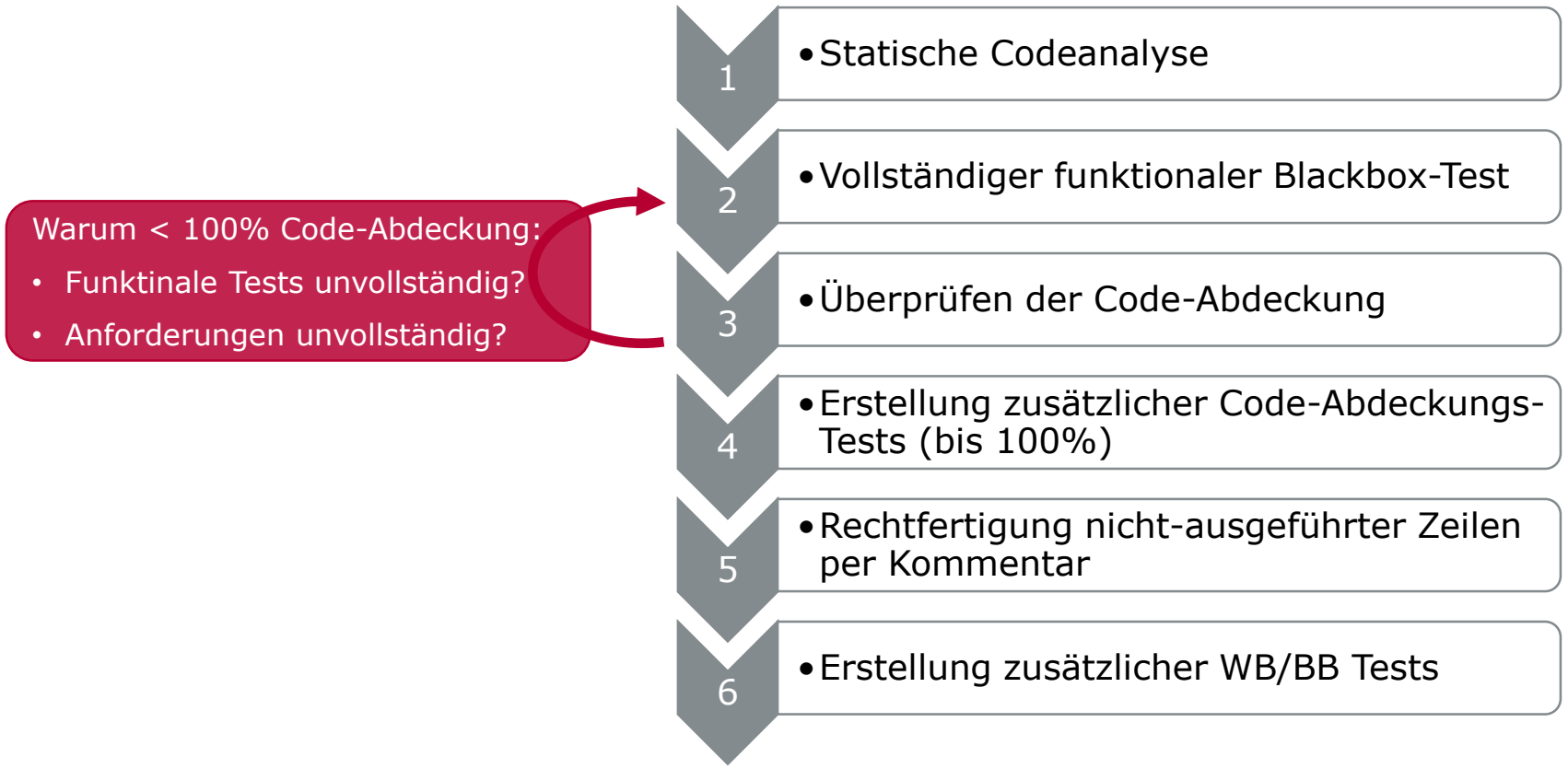
Was ist „Whitebox Testing“?

- Kenntnisse über die innere Struktur
- Zum Teil Ziel-Plattform-Abhängig (Wertebereiche)
- Vorteile
  - Vollständigkeits-Analyse und Nachweis
  - Testfallfindung
    - Code-Abdeckung
    - Wertebereiche
  - Problemzonen-Analyse und Test-Fokussierung
    - Komplexe Code-Bereiche
    - Algorithmen Analyse
    - Neuer Code vs. Re-used Code
  - Automatisiertes Testen
    - Statische Analysen
    - Finden dynamischer Testfälle
- Nachteile
  - Tester ist voreingenommen
  - Programmierkentnisse hilfreich
  - Offenlegung von Firmengeheimnissen

VECTOR

Graybox Testing?

- Jede Test-Methode hat Vor- und Nachteile
- Sinnvoll ist eine Kombination aus Black- und Whitebox-Tests



Agenda

- Wer wir sind
- Whitebox Testing
- **Code Coverage Grundlagen**
- 100% Code Coverage
- Weitere Vorteile des Whitebox-Testens
- Demo

VECTOR

Code-Abdeckung messen? Warum?

Metrics

Statement+MC/DC

Unit	Subprogram	Complexity	Function Coverage	Function Calls	Statements	Branches	Pairs
manager	Add_Included_Dessert	3	100%		2 / 4 (50%)	5 / 17 (29%)	0 / 6 (0%)
	Place_Order	5	100%	3 / 3 (100%)	10 / 22 (45%)	2 / 6 (33%)	
	Clear_Table	2	0%	0 / 2 (0%)	0 / 12 (0%)	0 / 5 (0%)	0 / 1 (0%)
	Get_Check_Total	1	0%	0 / 1 (0%)	0 / 2 (0%)	0 / 1 (0%)	
	Add_Party_To_Waiting_List	3	100%		4 / 7 (57%)	5 / 11 (45%)	0 / 3 (0%)
	Get_Next_Party_To_Be_Seated	2	0%		0 / 3 (0%)	0 / 5 (0%)	0 / 1 (0%)
TOTALS	**6**	**16**	**3 / 6 (50%)**	**3 / 6 (50%)**	**16 / 50 (32%)**	**12 / 45 (26%)**	**0 / 11 (0%)**
GRAND TOTALS	**6**	**16**	**3 / 6 (50%)**	**3 / 6 (50%)**	**16 / 50 (32%)**	**12 / 45 (26%)**	**0 / 11 (0%)**

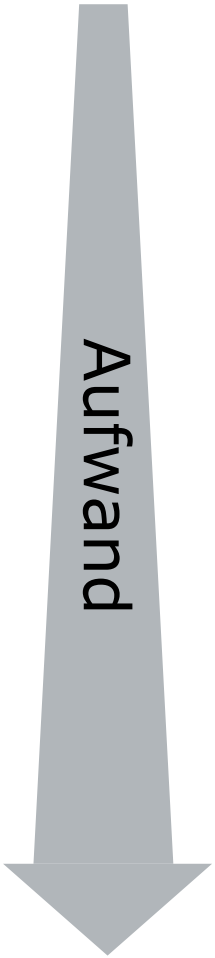
- Hilft bei der Testfall-Findung
- Nachweis des vollständigen Tests für Audit/Zertifizierung
- Wichtige Metrik:
  - Test (Un-)Vollständigkeit
  - Qualität bestehender Testfälle
  - Requirements Traceability
- Grundlage anderer Optimierungen
  - Z.B. Änderungs-basiertes Testen

VECTOR

Code Coverage Level

- Function coverage
  - Jede Funktion wurde aufgerufen
- Function call coverage
  - Jeder Funktionsaufruf wurde ausgeführt
- Statement coverage
  - Jede Code-Zeile wurde ausgeführt
- Decision (Branch) coverage
  - Jeder Entscheidungspunkt wurde mit ausgeführt mit Ausgang wahr und falsch
- Modified condition/decision coverage (MC/DC)
  - Jeder Entscheidungspunkt und jede Unter-Bedingung wurde mit Ausgang wahr und falsch ausgeführt
  - Nachweis der Unabhängigkeit der Unter-Bedingungen

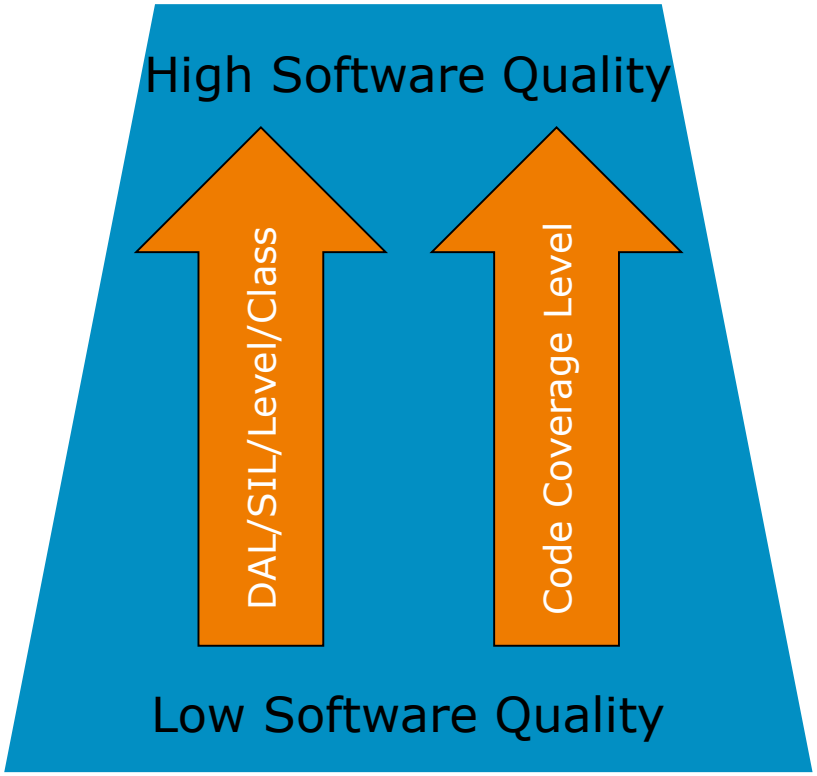
```
(T)          foo
( ) (F)        if (i==1)
                 bar1();
               else {
 *               bar1();
 *               bar2();

     *         if (a&&b)

( ) (F)        if (a&&b)

( ) (F)        if (
(T) ( )        a&&
( ) (F)        b)
```



VECTOR

Code Coverage und Zertifizierung

Requirement	Statement	Branch	MC/DC
DO-178 B/C (Avionics)			
Level A	•	•	•
Level B	•	•	
Level C	•		
ISO-26262 (Automotive)			
ASIL D	•	•	•
ASIL B/C	•	•	
ASIL A	•		
IEC-61508 (Industrial)			
SIL 4	•	•	•
SIL 3	•	•	
SIL 1/2	•		
EN-50128 (Railway)			
SIL 4	•	•	•
SIL 3	•	•	
SIL 1/2	•		
IEC-62304 (Medical)			
Class C	•	•	•
Class B	•	•	
Class A	•		



VECTOR

Statement Coverage

- Jede Zeile sollte ausgeführt werden

Beispiel für Statement coverage, condition = wahr

```
7 1     *       int* p = 0;
7 2     *       if (condition)
7 3     *           p = &variable;
7 4     *       *p = 123;
```

100% Code Coverage

- Das muss doch reichen. Oder?

Beispiel für Statement coverage, condition = falsch

```
7 1     *       int* p = 0;
7 2     *       if (condition)
7 3             p = &variable;
7 4     *       *p = 123;
```

Segmentation Violation

VECTOR

Branch Coverage

- Jede Entscheidung sollte mit beiden Ergebnissen wahr/falsch ausgeführt werden

Beispiel für Branch Coverage.
Testfall 1: condition1 = falsch, condition2 = ?
Testfall 2: condition1 = wahr, condition2 = wahr

100% Code Coverage

```
9 1     *       int* p = 0;
9 2     *       if (condition1 && (condition2 || function1(*p)))
9 3     *            statement1;
                 else
9 4     *            statement2;
```

- Das muss doch reichen. Oder?

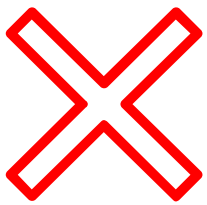
Testfall 3: condition1 = wahr, condition2 = falsch

Segmentation Violation

```
9 1     *       int* p = 0;
9 2     *       if (condition1 && (condition2 || function1(*p)))
9 3             statement1;
                else
9 4             statement2;
```

VECTOR

Warum 100% Code Coverage?



100% Code Coverage bedeutet **nicht**

- Der Code ist korrekt
- Der Code ist vollständig
- Der Test ist vollständig

100% Code Coverage **bedeutet**

- Jede Code-Stelle wurde ausgeführt ohne Crash
- Es gibt keinen „toten" Code
- Wir haben gewissenhaft getestet und ein Mindestmaß an Testfällen erreicht
- "Program elements […] covered by any test case see about half as many bug-fixes as those not covered" *

Kein Testfall sollte nur der Code Coverage dienen sondern immer auch etwas überprüfen!

*Source: Can Testedness be Effectively Measured? Rahul Gopinath; Oregon State University; based on 250 real world programs

Agenda

- Wer wir sind
- Whitebox Testing
- Code Coverage Grundlagen
- **100% Code Coverage**
- Weitere Vorteile des Whitebox-Testens
- Demo

VECTOR

Unerreichbarer Code?

- Ziel: 100% Code Abdeckung aus dynamischen, funktionalen Tests
- Problem: Sehr häufig Code unerreichbar
- Lösung:
  - Fault Injection
  - Covered by Analysis

VECTOR

Fault Injection

- Code Änderung um ansonsten unerreichbaren Code zu erreichen
- Beispiel: default case

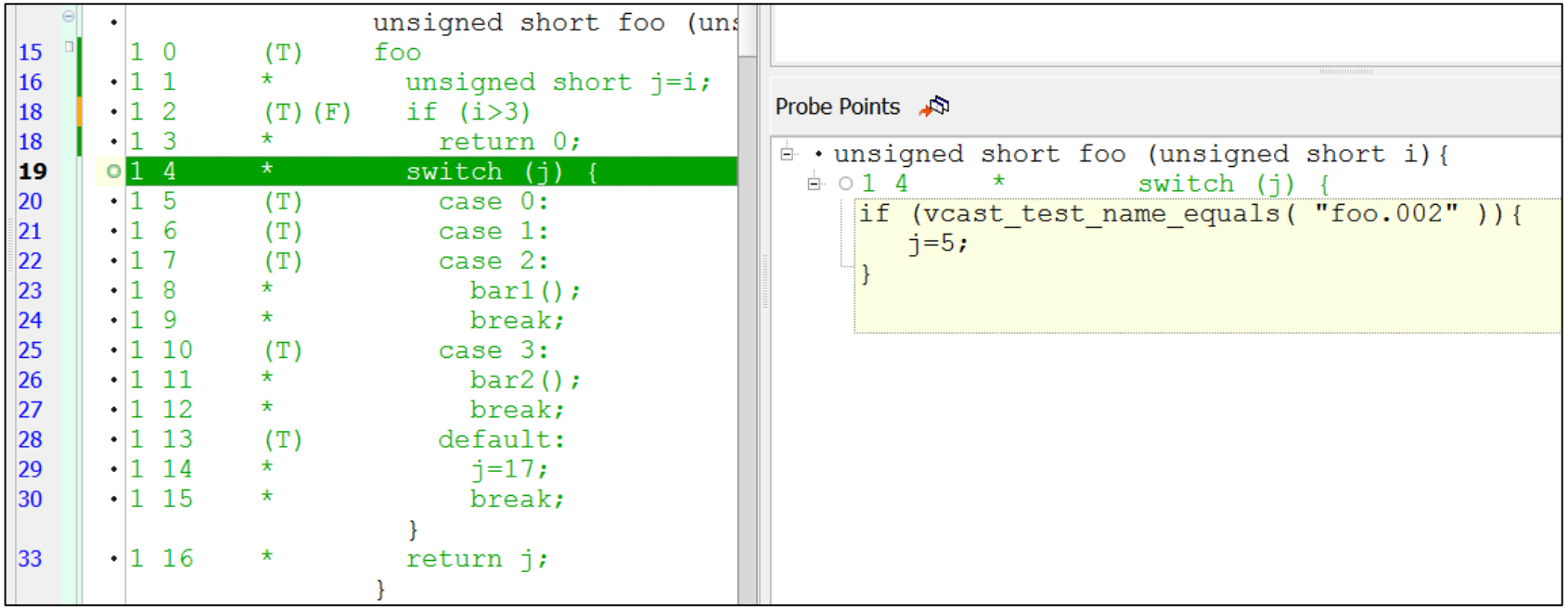
```
(T)       foo
 *          unsigned short j=i;
(T) (F)     if (i>3)
 *            return 0;
 *          switch (j) {
(T)           case 0:
(T)           case 1:
(T)           case 2:
 *              bar1();
 *              break;
(T)           case 3:
 *              bar2();
 *              break;
( )           default:
                j=17;
                break;
            }
 *          return j;
          }
```

- Mögliche Vorgehensweisen:
  - Per Debugger und breakpoint
  - Manuelle Code-Änderung
  - Automatisierbare Code Injection

VECTOR

Automatisierbare Code Injection - Beispiel

- Code Injection Editor



- Code Injection Listing

Applied probe points:

ID	Unit	Function	Line	Context Before	Context After	Code Before	Code After

1	manager	foo	switch (j	unsigned short j=i;
				if (i>3) return 0;
					case 0:
					case 1:
						if (vcast_test_name_equals("foo.002")){
						j=5;
						}	

VECTOR

Covered by Analysis

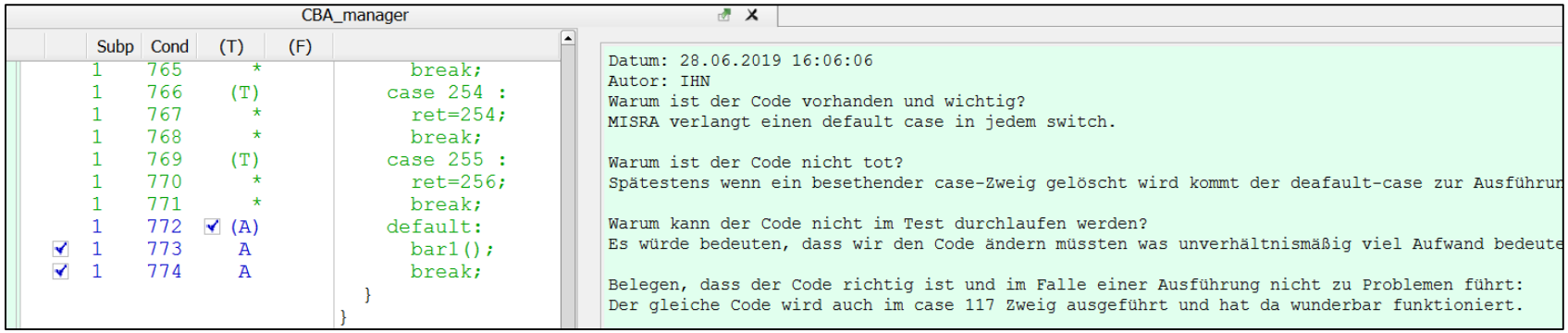
- Problem: Code der auch durch Code-Änderung nicht zu erreichen ist
- Beispiel

```
unsigned char cover_me (unsigned char i){

  unsigned char ret=0;
  unsigned char j = i;
  switch (j){
    case 0 :  ret=0; break;
    case 1 :  ret=1; break;
    case 2 :  ret=2; break;

    ...

    case 254 :  ret=254; break;
    case 255 :  ret=256; break;
    default: bar1(); break;
  }
}
```

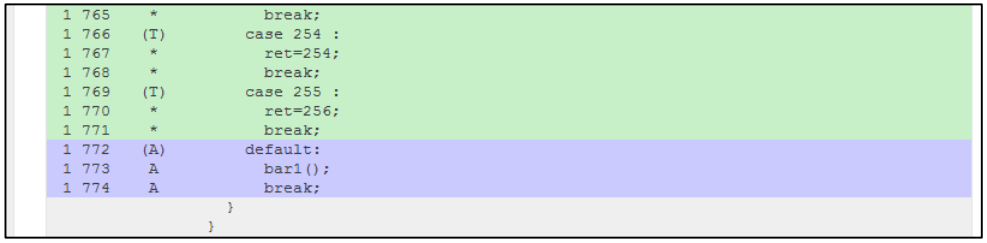
- Mögliche Vorgehensweisen:
  - Code rauswerfen
  - Rechtfertigung schreiben
    - Defensiver Code:
      - Warum ist der Code vorhanden und wichtig
      - Warum ist der Code nicht tot
      - Warum kann der Code nicht im Test durchlaufen werden
      - Belegen, dass der Code richtig ist und im Falle einer Ausführung nicht zu Problemen führt
    - Deaktivierter Code:
      - Warum ist der Code vorhanden
      - Belegen, dass der Code nicht angesprungen werden kann und keinen Einfluss auf das System hat

Covered by Analysis

- Covered-By-Analysis Editor



- Coverage on Source Code:



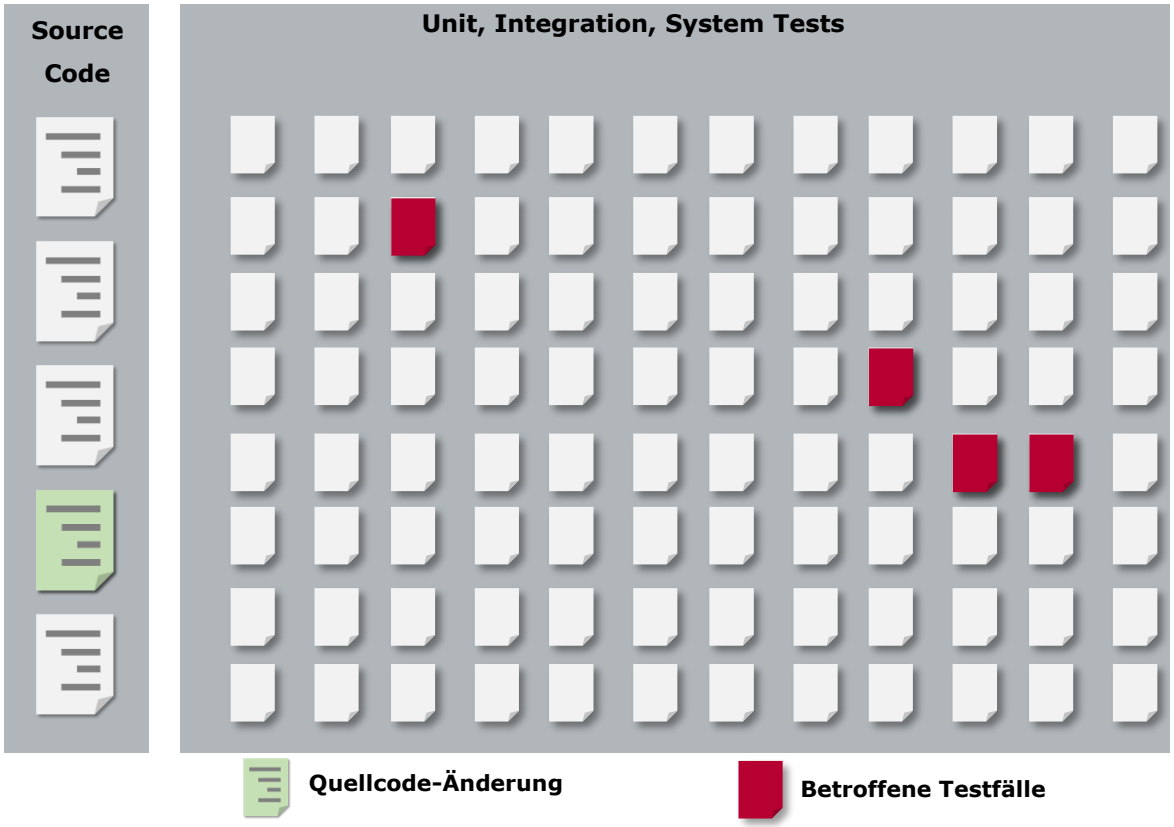
- Metriken:

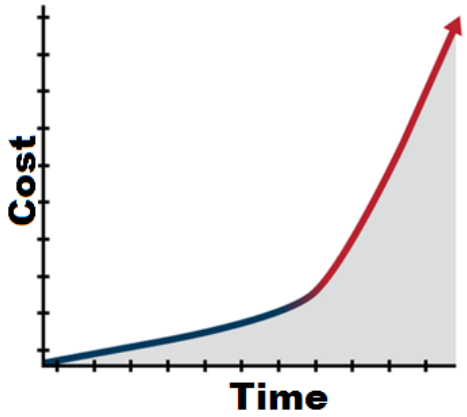
Unit	Subprogram	Complexity	Function Coverage	Statements	Branches
manager	cover_me	257	100%	774 / 774 (100%)	258 / 258 (100%)
	Analysis			*3 / 774 (1%)*	*1 / 258 (1%)*
	Execution			*771 / 774 (99%)*	*257 / 258 (99%)*
	foo	6	100%	16 / 16 (100%)	8 / 8 (100%)
	Add_Included_Dessert	3	100%	4 / 4 (100%)	5 / 5 (100%)

Agenda

- Wer wir sind
- Whitebox Testing
- Code Coverage Grundlagen
- 100% Code Coverage
- **Weitere Vorteile des Whitebox-Testens**
- Demo

Änderungs-basiertes Testen





VECTOR

Automatische Testfall-Generierung

- Analyse des Codes:
  - Pfad-Analyse
  - MC/DC-Analyse
  - Wertebereiche der Schnittstellenparameter
- Automatische Testfall-Generierung
  - Pfad-Abdeckung
  - MC/DC Testabdeckung
  - Äquivalenklassen Tests
    - Minimal-Werte
    - Maximal-Werte
    - Zwischenwerte
- Basis für funktionale Tests
  - Verknüpfen mit Requirements

VECTOR

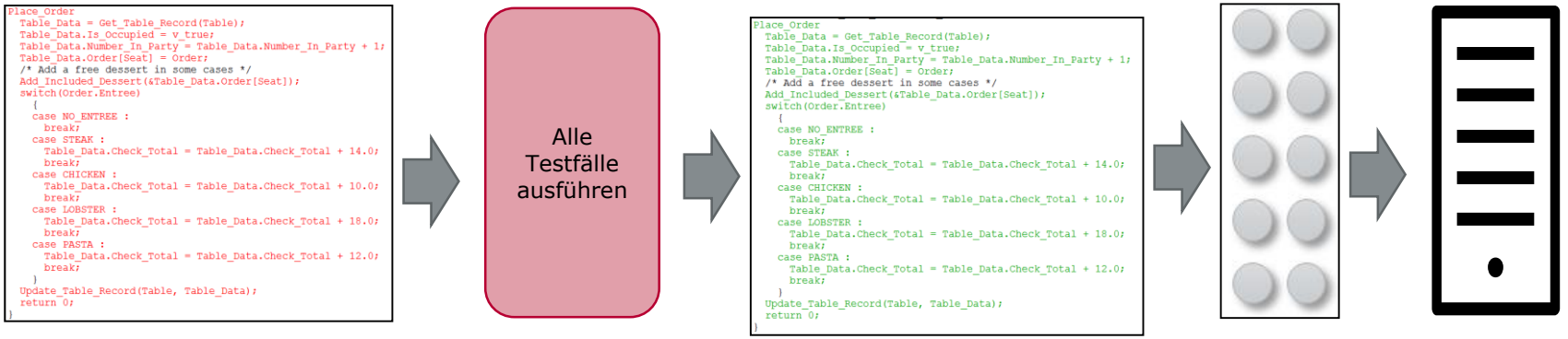
Fuzz-Testing-Optimierung

- Eigentlich eine Blackbox Testmethode
- Kann durch Coverage-Analyse optimiert werden
  - Beispiel: Bus-Meldungen
  - Zufälliges Ändern aller Parameter
    - Viele Meldungen werden im Protokoll-Stack abgewiesen
  - Ansteigende Coverage = Applikations-Level erreicht
    - Beibehalten der meisten Parameter
    - Variieren nur einzelner Parameter

VECTOR

Whiter-Than-White-box Testing

- Problem: Altlasten (Legacy Code)
  - Seit Jahren in Betrieb. Funktioniert und keiner weiß warum
  - Keine Testfälle, keiner traut sich den Code anzufassen. Kein Re-Design
- Lösung:
  - Automatisches "Baselinen" des Software-Verhaltens
    - > Automatische Testfall-Generierung auf Basis von Quellcode-Analyse
    - > Testfälle ausführen und Ist-Werte ermitteln
    - > Ist-Werte als Test-Ergebnis-Werte speichern
  - Regressions-Test für Re-design, Bugfixes, Neu-Entwicklung



Pfad-Analyse
Parameter Werte-Bereichs-Analyse
Automatische Testfall-Generierung
(Eingabe-Werte)

Ist-Werte speichern:
globale Variablen
Rückgabe-Werte

Gute Code Coverage

Baseline Testfälle

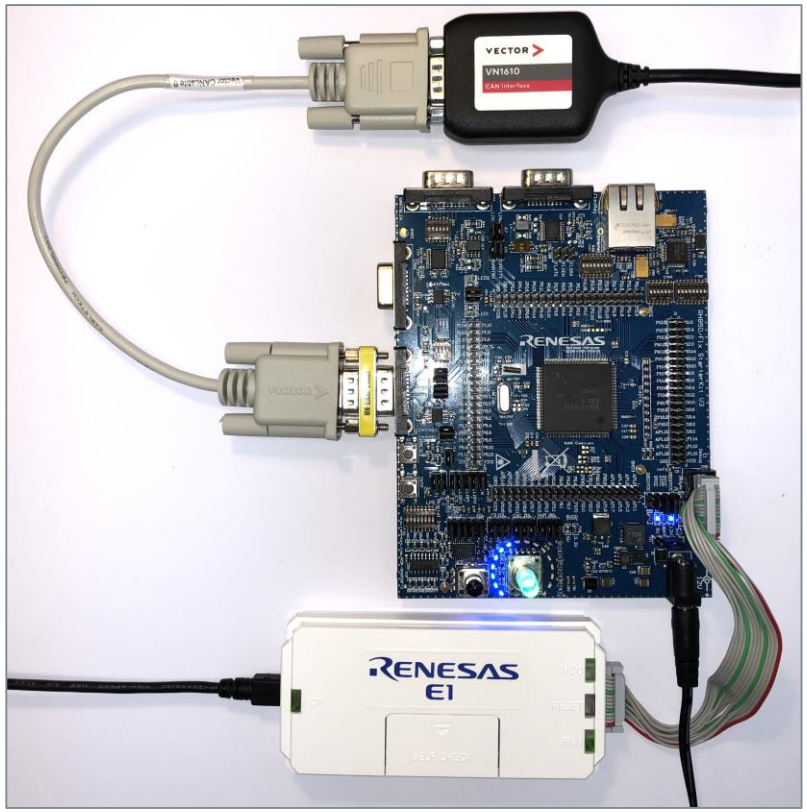
Regressions-Test

Agenda

- Wer wir sind
- Whitebox Testing
- Code Coverage Grundlagen
- 100% Code Coverage
- Weitere Vorteile des Whitebox-Testens
- **Demo**

Demo Hardware

CAN Cable



VN1610

USB to Laptop

Renesas

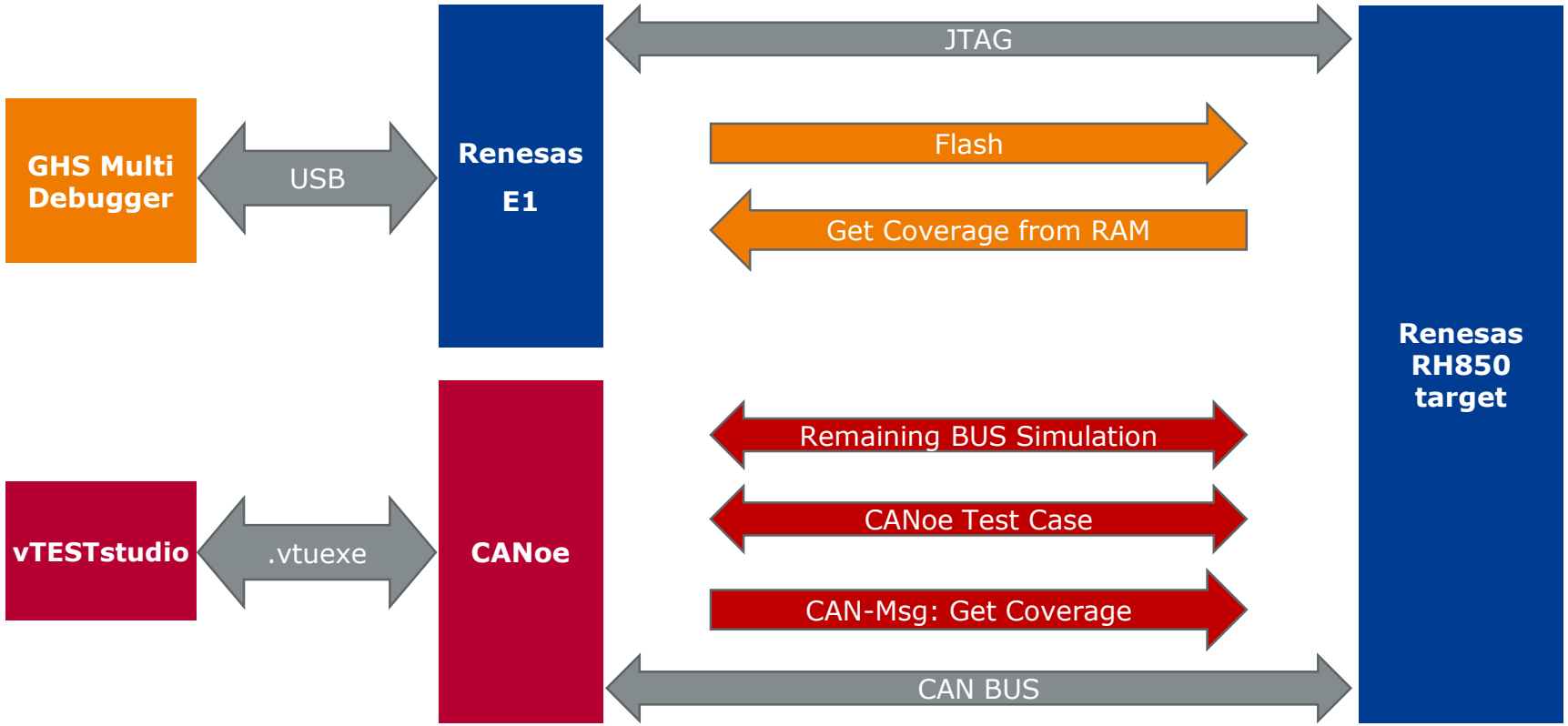
RH850

USB to Laptop

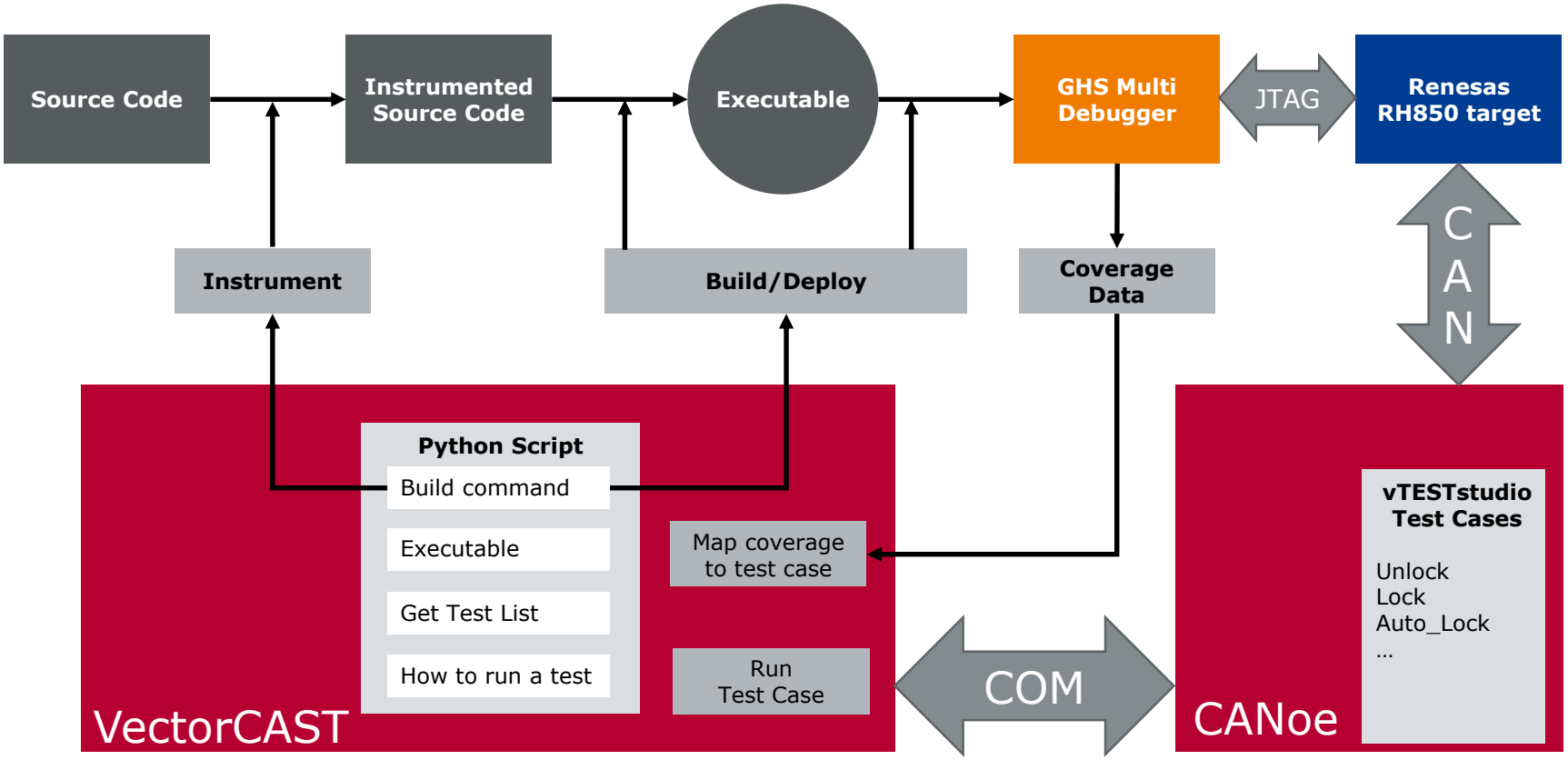
Renesas

E1 Emulator

Demo Setup


GHS Multi Debugger
USB
Renesas E1
JTAG
Flash
Get Coverage from RAM
Renesas RH850 target
vTESTstudio
.vtuexe
CANoe
Remaining BUS Simulation
CANoe Test Case
CAN-Msg: Get Coverage
CAN BUS

Demo Workflow



Mehr Informationen!
Besuchen Sie unsere Website für:
> News
> Produkte
> Demo-Software
> Support
> Seminare und Workshops
> Kontaktadressen

www.vector.com

Author:
Ingo Nickles
Vector Germany