

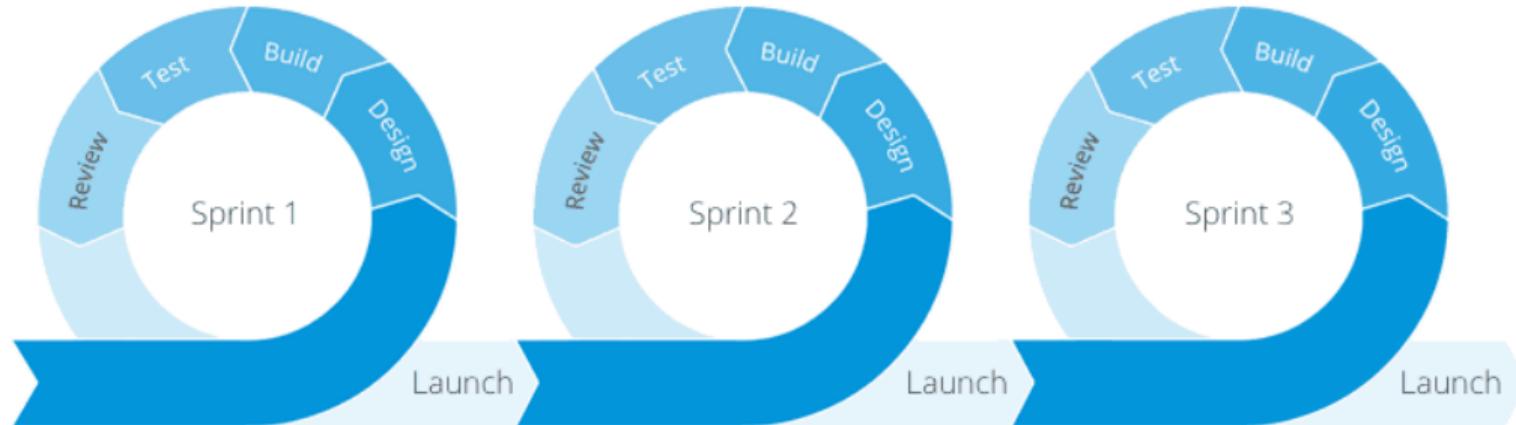


Agiles Testen für Mikrocontroller-Projekte

Daniel Penning

Embedded Testing 2025, Unterhaching

Agile Entwicklung



Quelle: Dzone.com

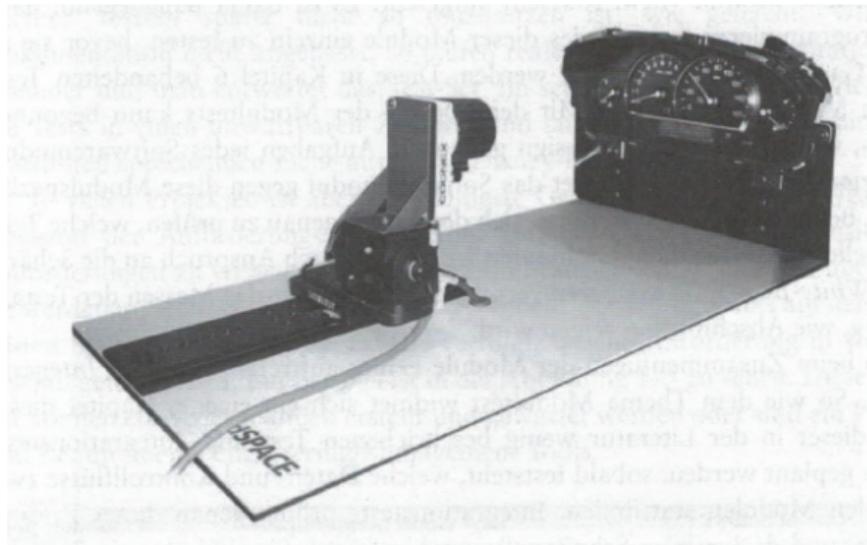
Agiles Testen



- | Schnelles Feedback durch Tests essentiell in agilen Projekten.
- | Entwicklungsbegleitendes Testen durch Entwickler (ohne weitere Qualifikationen) ersetzt nicht den Nachweis durch eine spezialisierte Test-Abteilung.

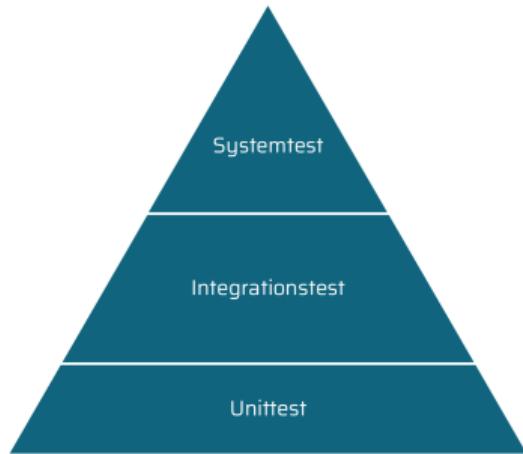
Agiles Testen in der Realität?

- | Ein agiler Testansatz erfordert ein Testen im Sprint.
- | Der Aufbau von spezifischen HiL-Testsystemen dauert Monate-Jahre.
- | Wie kann trotzdem sinnvoll agil getestet werden?

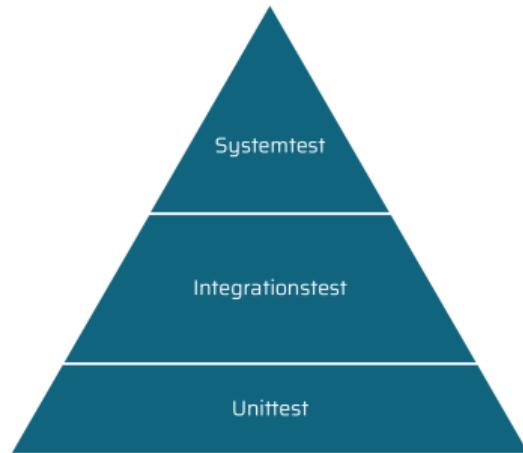


HiL-Teststand mit Bilderkennung zur Bestimmung der Tachonadel-Position.
Foto ©dSpace GmbH

Die klassische Testpyramide



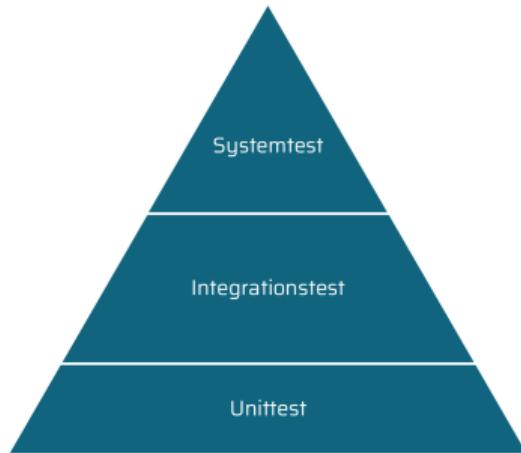
Die klassische Testpyramide



„Eine Teststufe mit dem Schwerpunkt auf einer einzelnen Hardware- oder Softwarekomponente. Eine Komponente ist die kleinste Einheit eines Systems, die für sich alleine getestet werden kann.“

Unitest

Die klassische Testpyramide



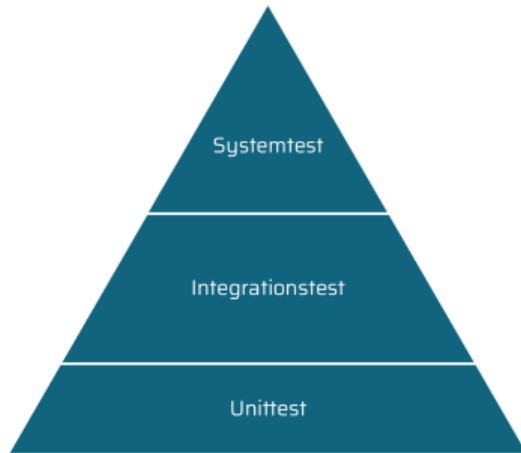
„Eine Teststufe mit dem Schwerpunkt auf einer einzelnen Hardware- oder Softwarekomponente. Eine Komponente ist die kleinste Einheit eines Systems, die für sich alleine getestet werden kann.“

Unitest

„Eine Teststufe mit dem Schwerpunkt auf dem Zusammenwirken zwischen Komponenten oder Systemen.“

Integrationstest

Die klassische Testpyramide



„Eine Teststufe mit dem Schwerpunkt auf einer einzelnen Hardware- oder Softwarekomponente. Eine Komponente ist die kleinste Einheit eines Systems, die für sich alleine getestet werden kann.“

Unittest

„Eine Teststufe mit dem Schwerpunkt auf dem Zusammenwirken zwischen Komponenten oder Systemen.“

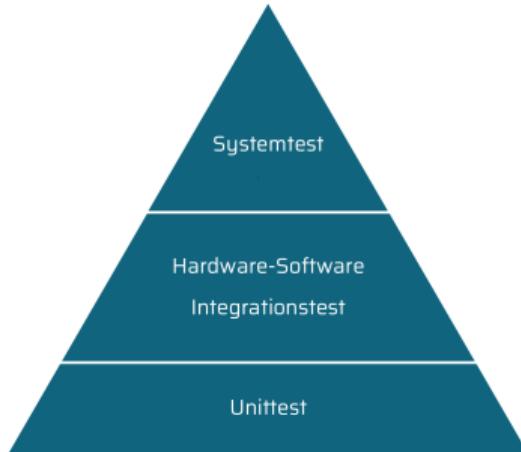
Integrationstest

„Eine Teststufe mit dem Schwerpunkt zu verifizieren, dass ein System als Ganzes die spezifizierten Anforderungen erfüllt.“

Systemtest

Definitionen aus dem ISTQB Glossar: <https://glossary.istqb.org>.

Die Testpyramide für Embedded-Projekte



- | Üblicher (Software/Software)-Integrationstest prüft die Interaktion von zwei oder mehreren Software-Modulen untereinander.
- | Hardware/Software-Integrationstest prüft das Zusammenspiel der Software mit der Hardware.
- | Zunehmende Bedeutung, da Komplexität im Interface zwischen Software und MCU steigt ("5000 Seiten Reference Manual")

Was sind Unitests?

„Eine Teststufe mit dem Schwerpunkt auf einer einzelnen Hardware- oder Softwarekomponente. Eine Komponente ist die kleinste Einheit eines Systems, die für sich alleine getestet werden kann.“ Unitest

```
int max(int a, int b) {  
    if(a < b)  
        return b;  
    return a;  
}
```

Whitebox

klein und schnell

keine externe Ressourcen

typisch: Entwickler-Tests

Unitests durchführen (manuell)

```
int max(int a, int b) {  
    if(a < b)  
        return b;  
    return a;  
}
```

```
bool a_smaller_than_b() {  
    int a=5, b=10;  
    int result = max(a, b);  
    return result == b;  
}
```

```
#include <stdio.h>  
#include "max.h"  
#include "test_a_smaller_than_b.h"  
  
int main() {  
    int failed = 0;  
    if(!a_smaller_than_b()) {  
        printf("failed.\n");  
        failed++;  
    }  
    return failed;  
}
```

Unitests durchführen (mit Framework)

Aufgaben

- | TestCase Registrierung
- | TestCase Ausführung
- | Reporting

Framework Beispiele

- | Unity
- | GoogleTest
- | Cantata, TESSY

```
#include "max.h"
#include <gtest/gtest.h>
TEST(max_tests, a_smaller_than_b) {
    ASSERT_EQ(10, max(5, 10));
}
```

```
[=====] Running 1 test from 1 test suite.
[-----] Global test environment set-up.
[-----] 1 test from max_tests
[ RUN      ] max_tests.a_smaller_than_b
[     OK  ] max_tests.a_smaller_than_b (0 ms)
[-----] 1 test from max_tests (1 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test suite ran. (4 ms total)
[  PASSED  ] 1 test.
```

Hardware-Abhängigkeiten im Code

```
int max(int a, int b) {  
    if(a < b)  
        return b;  
    return a;  
}
```

| Kann auf dem Host (*off-target*) getestet werden

```
uint32_t crc32(uint8_t * buffer, size_t len) {  
    uint32_t * arr = buffer;  
    uint32_t * end = arr + (len/4);  
    CRC->CR = CRC_CR_RESET;  
  
    while (arr < end) {  
        uint32_t val = *arr++;  
        CRC->DR = __builtin_bswap32(val);  
    }  
    return CRC->DR;  
}
```

| Muss auf dem MCU (*on-target*) getestet werden

Lösung für Unit-Tests

Off-Target (auf PC)

- | "schnell"
- | unbegrenzte Ressourcen
- | sehr leichte Automatisierung

Lösung für Unit-Tests

Off-Target (auf PC)

- | „schnell“
- | unbegrenzte Ressourcen
- | sehr leichte Automatisierung

On-Target (auf MCU)

- | „langsam“
- | begrenzte Ressourcen
- | Automatisierung erfordert Hardware

Lösung für Unit-Tests

Off-Target (auf PC)

- | „schnell“
- | unbegrenzte Ressourcen
- | sehr leichte Automatisierung

On-Target (auf MCU)

- | „langsam“
- | begrenzte Ressourcen
- | Automatisierung erfordert Hardware

Vorteile von **On-Target** Tests

- | nur eine Toolchain notwendig
- | gleiche Runtime Umgebung wie in Production
- | **erweiterte Testmöglichkeiten:** Laufzeiten bestimmen und gegen Regressionen sichern

Strategie für Unit-Tests

Für **jedes** ernsthaftes Projekt:

1. Off-Target Tests mit einem Unit Test Framework einrichten
2. Diese Tests in Continuous Integration (CI) automatisieren.

Strategie für Unit-Tests

Für **jedes** ernsthaftes Projekt:

1. Off-Target Tests mit einem Unit Test Framework einrichten
2. Diese Tests in Continuous Integration (CI) automatisieren.

Für sicherheitskritische Projekte:

1. Die gleichen Tests auch für MCU kompilieren.
2. Testumgebung mit MCU aufbauen
 - ▶ Evalboard, Custom-Hardware oder Testplatine
 - ▶ Flashing scripten (bspw. mit SEGGER J-Link)
 - ▶ Testergebnisse in Host zurücklesen (bspw. UART-USB Dongle)

Strategie für Unit-Tests

Für **jedes** ernsthaftes Projekt:

1. Off-Target Tests mit einem Unit Test Framework einrichten
2. Diese Tests in Continuous Integration (CI) automatisieren.

Für sicherheitskritische Projekte:

1. Die gleichen Tests auch für MCU kompilieren.
2. Testumgebung mit MCU aufbauen
 - ▶ Evalboard, Custom-Hardware oder Testplatine
 - ▶ Flashing scripten (bspw. mit SEGGER J-Link)
 - ▶ Testergebnisse in Host zurücklesen (bspw. UART-USB Dongle)

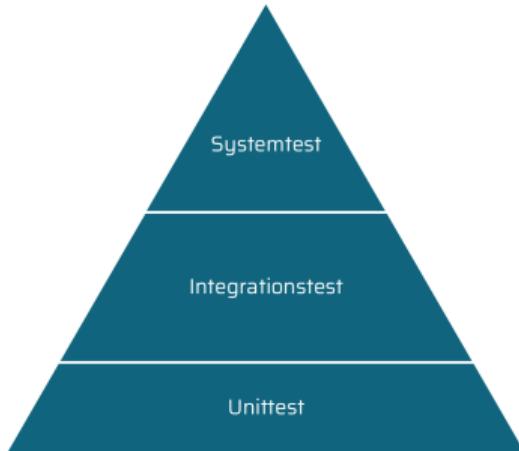
Für Projekte mit mehreren MCUs oder Laufzeitmessungen:

- | Testaufbau erweitern
- | Kommerzielle Lösungen evaluieren



embeff ExecutionPlatform

Der Integrationstest



„Eine Teststufe mit dem Schwerpunkt auf dem Zusammenwirken zwischen Komponenten oder Systemen.“

Integrationstest

Software-Software Integrationstest

- ▶ Konsequent strukturierte Unitests können als SW-SW Integrationstest verstanden werden.
- ▶ Literatur: *Bottom-Up Unitests*

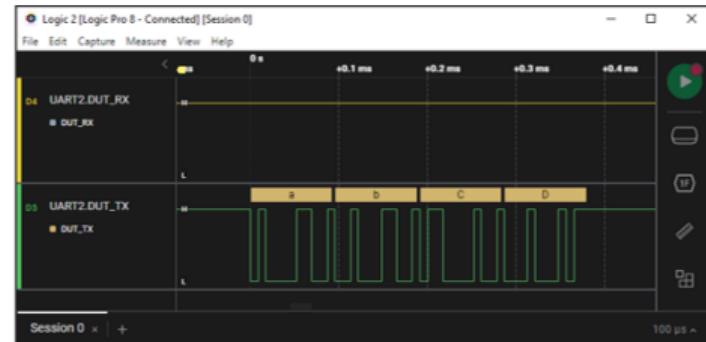
Hardware-Software Integrationstest

HW-SW Integration Tests auf Pin-Ebene

1. Code auf MCU ausführen

```
UART_HandleTypeDef huart2{};  
huart2.Instance = USART2;  
huart2.Init.BaudRate = 115200;  
huart2.Init.WordLength = UART_WORDLENGTH_9B;  
huart2.Init.StopBits = UART_STOPBITS_1;  
huart2.Init.Parity = UART_PARITY_ODD;  
/* More init settings... */  
HAL_UART_Init(&huart2);  
uint8_t transmitData[] = "abCD";  
HAL_UART_Transmit(&huart2, transmitData, 4, 0);
```

2. Pin-Verhalten prüfen

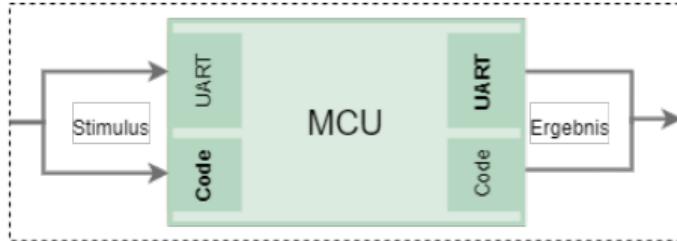


UART TX Signal abgleichen:

| Form: BaudRate, Parity, Stop Bits.

| Inhalt: 4 Bytes (abCD).

Ziel: Ein HW/SW Integration Test für UART



*** Test Cases ***

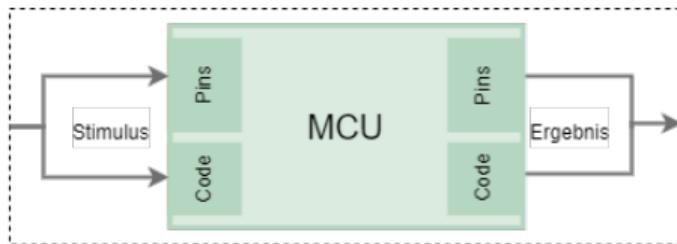
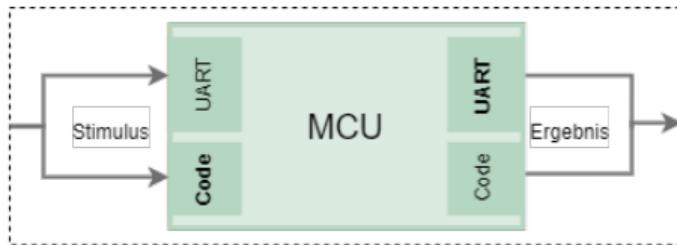
Send UART data with MCU and check that it is received with right settings

DUT Flash Firmware `example_endpoint_uart.bin`
UART0 Start Bitrate=115.2kbit/s Parity=No StopBits=1 DataBits=8

DUT Invoke `send_uart_data` # Cause microcontroller to send "abCD"

```
 ${expected} =     Convert To Bytes    abCD  
 ${received} =     UART0 Get Received  
 Should Be Equal    ${expected}    ${received}
```

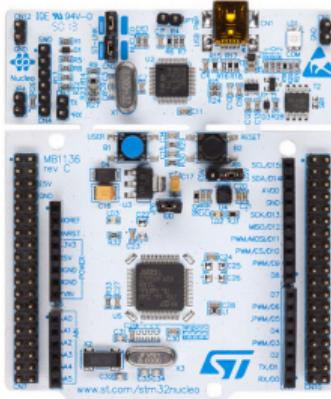
Lösung: Integrationstests als Open Loop interpretieren



1. Test-Start über Stimulus an den Pins *oder* Aufruf einer Funktion.
2. Ergebnis-Bewertung über Verhalten an den Pins und/oder Ergebnisse von Funktionen.

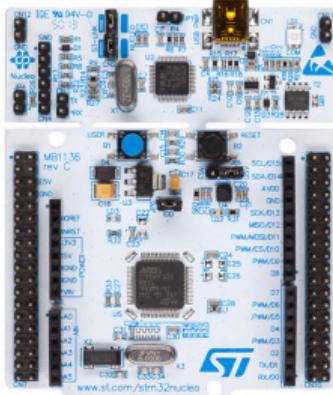
Lösung 1: Integrationstests selbst aufbauen

1. Relevante Pins zugreifbar machen



Lösung 1: Integrationstests selbst aufbauen

1. Relevante Pins zugreifbar machen



2. Dongles als Interface PC <-> Pins

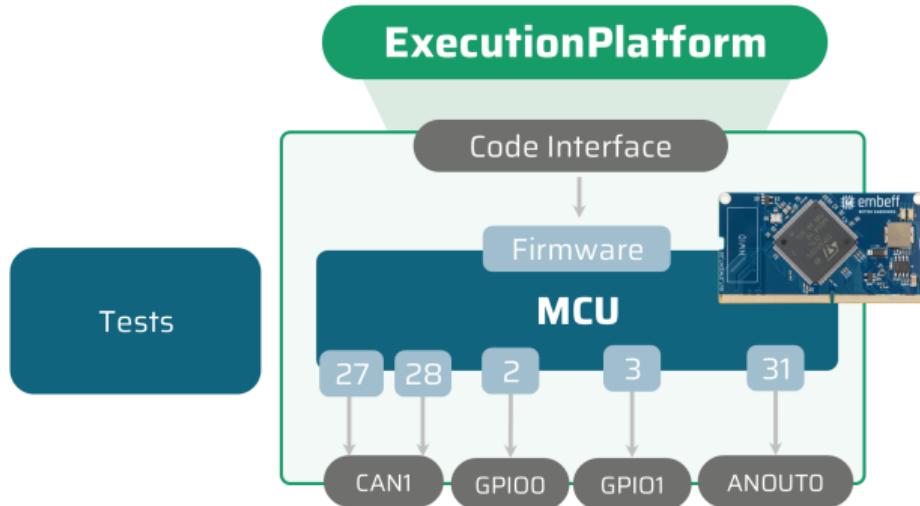


LABJACK T4



Vector USB-to-CAN

Lösung 2: Ausblick kommerzielle HW/SW Integration Tests



💡 Endpoints: Rekonfigurierbares Interface Test ↔ Pins.

Code Interface

- | DUT Flash Firmware

- | DUT Invoke

GPIO Endpoint

- | Set High, Set Low, Read

- | Generate High Pulse On Pin, ...

Analog Out Endpoint

- | Set Static

- | Set Sine, Set Square, ...

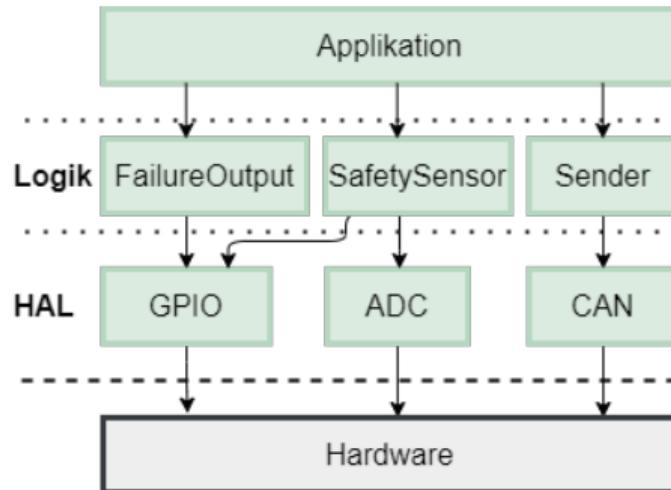
CAN Endpoint

- | Send Frame, Get Received Frames

- | Corrupt Next Received Frame, ...

Strategie für Integrationstests

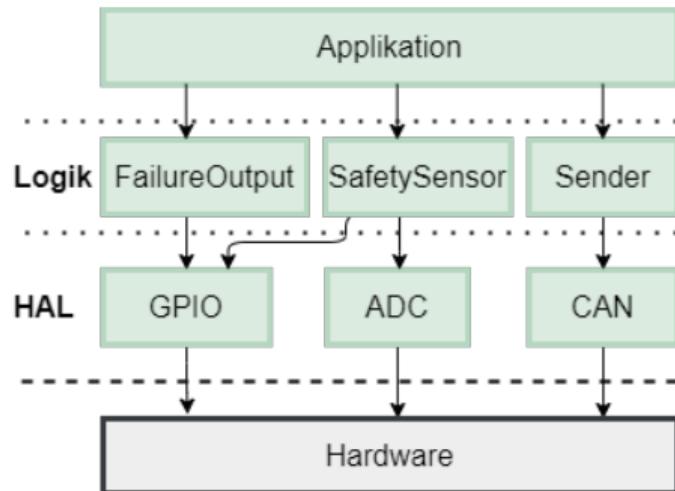
1. Fokus auf HAL-Ebene



- | HAL per HW-SW Integration Test prüfen.
- | Alles darüber per Unitest und HAL-Doubles prüfen.

Strategie für Integrationstests

1. Fokus auf HAL-Ebene

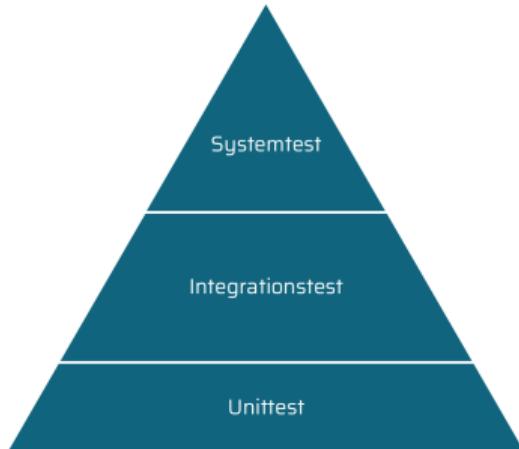


2. Automatisierung und Testtiefe festlegen

- | Annahme *Automatisierung = Mehraufwand* ist im agilen Kontext oft falsch.
- | Integrationstests oft ein Sweetspot zwischen hoher Testtiefe bei geringer Komplexität
- | Eigenaufbau vs. kommerzielles System?
 - ▶ Vielfalt in Peripherien?
 - ▶ Fault Injection notwendig?
 - ▶ Langfristig im Team nutzen?

- | HAL per HW-SW Integration Test prüfen.
- | Alles darüber per Unitest und HAL-Doubles prüfen.

Der Systemtest

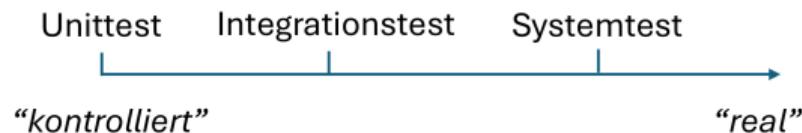


„Eine Teststufe mit dem Schwerpunkt zu verifizieren, dass ein System als Ganzes die spezifizierten Anforderungen erfüllt.“

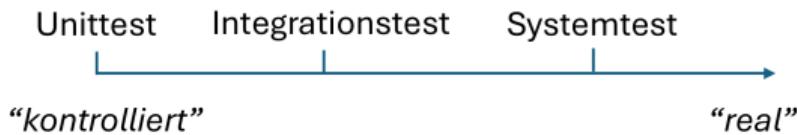
Systemtest

Sicht	Testobjekt
Software	Finale Firmware für Mikrocontroller.
Hardware	Leiterplatte inklusive Mikrocontroller mit Firmware.

Einordnung



Einordnung

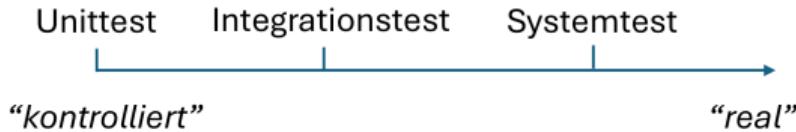


Verifizierung: Baue ich das Produkt richtig?

„Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.“

ISTQB Glossar

Einordnung



Verifizierung: Baue ich das Produkt richtig?

„Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.“

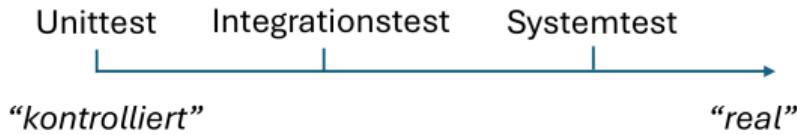
ISTQB Glossar

Validierung: Baue ich das richtige Produkt?

„Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind.“

ISTQB Glossar

Einordnung



Verifizierung: Baue ich das Produkt richtig?

„Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.“
ISTQB Glossar

Validierung: Baue ich das richtige Produkt?

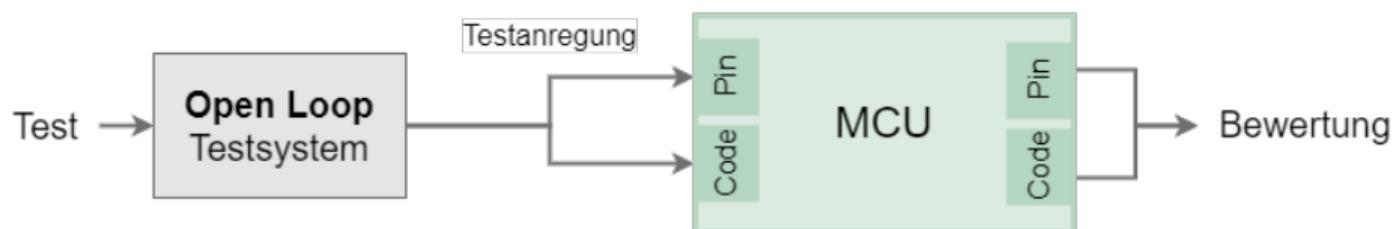
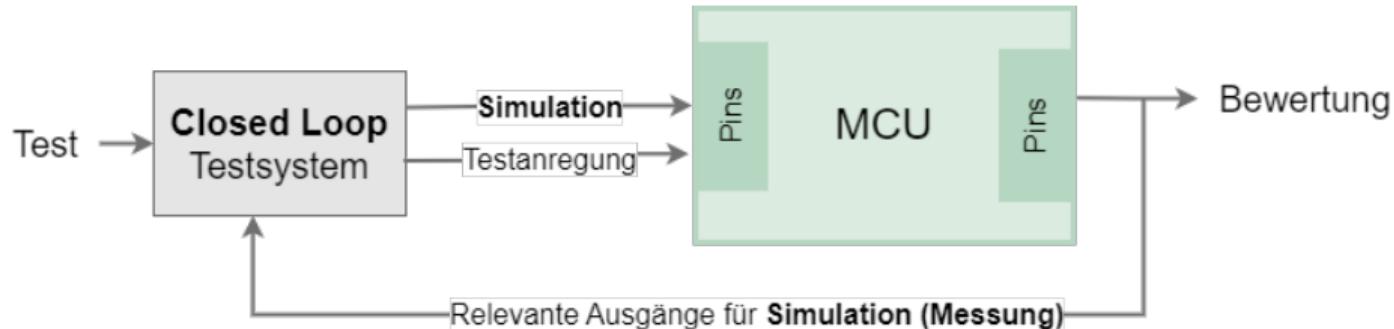
„Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind.“

ISTQB Glossar



Testsysteme und In-The-Loop Analogie zur Regelungstechnik

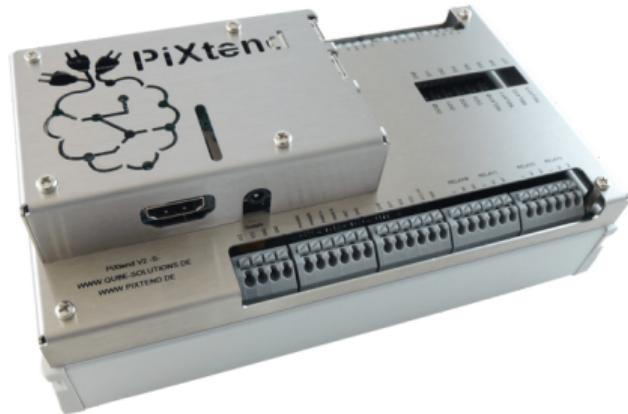
- | Closed-Loop: Ein Regler beeinflusst einen Prozess auf Grundlage einer kontinuierlichen Messung.
- | Open-Loop: Eine Steuerung beeinflusst einen Prozess ohne die Wirkung zu messen.



💡 Unitests und HW-SW Integrationstests haben Open-Loop Charakter.

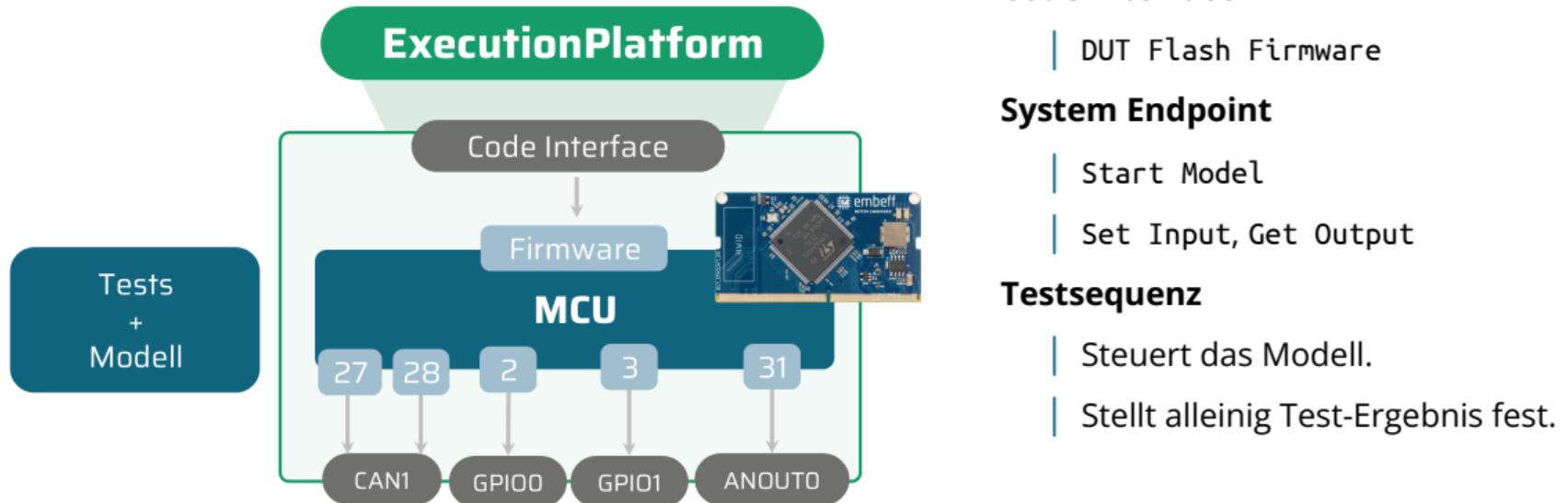
Lösung 1: Eigenbau

- | Empfehlung: RobotFramework als Testsprache
- | Beschränkung auf realistischen Testaufbau
- | Externe Geräte / Simulationen nötig für Test?



PiXtend als Testgerät

Lösung 2: Ausblick kommerzielle Systemtests auf Pin-Ebene



💡 Modell: Nachbildung der Umgebung als *digitaler Zwilling* auf Pin-Ebene.

Strategie für Systemtests

1. Agiles Testen heißt entwicklungsbegleitendes Testen

- | Testentwicklung im Sprint nötig!
- | Testsystem-Nutzung von Entwicklern und Testern.
- | Fehlende Endhardware darf Tests nicht blockieren.

Strategie für Systemtests

1. Agiles Testen heißt entwicklungsbegleitendes Testen

- | Testentwicklung im Sprint nötig!
- | Testsystem-Nutzung von Entwicklern und Testern.
- | Fehlende Endhardware darf Tests nicht blockieren.

2. Systemtests haben die größte Komplexität

- | Nötige Simulation der Umgebung steigert Komplexität.
- | Test-Debugging als Blocker (warum schlägt der Test fehl?)



NI / TestStand

Strategie für Systemtests

1. Agiles Testen heißt entwicklungsbegleitendes Testen

- | Testentwicklung im Sprint nötig!
- | Testsystem-Nutzung von Entwicklern und Testern.
- | Fehlende Endhardware darf Tests nicht blockieren.

2. Systemtests haben die größte Komplexität

- | Nötige Simulation der Umgebung steigert Komplexität.
- | Test-Debugging als Blocker (warum schlägt der Test fehl?)

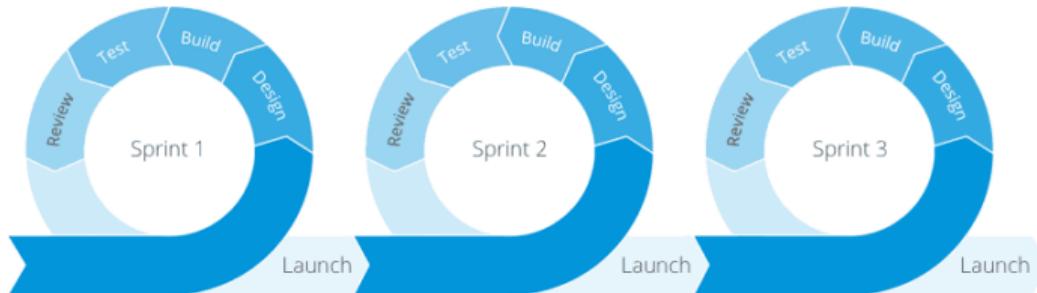
3. Im Sprint auf Sweet-Spot beschränken

- | Vollständige Tests (Zulassungen, ...) praktisch oft nicht im Sprint möglich.
- | Klare Trennung zu agilen und traditionellen "Nachweis"-Tests.



NI / TestStand

Agile Entwicklung für Mikrocontroller ist möglich!



Folien: <https://github.com/embff/talks>

- | Fokus auf Tests, die laufend nötiges Feedback liefern.
- | Langfristige Wartbarkeit von Anfang an durchdenken.
- | Agile Entwicklung macht Spaß!

Direkter Kontakt
daniel.penning@embff.com
Phone +49 (451) 16088698

